Training AI to Classify Camera Data

Heather Gaya

2024-09-09

Overview

Training an AI model is becoming increasingly simple thanks to modern technology and the availability of openAI. Unfortunately, a lot of the resources available are written for engineers and data scientists - rarely are they written in a way that an average ecologist with an annoying backlog of camera data can easily understand. Let's fix that.

This tutorial assumes you have:

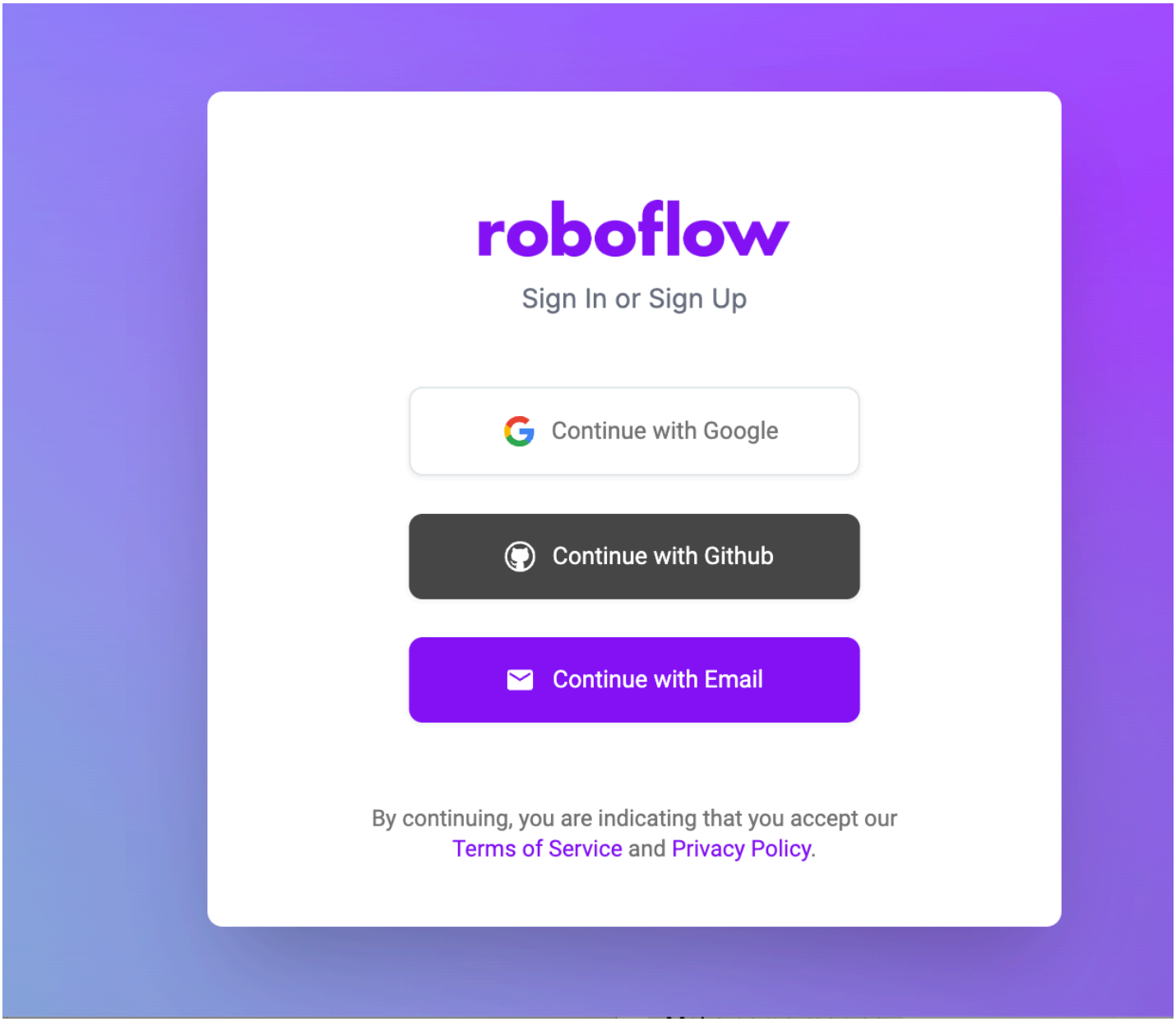
- an email account that you can link to Roboflow
- a bunch of photos that need classifying
- a pre-defined number of objects/animals/classes that you want to tag in photos
- access to Python
- Time

Part 1 - Roboflow

Step 1 - Setup a Roboflow Account

First step, we need a program that lets us upload photos and draw boxes/outlines around objects of interest inside the photos. There are several programs that will do this, Roboflow is just the easiest that I've found. One of my favorite aspects is that you can have multiple users doing the tagging, so the creation of your training dataset can be a speedy process (In theory. I did all my tagging solo and therefore have no idea if multiple people are helpful or annoying. But I like the concept!)

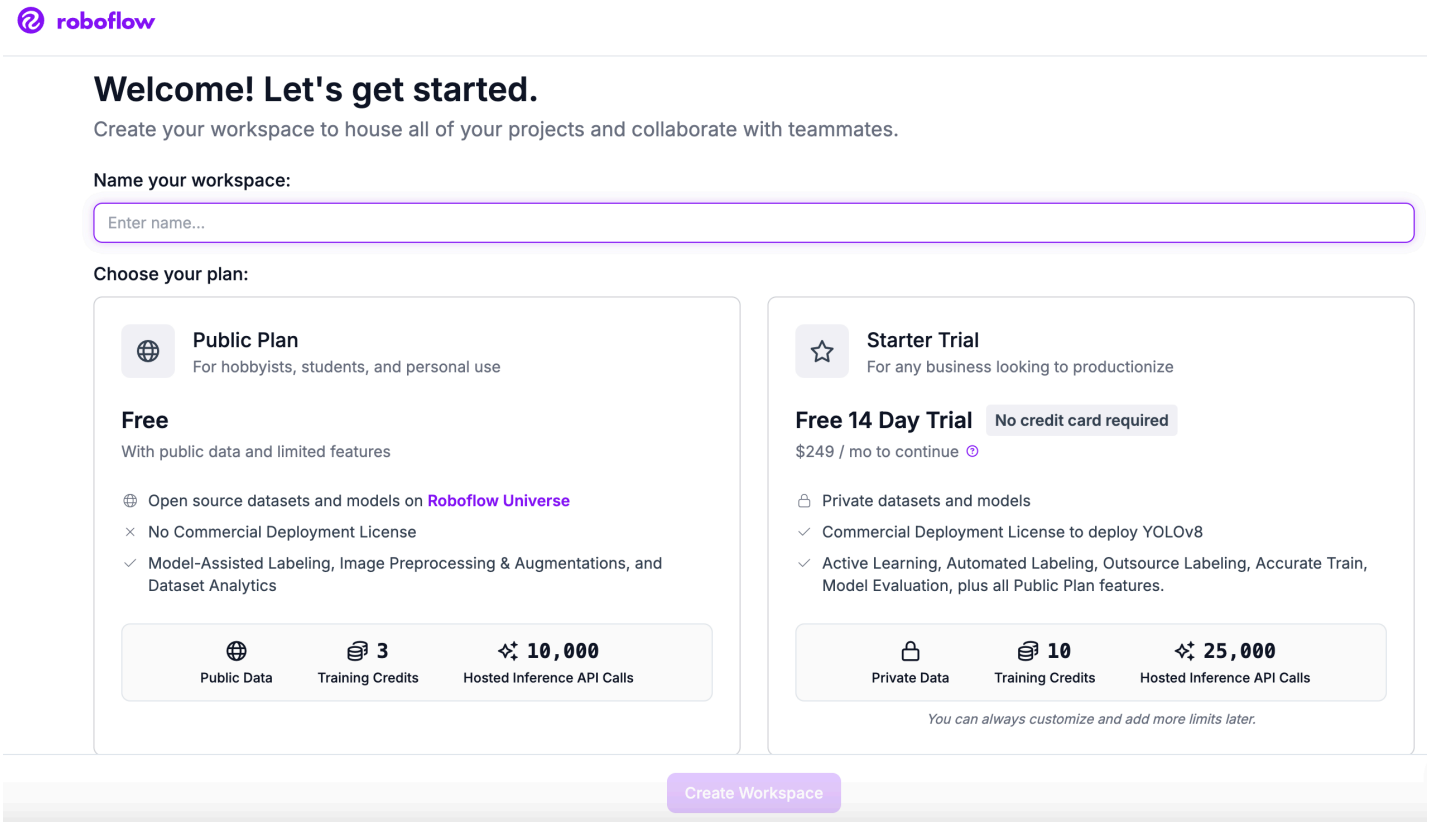
Roboflow can be found here: https://app.roboflow.com/login (https://app.roboflow.com/login)



