THOMAS MORE

IT FACTORY – APPLIED COMPUTER SCIENCE

Big Data Project

Image classification using transfer learning and Pytorch

Team: 8

Team Members: Daan Michielsen & Sharon Maharjan

2023 - 2024

Contents

Introduction

Part 1: Computer vision

In the first part of this project, we were challenged to create a custom image classifier model using our own data which consists of 5 similar classes, we chose for tomato types. We then had to do EDA on that data and try to create a balanced dataset which is used to create a model using transfer learning with Pytorch. After training the model we created a Streamlit application where a user can upload images of the classes which then get classified by the model.

We also had some options for extra's, and we chose the following:

1. **Flask API**: We created a Flask API endpoint that uses our model so we can inference using a request to the endpoint and get a result back.
2. **Google Teachable machine benchmark**: Comparing the performance of our model to the performance of Teachable Machine.
3. **Google Vertex AI model benchmark**: We created a model using Google Vertex to see the difference in performance. This model can be accessed by sending a request to the endpoint where the model is deployed.
4. **Roboflow model benchmark**: We created a model using Roboflow to compare the performance with our own model. This can be done by sending a request with an image to the deployed model on Roboflow.
5. **ROC/AUC metrics**: We generated the ROC/AUC curve for our custom models to see the performance and the difference between the 1st iteration and our best model.

Part 2: Natural Language Processing

For the second part of this project, we were challenged to create an NLP classification model on the famous "toxic comments" dataset from Kaggle. This dataset contains reddit comments which contain some language that belongs to certain classes like:

- toxic
- severe toxic
- obscene
- threat
- insult
- identity hate

Unfortunately for this project we did **not** get the chance to finish part 2 since we are a team of 2 students instead of the recommended 3 students. However, we looked at the data and tried to do an easier version by classifying between toxic and non-toxic comments.
This means we had no time to do both parts.

Part 1: Computer vision

Data Collection

All our used data originates from web scraping using Selenium and from Bing on Google images. This approach did force us to perform quite a lot of cleaning of the data since Google does not show images of what you look for all the time. To collect the images of tomatoes, we have used different search tags: 'Solanum lycopersicum Celebrity', 'Solanum lycopersicum Green Zebra', 'Solanum lycopersicum Pineapple', 'Solanum lycopersicum Super sweet 100', 'Solanum lycopersicum Yellow Pear'. While scraping the images, we get irrelevant images or the duplicate one. We have to it manually remove the bad ones.

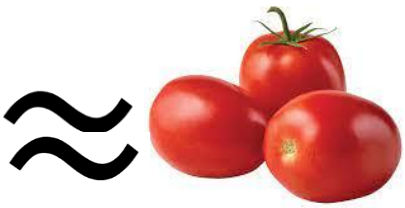
EDA

We know that for transfer learning you do not need that much data, so we aimed for 100 images per class. So, we started scraping 100 images with the following search queries:

- Celebrity tomato
    - Solanum lycopersicum Celebrity
- Roma tomato
- Pineapple tomato
    - Solanum lycopersicum Pineapple
- Green zebra tomato
    - Solanum lycopersicum Green Zebra
- Yellow pear tomato
    - Solanum lycopersicum Yellow Pear

After removing the bad images this left us with image numbers for each class ranging from 60-80 depending on the success of the search result. So, this was not enough but this was just to start so we also trained our first model on this using the default settings and a resnet-50 model. But our confusion matrix showed mixed results (Unfortunately we did not save it). We also got the insight that our "Roma tomato" was too like the "Celebrity tomato", so we changed the "Roma tomato" to the "Super sweet 100" tomato since the images we're more different.