Step 2 - Create a Project

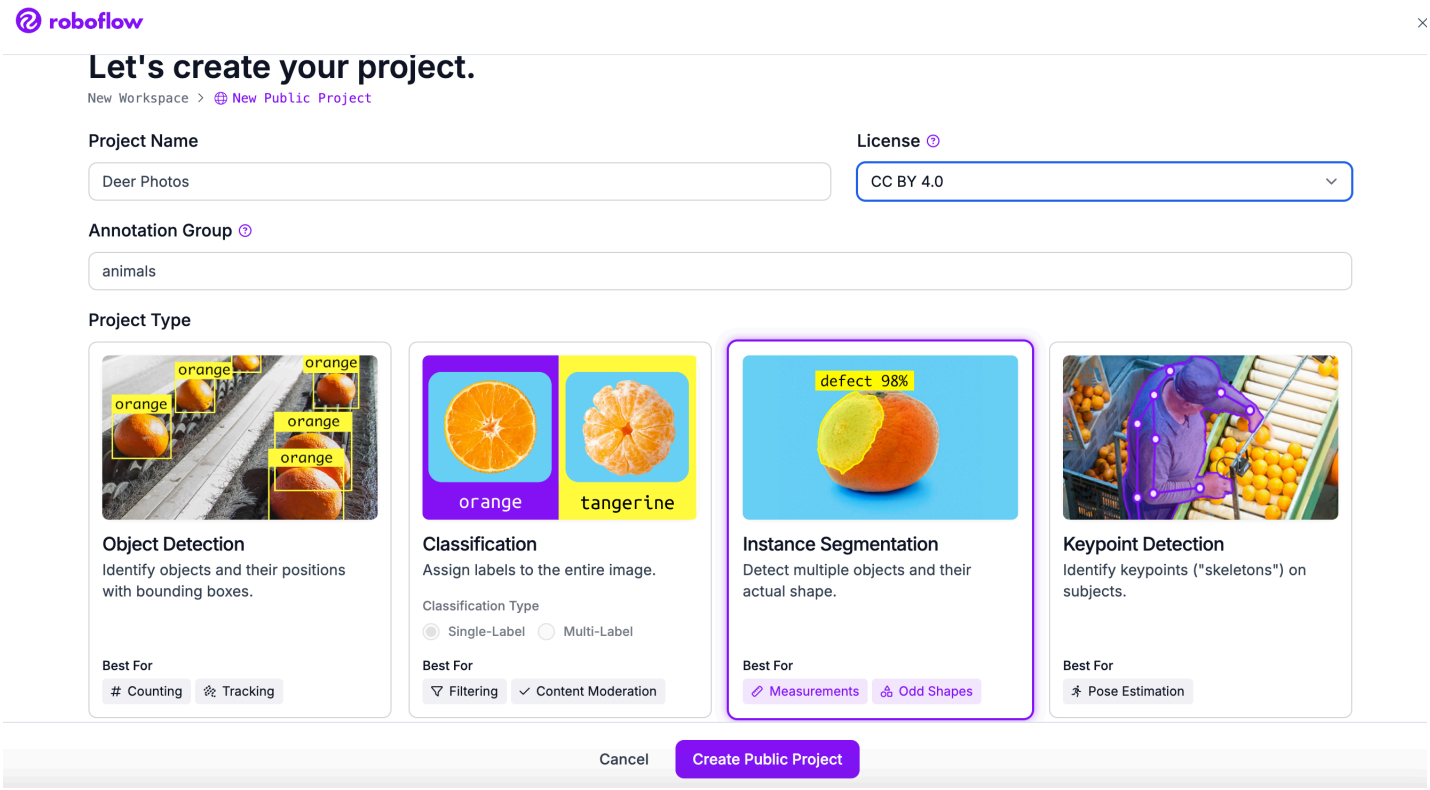
First Roboflow will ask you to name your workspace. Do so. I named mine “Deer AK” because I forgot that AK is Alaska and not Arkansas (where my project is based). Luckily this has no bearing on anything, except if you misspell something you will see it for the rest of eternity and it WILL bother you.

You have the option of the public plan or the starter trial. If you’re just starting out and aren’t sure if you want to go down this route, the public plan is a great option. Once you know what you’re doing, the Starter Trial is ideal, because you can train your model 10 times! You can always add the Starter Trial later, so I recommend starting with the public plan.



It will ask if you want to add teammates, but you can skip that step for now.

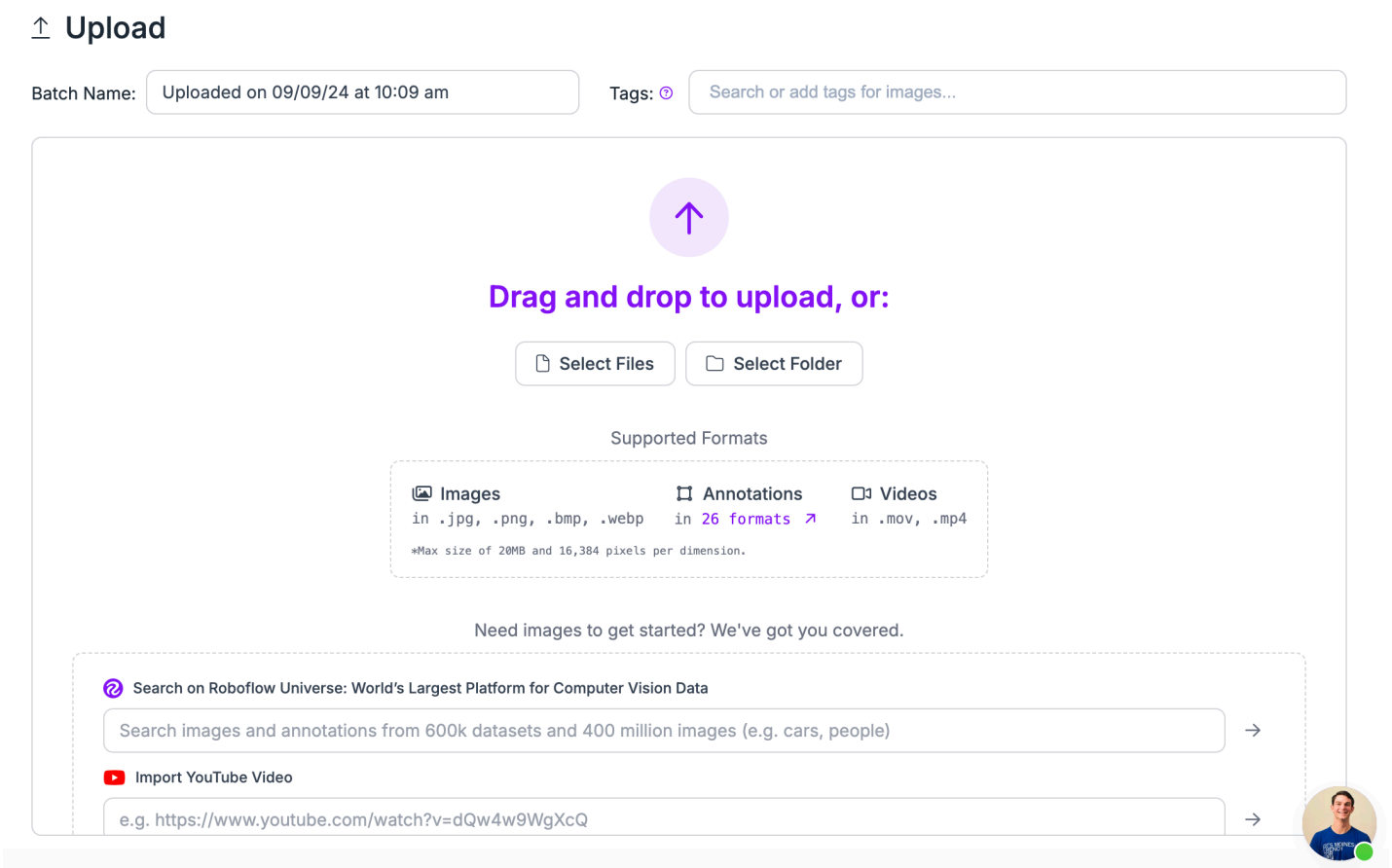
Now we create a project. I have found that Instance Segmentation works slightly better than Object Detection for camera trap photos, but they are both powerful AI models. Choose whichever you like. (Note: If you're planning to use videos and not images, bounding boxes are probably easier)



Step 3 - Upload Data

Pretty simple. Drag and drop baby. I like to tag my image batches by various words in case I want to refer to a specific upload later. You can also use the Roboflow universe to use other people's photos if you don't have enough of your own! If you're not working on a game species… probably not going to be helpful.

Roboflow will then ask you how you want to label your images. I like to start with Auto Label, but be aware this will be VERY inaccurate until you've trained your first model.



How much data do I need?

How much data should you upload? How many photos does it take to train your AI? This depends on how many classes you plan on tagging. Let's say I'm working on feral hogs and all I want to know is "does this photo contain a hog? If so, how many?" In that case, we need one class: "hog." For this type of model, we might only need 3-5K images. Maybe even fewer!

On the other hand, what if I'm doing community trapping and have 25 species? That might be 25 classes! In this case, I might suggest training separate models (aka make multiple projects in Roboflow), with an AI that first separates your pictures into general classes (bird, frog, mammal, human, etc) and then train a second AI that works just on "birds" and classifies down to species. Either way, this type of model will need at least 8-10K photos to learn on, maybe more.

In my own work, I produced an 80% accurate model with 8K images using 7 classes. It still makes errors, but our tagging rate of photos has gone from 200/hour to >3K per hour. Awesome!

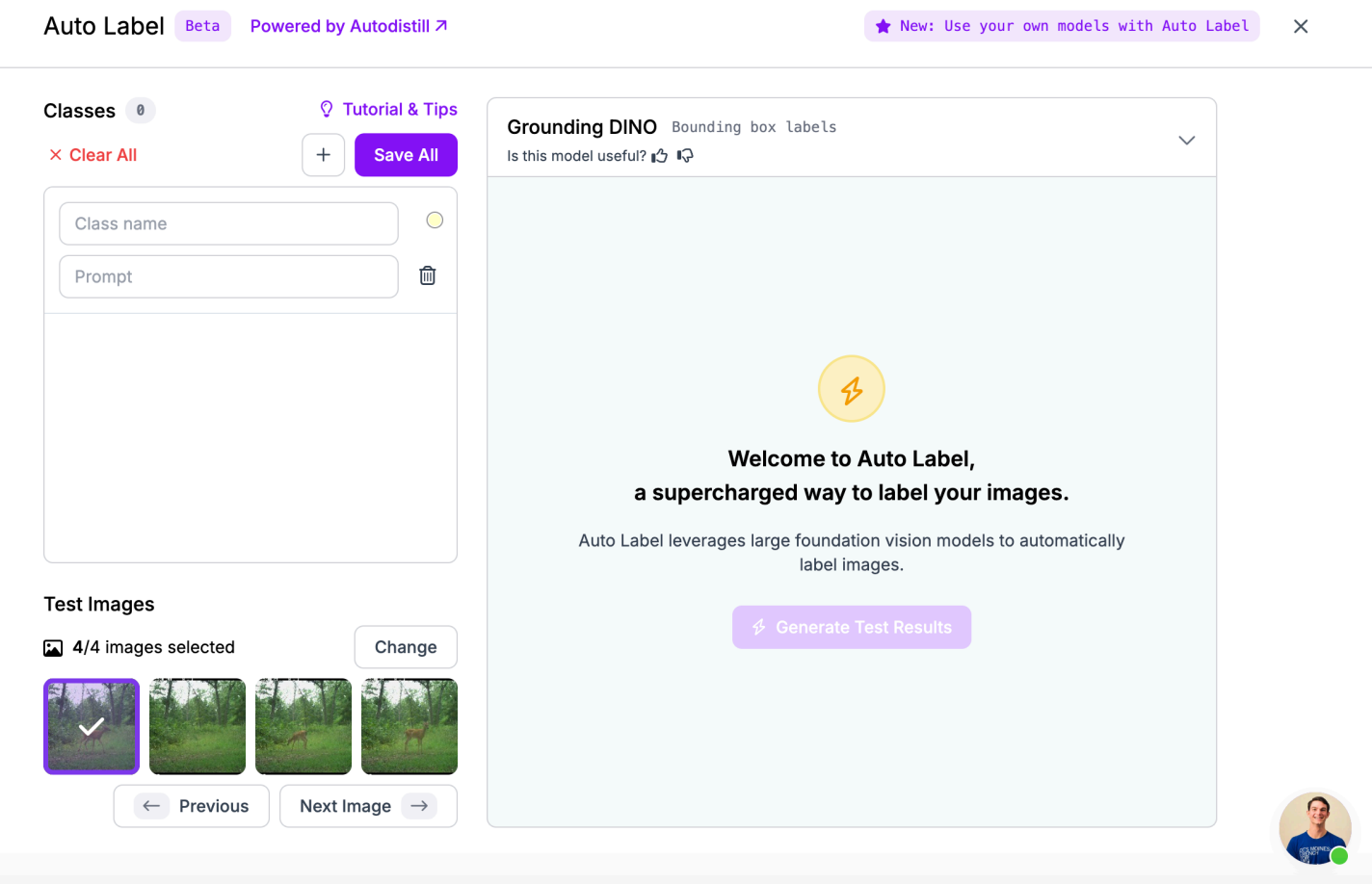
Should I upload it all at once?

Short answer - no. You can and that won't hurt you, but you don't need to. What I find is easiest is to upload a chunk (maybe 1000 photos) and manually tag those photos first. Then train a model (which won't do well probably) and use THAT model to help you tag a big batch of photos (like the next 5K). Then train the model again and use version 2 to tag more photos… etc. It will get better overtime and save you a lot of headache.

Step 4 - Data Classes

When you first start labeling, you'll need to define classes. If you choose auto-label and haven't fit a model yet, you'll be asked to provide both a Class Name and a Prompt.

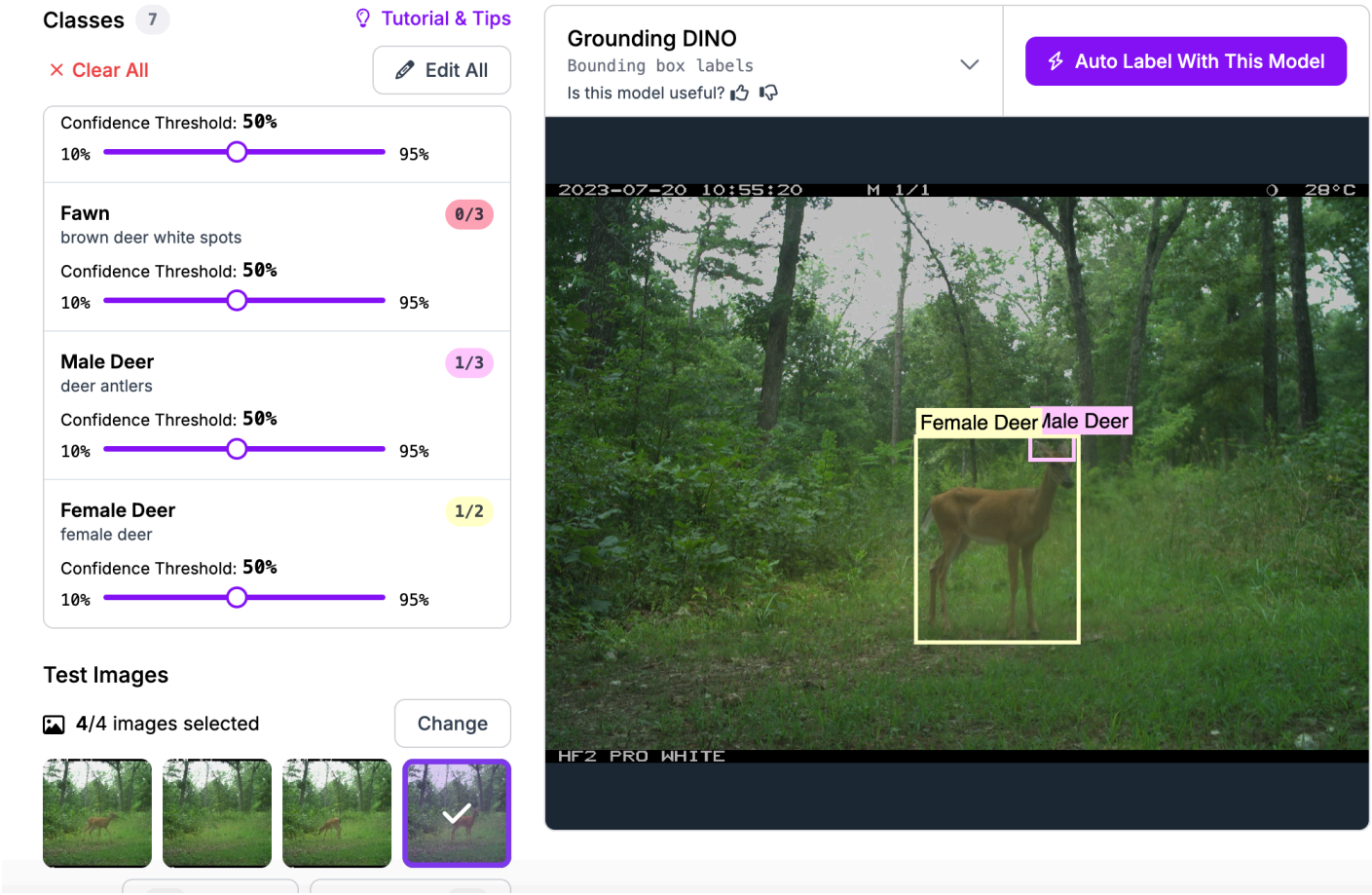
The class name can be whatever you want, but the prompt should be “keywords” that the auto-labeler can use to see if it can use the pre-trained COCO library to find the class in your image.



I’m tagging deer, so my classes are deer specific but here’s what my class/prompts look like:

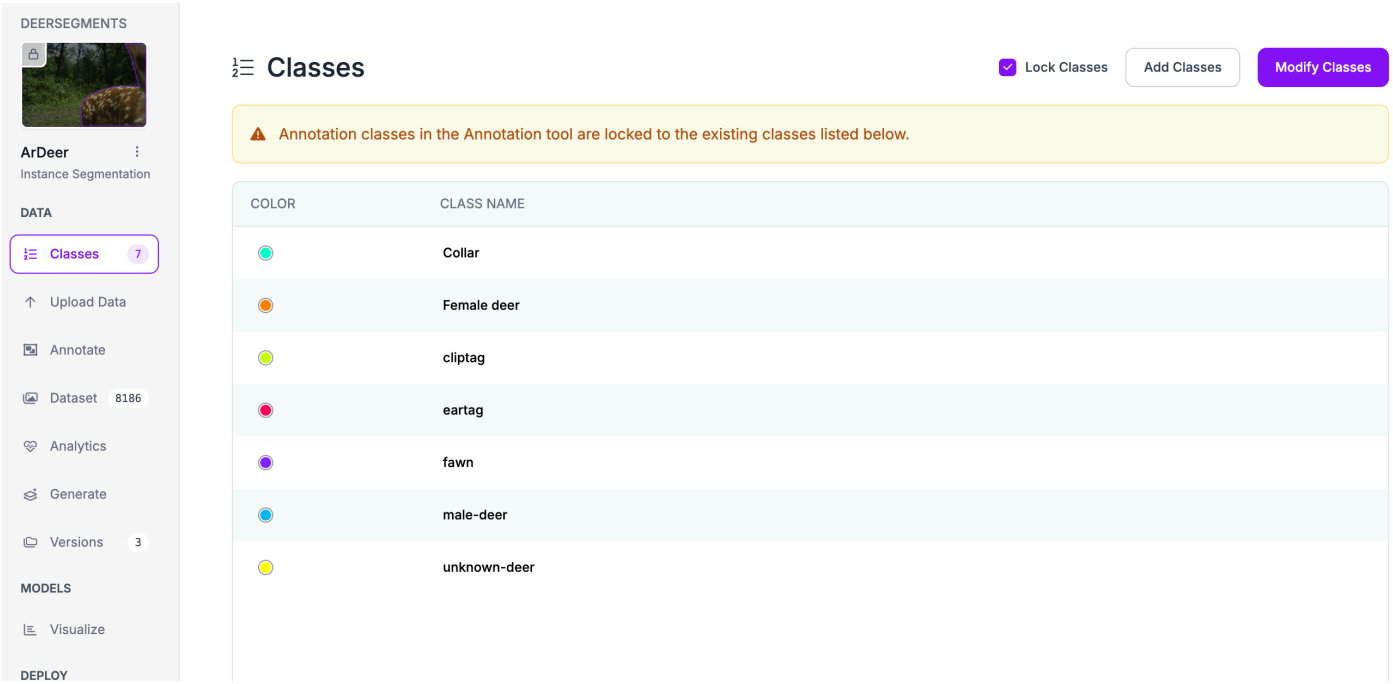
- Class: Female Deer; Prompt: female deer
- Class: Male Deer; Prompt: deer antlers
- Class: Fawn; Prompt: brown deer white spots
- Class: Unknown deer; Prompt: deer no head
- Class: Collar; Prompt: collar on neck
- Class: Eartag; Prompt: numbers
- Class: Earclip; Prompt: colors