Celebrity tomato



Roma tomato

Super sweet 100

Out of the box model

For the modeling we started by using the out of the box model provided in the second chapter of our Big Data course which has the following:

Settings:

Pretrained model: Resnet50

Total images: 360

Training set: 80%

Validation set: 20%

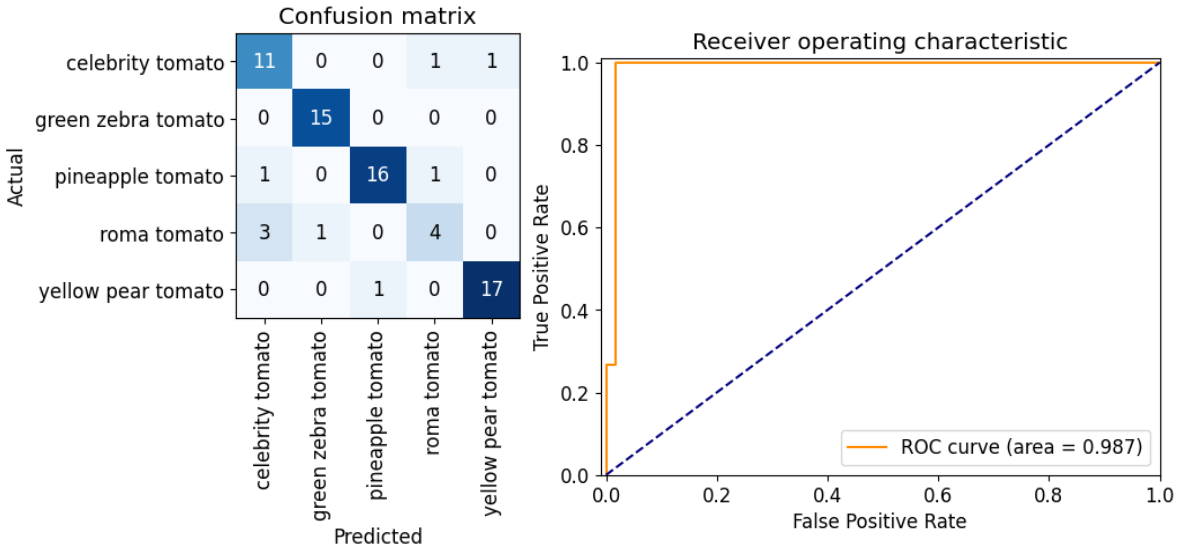
Image size: 460x460

Augmentation:

Size: 224x224

Min_scale: 0.75 => zoomed up to 75% of original size

Performance metrics



Note: For some reason the loss plot was empty for this approach

Advanced model

For our advanced model we worked with a more advanced topic from our third lesson which used some transfer learning specific parameters that are shown in our settings.

Settings:

Total images: 360

Training set: 80%

Validation set: 20%

Image size: 460x460

Augmentation:

Size: 224x224

Min_scale: 0.75 => zoomed up to 75% of original size

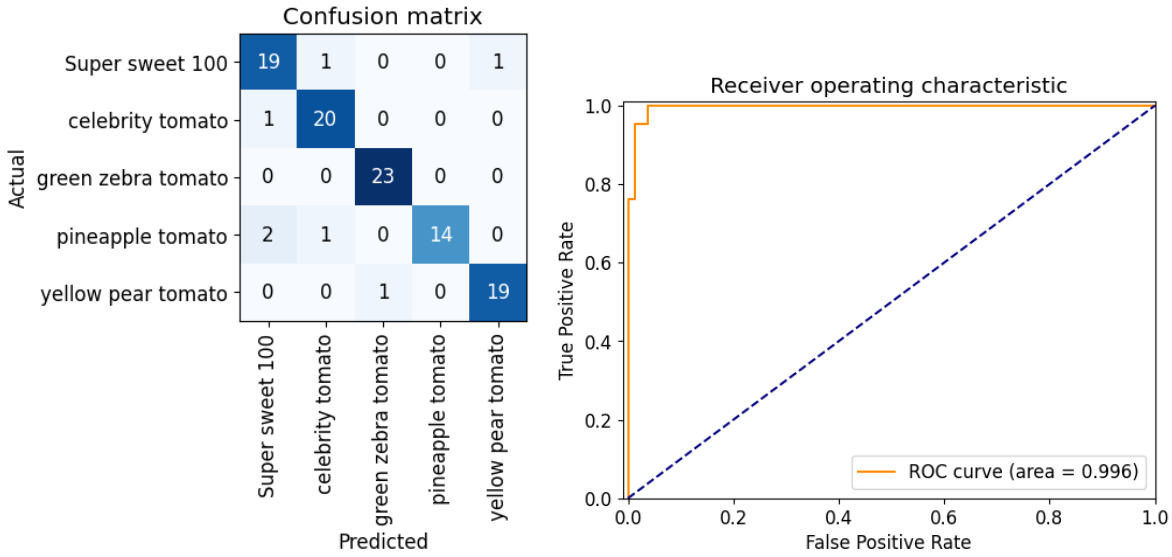
Up till this point the settings have been the same but right here things start to change. To get our model to perform better we used "Discriminative learning rates". So, we use a "one cycle" approach but this method does not update the weights of each layer as hard. The weights closest to the output layer get updated the most (still not that much), and the further we go back into the layers the less we will tweak the weights.