As previously mentioned, it will not do a great job at first:

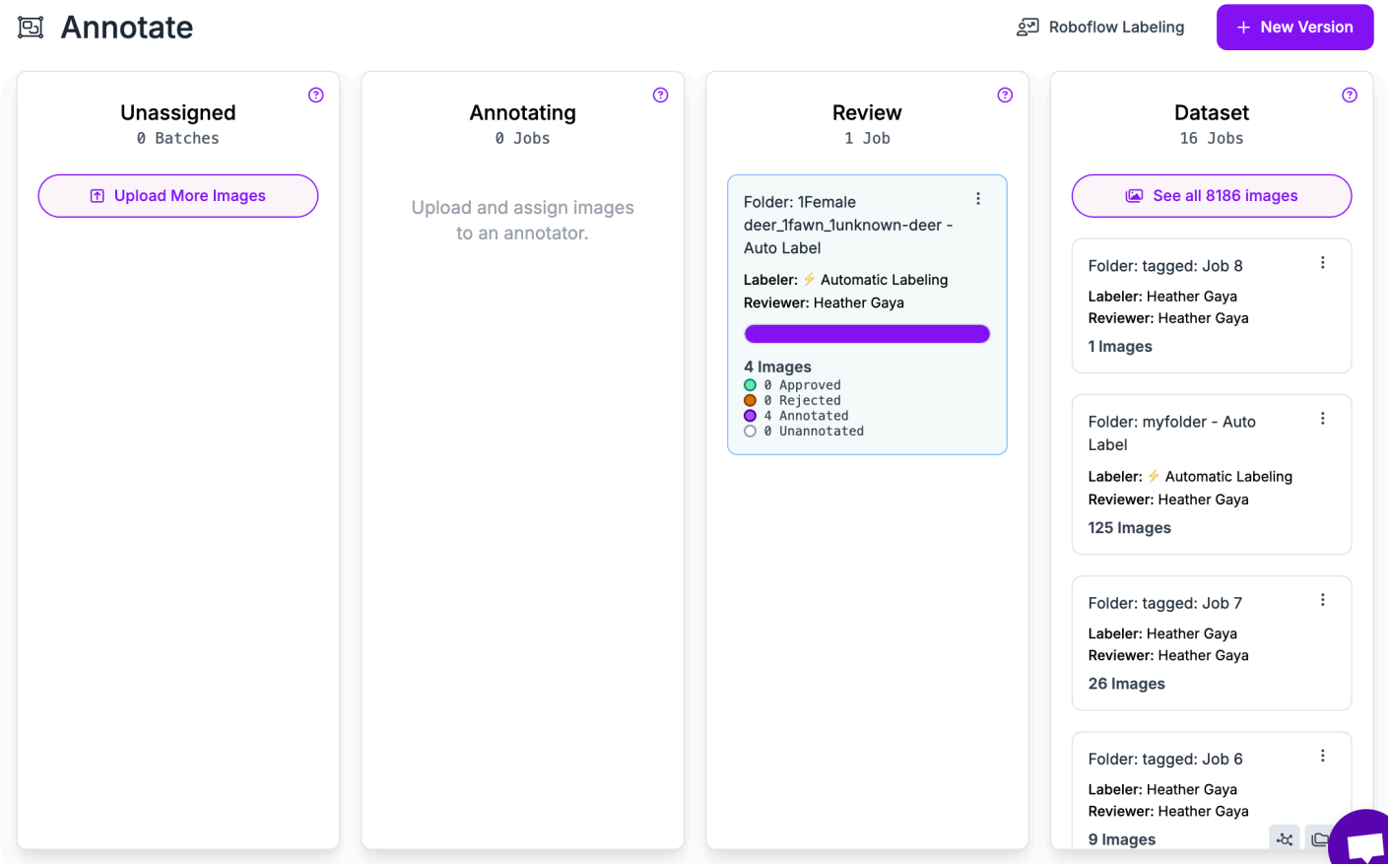


Step 5 - Annotating

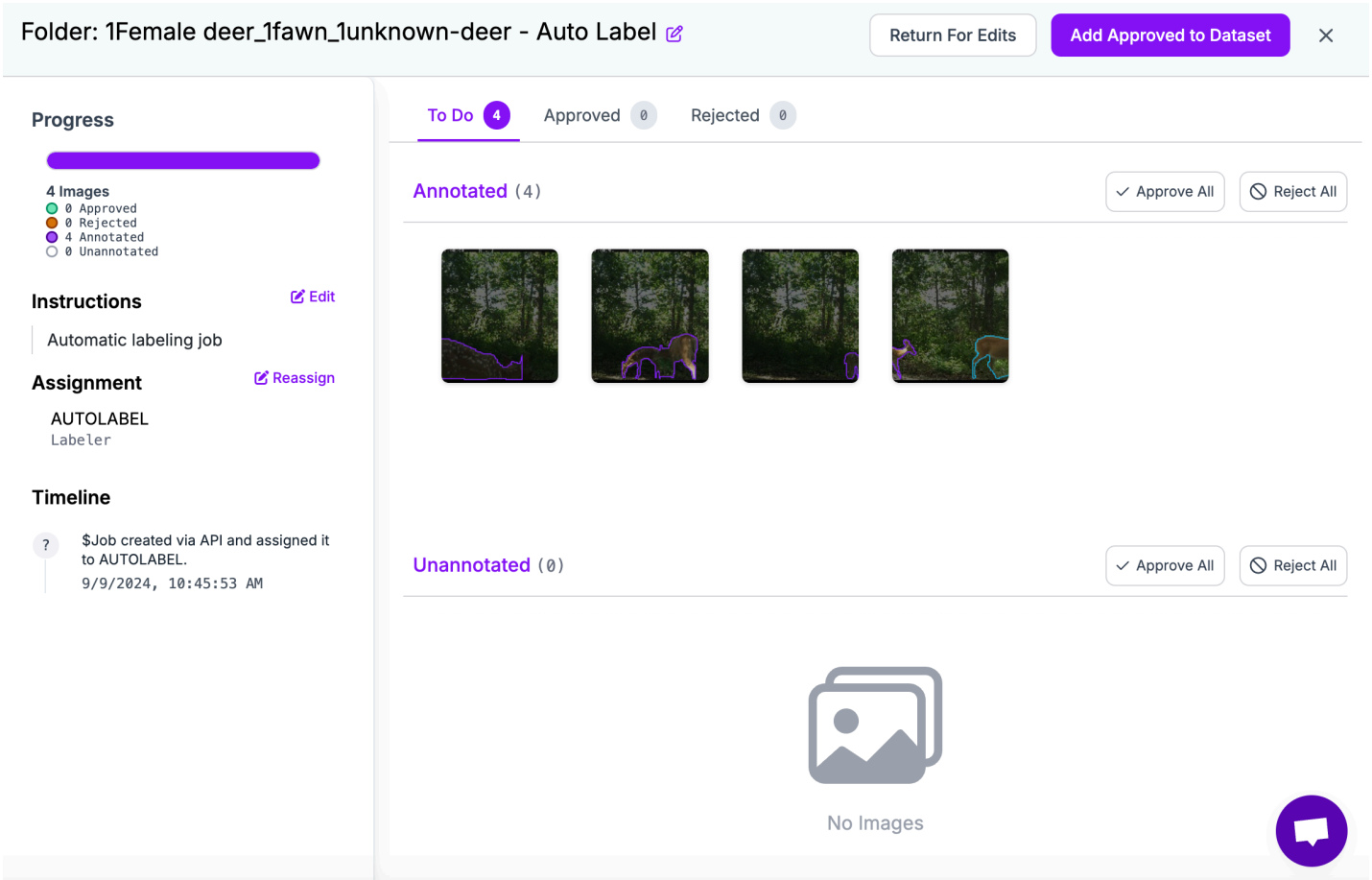
Now the fun begins. If you let the auto-labeler work, it'll take a few minutes to get through your photos. In the meantime, go to Classes (on the left menu bar) and hit "lock classes" so that you don't add spurious annotations on accident



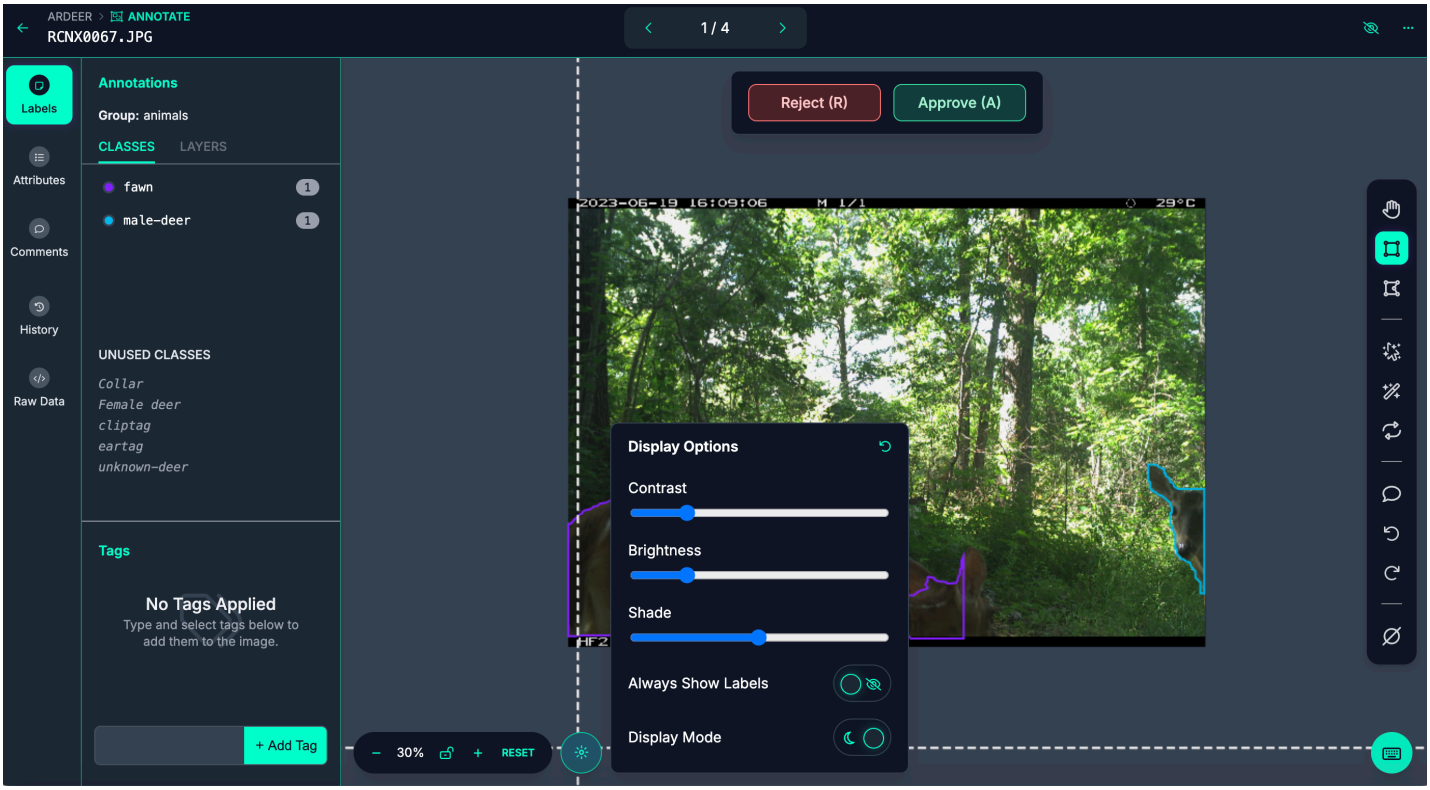
Once the images are done being auto-labeled, they'll show up in the annotate tab under the Review header. Click the job to see the details.



You'll see that photos you haven't looked at yet are in the "To Do" section and photos that are approved and ready to be processed by the model are labeled "Approved." Click on an image to see what the auto-labeler did:



For my first photo, we can see that the AI thinks there's a fawn and a male deer in the photo (it's a fawn and a female deer, but good try!)



Click on the light button on the bottom to adjust the display for easier tagging. On the right, you'll see the different options for tagging. The hand symbol is for dragging around the image. The square symbol is for rectangular bounding boxes. The button with the mouse picture is for 'smart labeling' which lets you click on an object and

it'll try to find the bounds of it. Click on outlines in the photo to change the labels or remove the label altogether.

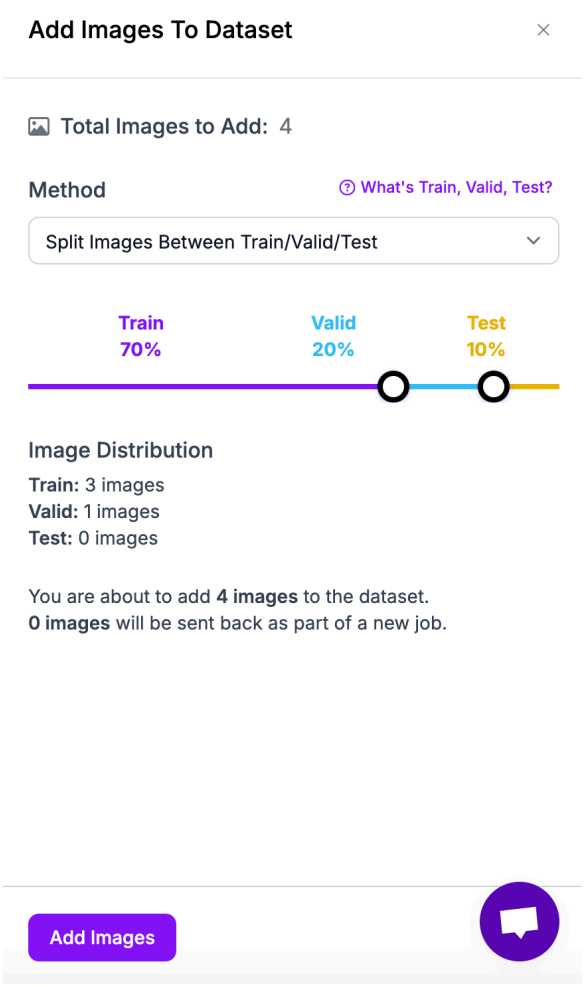
Once you're satisfied with the image, you can hit "approve" (or type "A") and it'll take you to the next photo for labeling. I don't recommend rejecting photos unless you sign up for a membership to Roboflow. In an ideal world, you would constantly be re-training your AI, telling it "you did this wrong, you did this right" and it would correct itself over time. However, with the free trials on Roboflow you only get a limited number of times you can train the model, so I wouldn't bother with this feature.

When you're done for the day or done with the dataset, you can exit out of the annotating screen.

Step 6 - Annotating

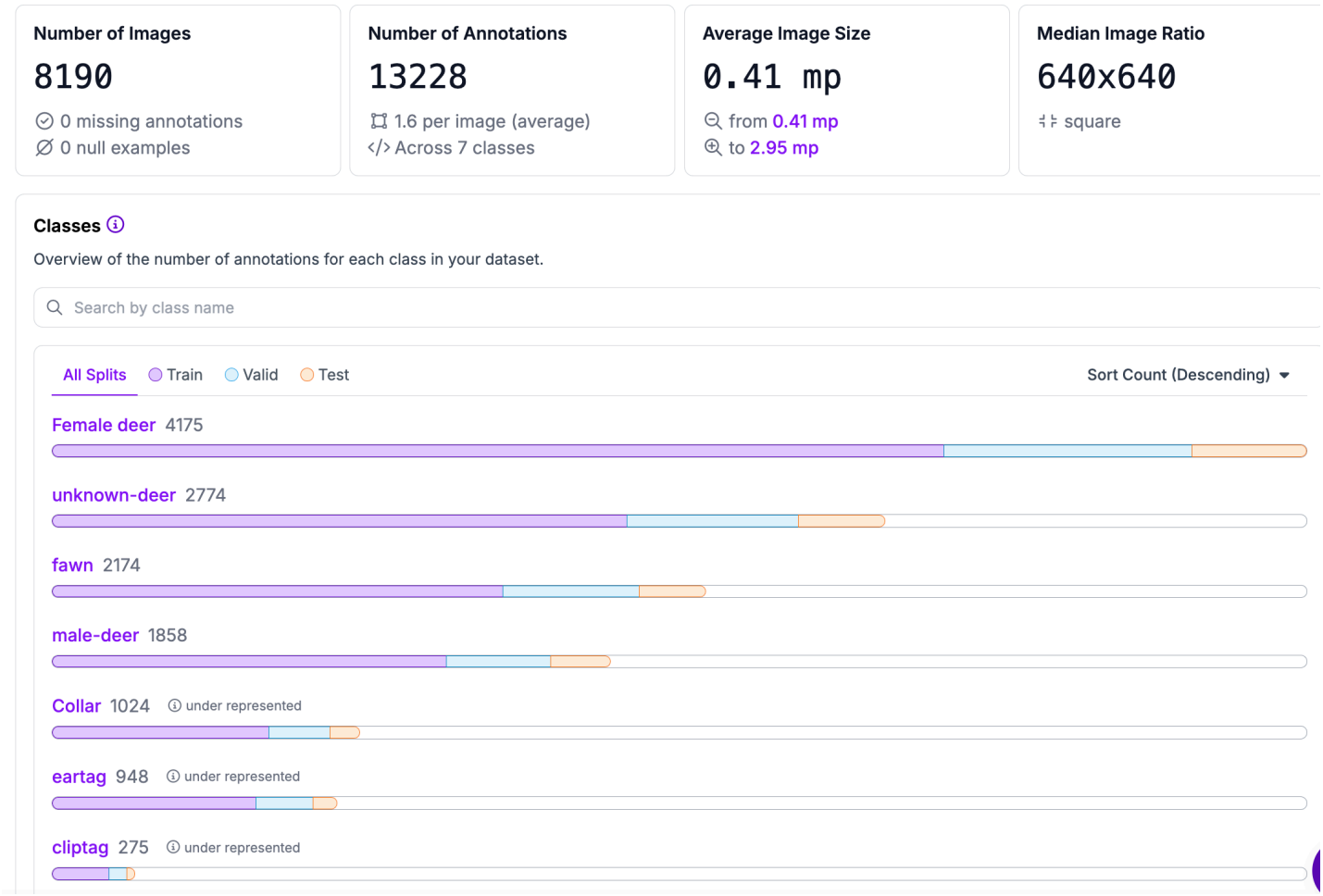
Once all your photos in a set are approved, click "add approved to dataset." You'll see a screen asking how to split images between the train/valid/test datasets. General guidelines suggest somewhere between an 80/10/10 and a 70/20/10 split is going to give you your best performance. Here's what those terms mean:

- Train: This is the data the AI will train on. Remember that AI is just a big ol' regression in the background, but at the pixel level. Think of each pixel as having a lot of attributes such as rbg and contrast, plus the level of difference between that pixel and the pixels near it, the pixels kind of close, the pixels far away… etc. In the background, the AI is literally just doing a regression on all that info. The more data it has, the better.
- Valid: Unlike normal regression, where you just choose a bunch of variables and let math do its thing, we don't know which variables we want to use! So at each iteration of the regression (called "epochs" in AI lingo), we check to see which variables were useful by using the current model to predict results on a subset of photos. These are the "valid" photos. If the prediction doesn't work very well, it will change which variables to use and try again. Note that these "valid" photos are still used during training.
- Test: Once the model is trained, we try to predict the results for photos it hasn't seen at all! These photos give you your stats that you can use to estimate how accurate the model is overall.



Step 7 - Check Spread of Tags

After you've tagged a solid portion of your photos, it's a good idea to check how your classes are spread out throughout your dataset. Click on the Analytics tab. Here you'll see a lot of information about your dataset.



We can see that I have 8190 images in my dataset, with 0 of those images missing annotations or being null examples. In my case, we have 244 cameras taking photos, so it isn't very helpful to have images of just the background with no animals in it. If you only have a few cameras, it may be useful to train the AI on what a "no detection" image looks like, to minimize false positives from leaves or interesting shaped trees in the background.

We can also see that I have a LOT of female deer in the dataset, with fewer fawns and male deer. Only 275 of the images have cliptags in them, which Roboflow suggests is too few for the AI to accurately pick up. For best results, I should increase the number of photos with cliptags, collars, and ear tags. (Note that Roboflow doesn't know that I CAN'T have just a photo of a cliptag, it must always have a deer with it. So this ratio looks worse than it is).

Side Note About Bias

You ideally want your dataset to be representative of all the data it will ever be used on. This is a lofty goal, and it may not be possible to achieve. But the more similar your training set is to your overall study, the better your AI will perform. This is not just limited to the number of photos in which tags appear.

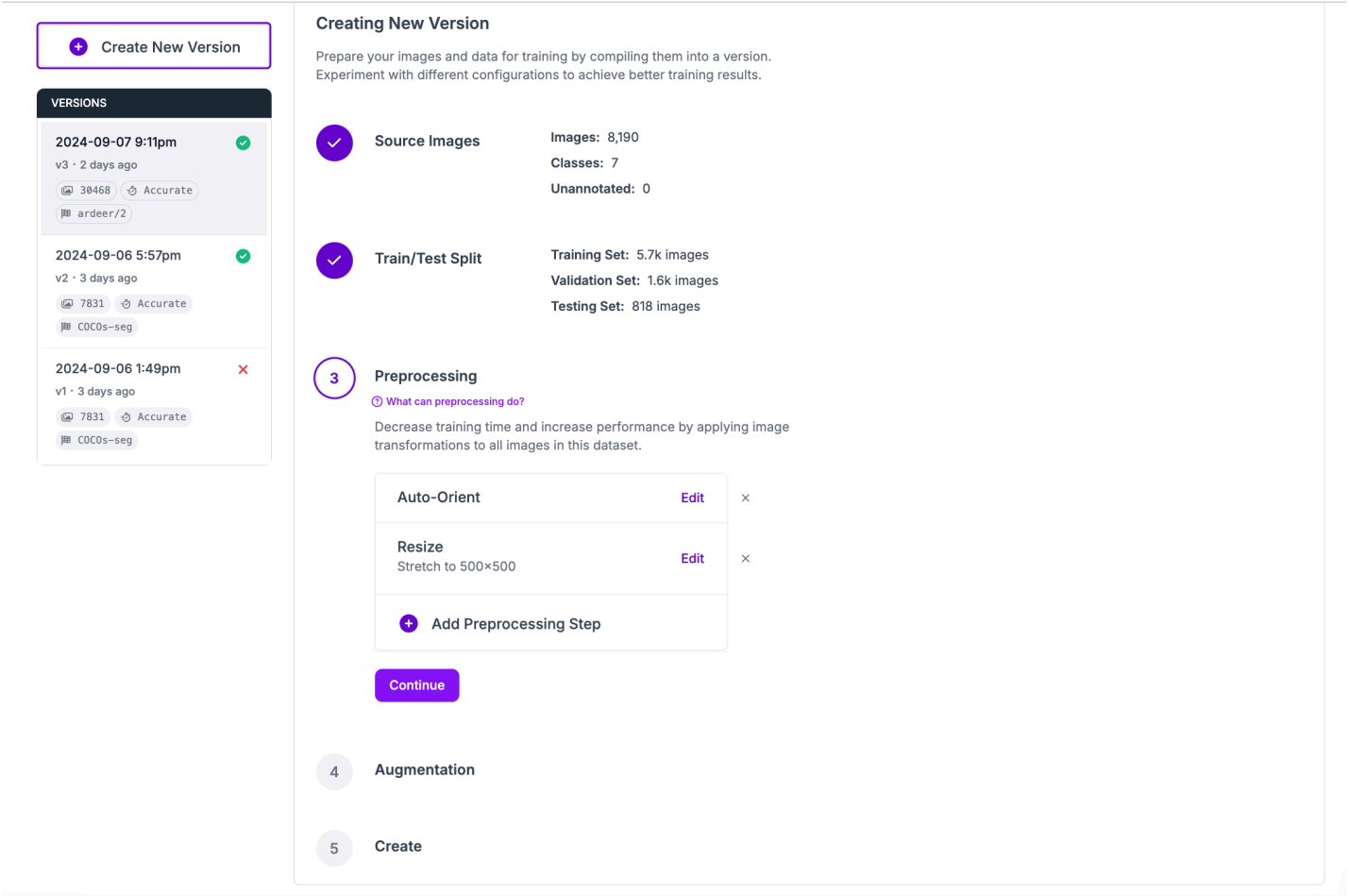
For instance, if you have a lot of photos with just one bird in it, but in your study you frequently see 3 or 4 birds in the same image, your AI will not be 'used' to seeing multiple birds and will not tag some of those birds. So the NUMBER of objects in each photo is also important.

Additionally, if you only train the model on male deer with eartags, it will likely give you false positive for eartags if you upload a photo of a male without the eartag. A similar thing will happen if all your training photos of otters only have otters in the bottom left corner of the image - if given a photo of an otter in the top right, the AI will likely not notice it, because it's been trained to only look in the bottom left for otters.

All that to say, make sure your training data is as varied as possible!

Step 8 - Create a Version

Once you have enough photos, it's time to create a version to train the model on! Go to "Generate" in the side menu and click "Create New Version."

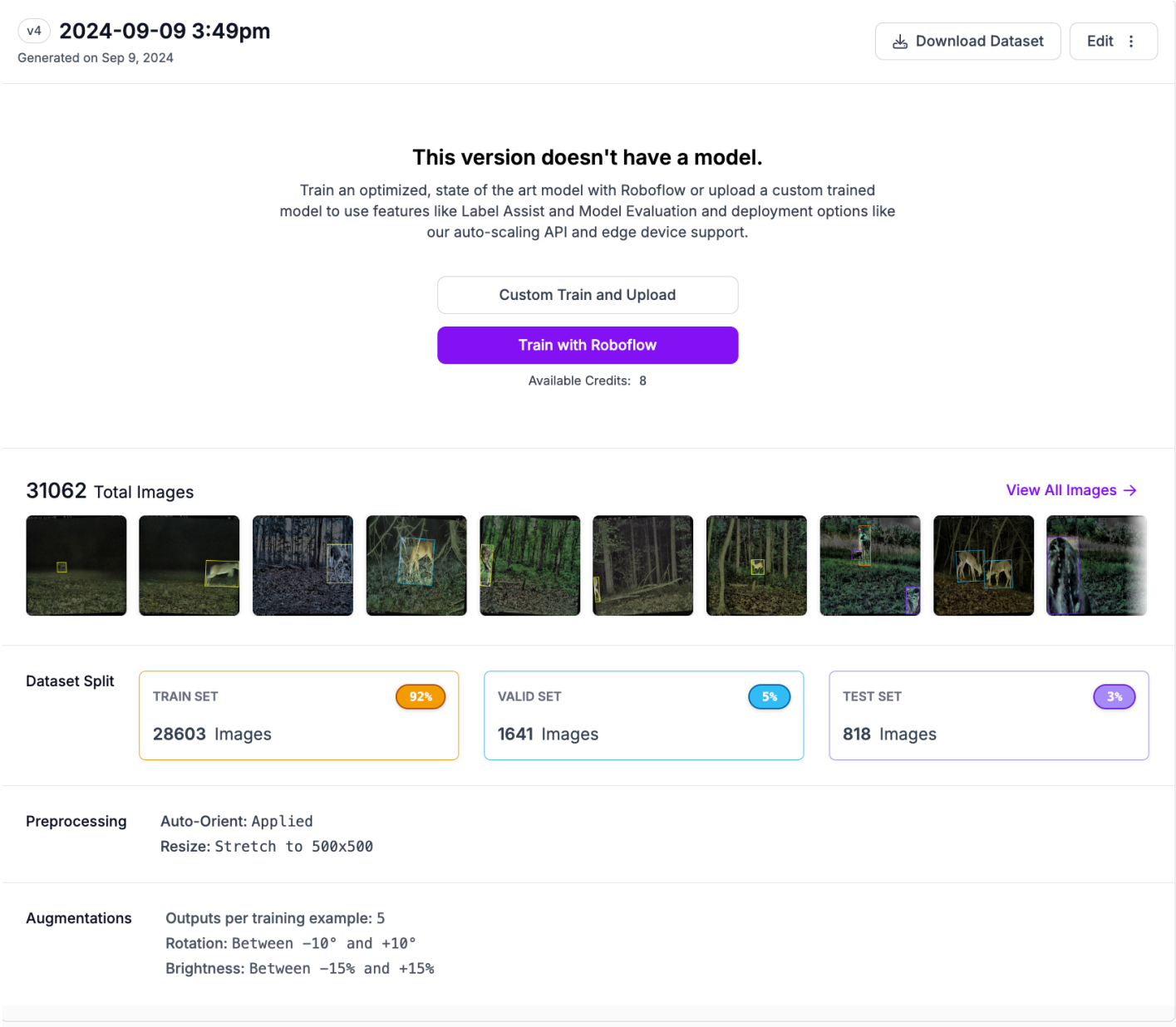


Here you can decide if you want to change the split of your data, only include photos that are tagged a certain way, or do any preprocessing. I found pre-processing to be helpful when I had classes with obvious edges (so fawns/male deer/female deer/unknown) but less helpful once I started have things like tags be included as classes. This is something you may have to play with for your own data.

Augmentation is similarly choose-your-own-adventure. I add in both Rotation (-10, + 10) and Brightness (-15, +15) but exposure or blur may be helpful depending on your camera settings. These steps take the original photos, adjust them and then add them to the dataset in addition to the original photos. The benefit is that the AI is trained on extra photos without any effort, but if your dataset is heavily biased, this will bias it even more.

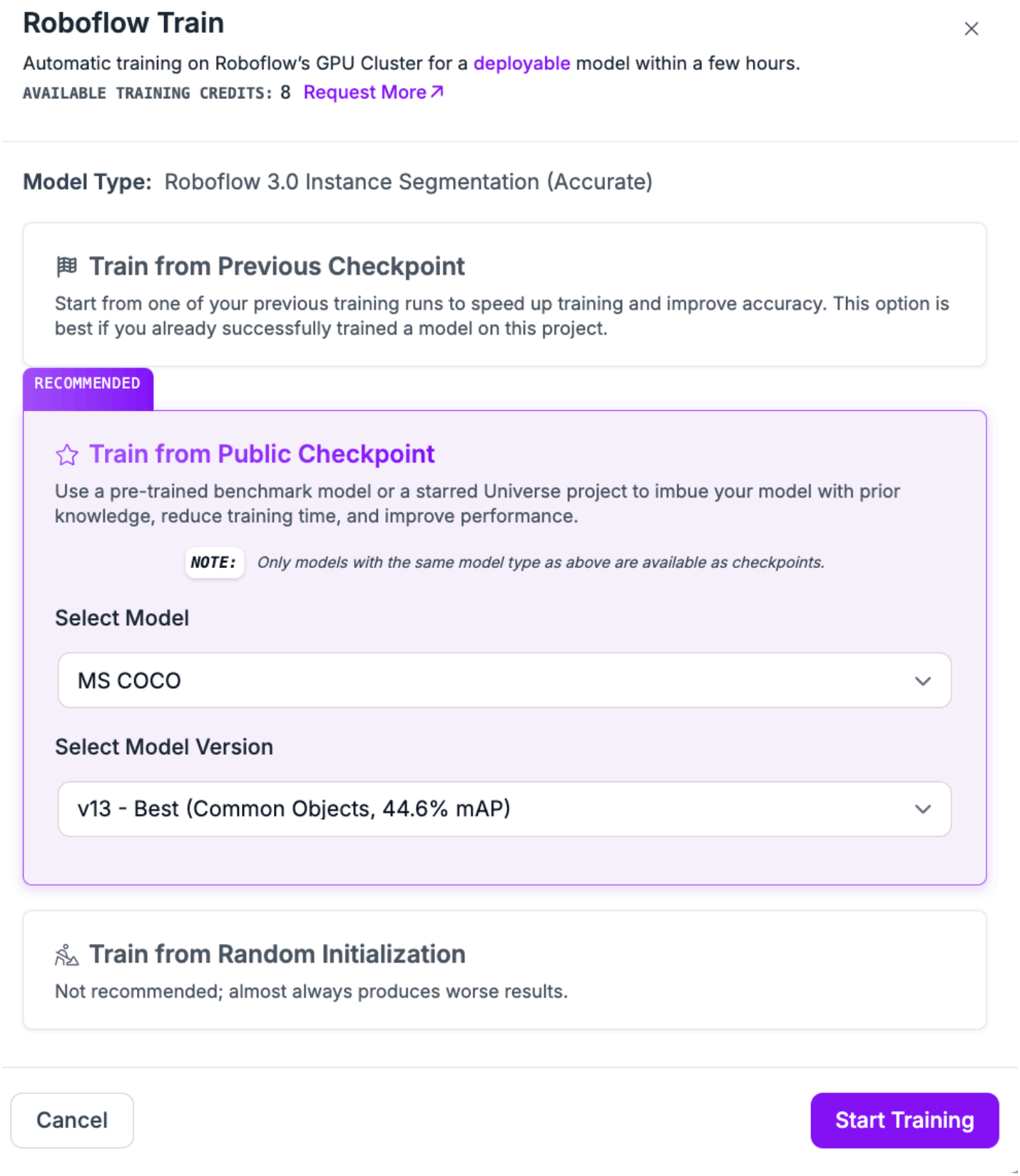
Step 9 - Train a Model!

Wow so much work to get to this step! Great job. Now we can train our model



If you're fancy or you know someone who has a model already built, you can use the "custom train and upload" feature. But for the rest of us, we'll use the Train with Roboflow option.

Choose the most accurate option available with your version of Roboflow. It will take a long time (sometimes more than 1 day), but it will be worth it compared to hand tagging all your photos forever.

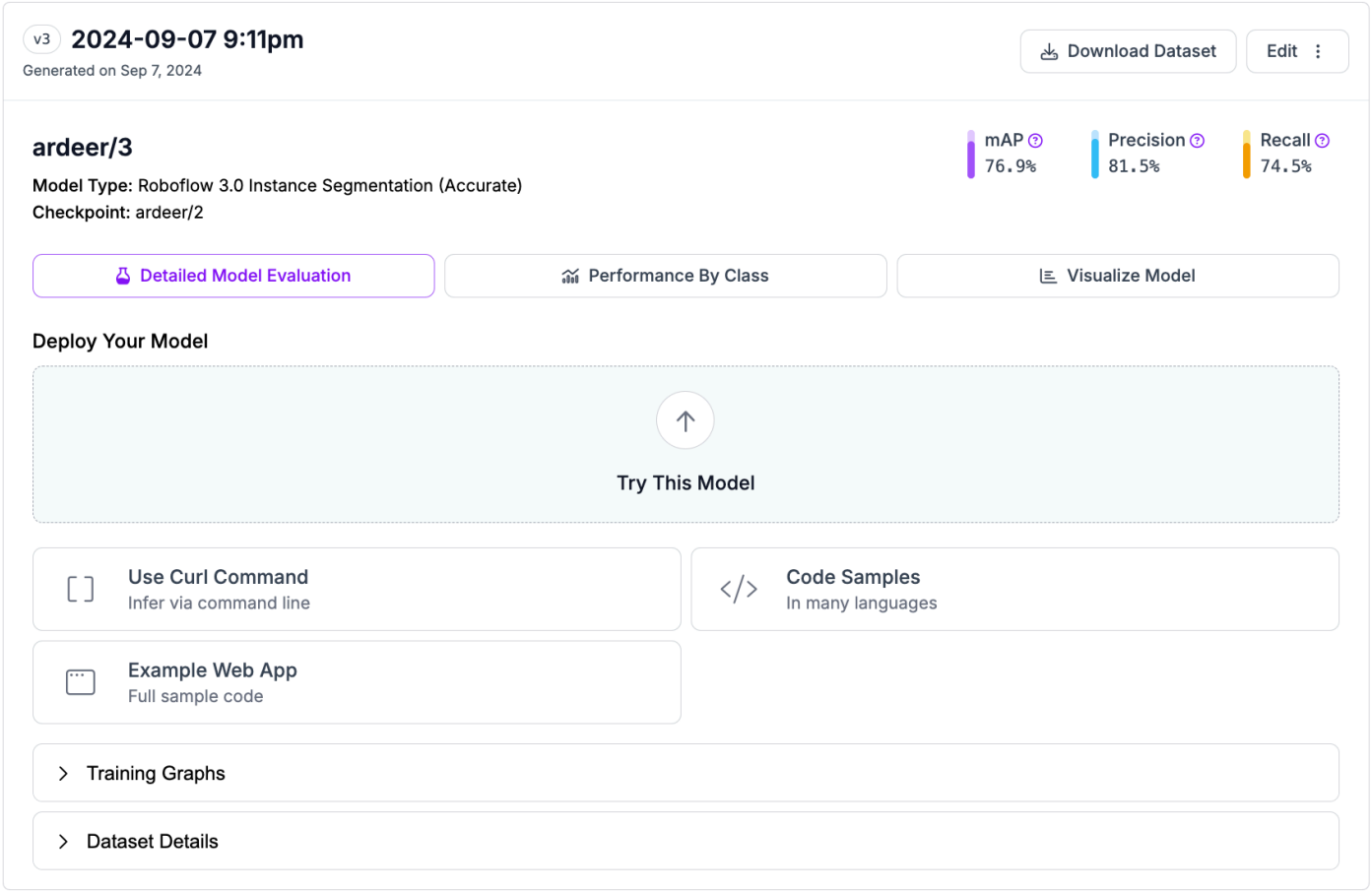


If this is your first model, you'll want to train from the Public Checkpoint. There's not much to do here, just click "public checkpoint" and then "start training!"

If you're doing this step after training your first model, you can save time by training from a previous checkpoint - just select the "Train from Previous Checkpoint" option. Now sit back, relax, and let your model start learning (or go tag more photos, you'll need more eventually anyway).

Step 10 - Evaluate the Model

Once the model finishes, it's time to see how it did.

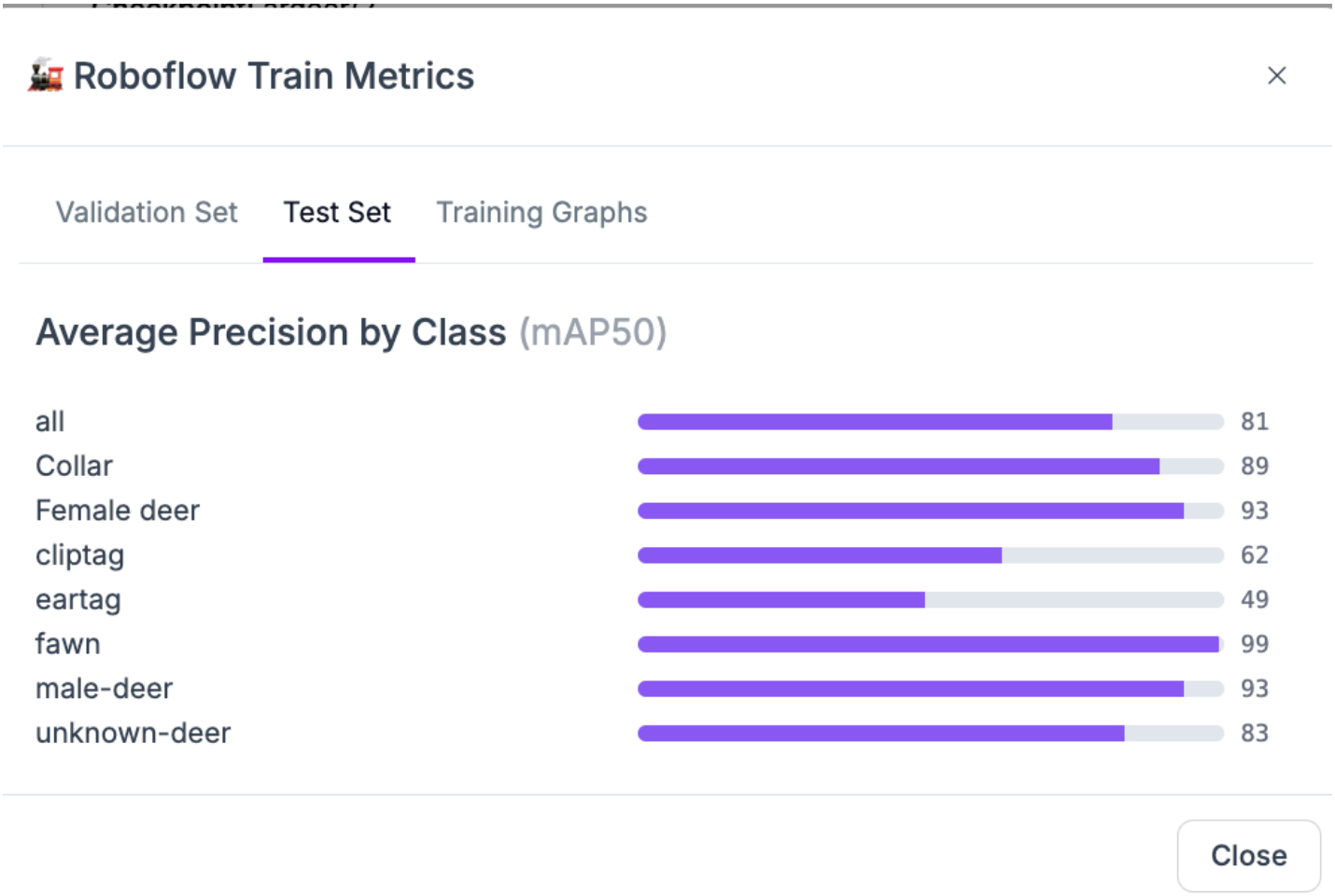


For all models, you'll see 3 scores: mAP, Precision and Recall:

- mAP: This can be thought of as "average accuracy" across all classes in your model across all confidence levels. In this example, my model has 76.9% mAP. Not too shabby, but not perfect.
- Precision: This number is how often your model was correct given that it identified an object. Note that missing objects does not affect precision. Precision should be prioritized if false positives are undesirable. The formula for precision is True Positive divided by the sum of True Positive and False Positive (P = TP / (TP + FP).
- Recall: This number is how often the model identified SOMETHING was present, regardless of what it said that something was. The formula for recall is True Positive divided by the sum of True Positive and False Negative (P = TP / (TP + FN). Note that False positives (aka adding in extra tags) are not part of this equation

We can improve Precision by increasing the training data with clearly outlined images that could be mistaken between two classes. For deer, this would be giving more examples of fawns vs female deer (neither of which have antlers).

We can improve Recall by adding more variations of animals at different angles and locations in the image. For deer, this could be a collection of photos of male deer straight on, looking left, looking right, at the back of the image, etc.



We can also look at performance by class. This is just an average. In the above, we can see that only about 49% of the time does my model correctly detect eartags. Bummer! But hey, that 99% fawn detection is pretty great!

Part 2 - Python

The goal of this part is to take images, send them to the model for classification, and then move them into directories where each folder is the name of what's in the image. This can then be easily sent to other programs (like Camelot) for quick batch tagging, or can be referenced from an R script to get capture histories.

Start the Script

I love Python, but I don't love setting it up. I'm going to make a rude jump here and assume you have either:

- Jupyter Notebook setup on your computer (https://jupyter.org/install (https://jupyter.org/install))
- The reticulate package in R setup to run Python in RStudio

It's impolite of me to hand-hold up to this point and then leave you to your own devices, but I do not get paid enough :)

Anyway. Time for our python script. We'll begin by importing libraries:

```
import inference
import os
import shutil
import numpy as np
from collections import Counter
import supervision as sv
```

Next we want a way to thin our false positives that overlap on top of true positives. We'll write two filtering functions:

```
### For filtering results, define a way of removing the lower confidence tag
def compute_iou(box1, box2):
    # Compute intersection coordinates
    x1 = max(box1[0], box2[0])
    y1 = max(box1[1], box2[1])
    x2 = min(box1[2], box2[2])
    y2 = min(box1[3], box2[3])

    # Compute area of intersection
    inter_area = max(0, x2 - x1) * max(0, y2 - y1)

    # Compute area of both bounding boxes
    box1_area = (box1[2] - box1[0]) * (box1[3] - box1[1])
    box2_area = (box2[2] - box2[0]) * (box2[3] - box2[1])

    # Compute IoU
    union_area = box1_area + box2_area - inter_area
    iou = inter_area / union_area
    return iou

def filter_detections(detection):
    boxes = detection.xyxy
    confidences = detection.confidence
    class_ids = detection.class_id

    num_boxes = len(boxes)
    to_keep = [True] * num_boxes

    for i in range(num_boxes):
        if not to_keep[i]:
            continue
        for j in range(i + 1, num_boxes):
            if compute_iou(boxes[i], boxes[j]) > 0.9:
                if confidences[i] < confidences[j]:
                    to_keep[i] = False
                else:
                    to_keep[j] = False

    # Filter the boxes based on the to_keep list
    filtered_boxes = boxes[to_keep]
    filtered_confidences = confidences[to_keep]
    filtered_class_ids = class_ids[to_keep]

    return {
        #Don't need most of this
        #'xyxy': filtered_boxes,
        #'confidence': filtered_confidences,
        #'class_id': filtered_class_ids,
        'class_name': detection['class_name'][to_keep]
    }
```

The first function is a way of estimating overlap between bounding boxes. The second function says “if the overlap is >90%, throw away all but the most confident guess.” If you want to set a different limit, change the >0.9 to something else.

Define Pathways

This section of the code needs to be modified, probably each time you run it. I am a messy programmer, so you could probably make this less messy. But here we are.

```
#Grab model, specify where your images are stored and where you want results folders to
go
model = inference.get_model("MYMODELNAME", api_key = "MYKEY")
image_folder = '/Users/heather/Desktop/U_Georgia/Chandler_Meetings/heather/CWD_Postdoc/B
illionphotos/deer copy'
destination_folder = '/Users/heather/Desktop/U_Georgia/Chandler_Meetings/heather/CWD_Pos
tdoc/Billionphotos/deer copy'

#initialize empty list:
results_list = []
```

Here’s what’s going on

- Line one is defining where my model is. You can find the model name and api_key by going back to Roboflow and hitting “Deployments” on the side menu. Click on “Self-Hosted Inference” and then “Native Python”. In this case, you can see my model is called “ardeer/3” indicating I’m in the ardeer project, using model version 3.

  Add your Roboflow API Key as an environment variable or in a `.env` file.

  ```
  export ROBOFLOW_API_KEY="[redacted]"
  ```

  Use it in your python project.

  ```
  import inference
  model = inference.get_model("ardeer/3")
  model.infer(image="YOUR_IMAGE.jpg")
  ```

- Image_folder: Where are your images that you want tagged located? Mine are in a folder called “deer copy”
- destination_folder: Where do you want the photos moved to when they’re classified? For me, I want them to stay in that same folder, just be categorized by what’s in each folder.

Define Folder Names

Now we write a function to name all the folders in the output

```
def get_folder_name(predictions):
    class_counts = Counter(predictions['class_name'])
    #print(class_counts)
    # If there are no class names, use "no_dets"
    if not class_counts:
        return "no_dets"

    # Create a list of class names with counts, sorted alphabetically by class name
    folder_parts = [f"{count}{class_name}" for class_name, count in sorted(class_counts.
items())]

    # Join parts with underscores
    folder_name = "_".join(folder_parts)

    ### THIS IS SPECIFIC TO MY PROJECT:
    if any(word in folder_name.lower() for word in ["collar", "eartag", "cliptag"]):
        return "tagged"

    return folder_name


# Ensure destination folder exists
os.makedirs(destination_folder, exist_ok=True)
```

Note that for my project, photos with tagged animals are just combined into a folder called “tagged”, since they will require human supervision to determine the number of the tagged animal. If you don’t need that in your work, just comment out the if statement.

Classify and Sort Images

This function will take each image, run it through the AI, filter out detections that overlap and come up with a string that describes the photo (example: 1female-deer_2fawn_1male-deer). It will then check if that folder exists yet and create that folder if it does not. Then it moves the image into that folder and moves to the next image.

```
for filename in os.listdir(image_folder):
    # Check if the file is an image (you might want to adjust the extensions)
    if filename.lower().endswith(('.png', '.jpg', '.jpeg')):
        # Construct the full path to the image
        image_path = os.path.join(image_folder, filename)
        # Process the image
        result = model.infer(image=image_path)
        #save it:
        results_list.append(result)
        predictions = result[0].predictions if result[0].predictions else []
        detections = sv.Detections.from_inference(result[0])

        filtered_detection = filter_detections(detections)
        #print(filtered_detection)
        # Get the class names folder name
        folder_name = get_folder_name(filtered_detection)
        #print(folder_name) #uncomment this if you want each result printed
        # Define the destination path
        dest_folder = os.path.join(destination_folder, folder_name)
        # Create the folder if it doesn't exist
        os.makedirs(dest_folder, exist_ok=True)
        # Define the source and destination paths
        src_path = os.path.join(image_folder, filename)
        dest_path = os.path.join(dest_folder, filename)
        # Move the file
        shutil.move(src_path, dest_path)
        #print(f"Moved {src_path} to {dest_path}")

print("All images processed and moved to correct folders")
```

Here's what all this looks like when it runs:

Ta-da!

Be sure to double check all sorted images, but this should save you lots of time!