It looks like this:

```
our_model_really_adv.fit_one_cycle(3, 3e-3)
our_model_really_adv.unfreeze()
our_model_really_adv.fit_one_cycle(12, lr_max=slice(1e-6,1e-4))
```

The slice defines the range over which layers the learning rates should be spread out.

Performance metrics



Note: For some reason our loss plot was empty for this approach of training

If you compare it with our first model, you can see the improvement.

Classifying

With our first model the classifying did not go that well because the model was using the confusing classes of “celebrity tomato” and “roma tomato” which was making 2/5 classes hard to get right most of the time.

When we saw this problem, we looked for other classes which made us swap out “roma tomato” for “super sweet 100 tomato”. At first, we thought that the shape and color of the tomatoes were still too similar, but the images were actually fine since the “super sweet 100 tomato” is usually many tomatoes on a branch.

Celebrity tomato

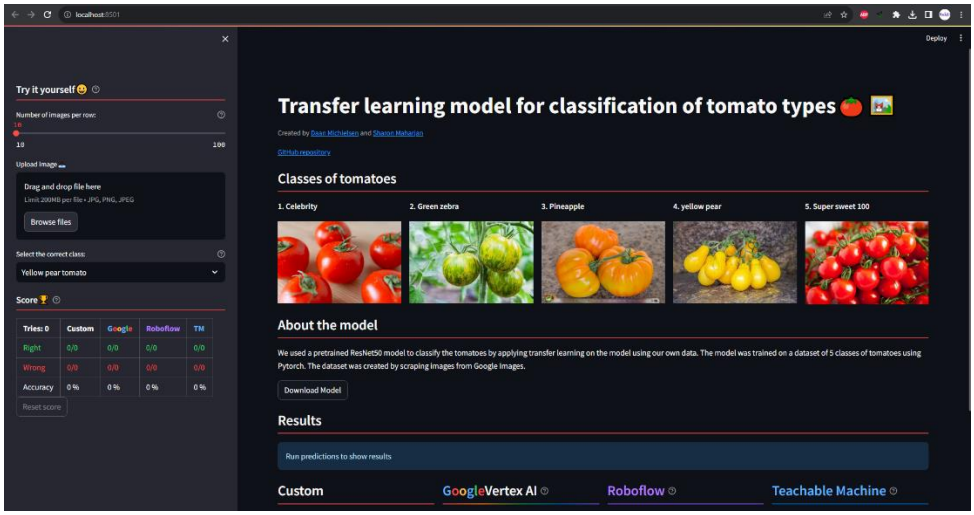
Super sweet 100

Streamlit

We created a Streamlit app where all our models can be used for inferencing on images that can be uploaded in jpg, jpeg, or PNG.

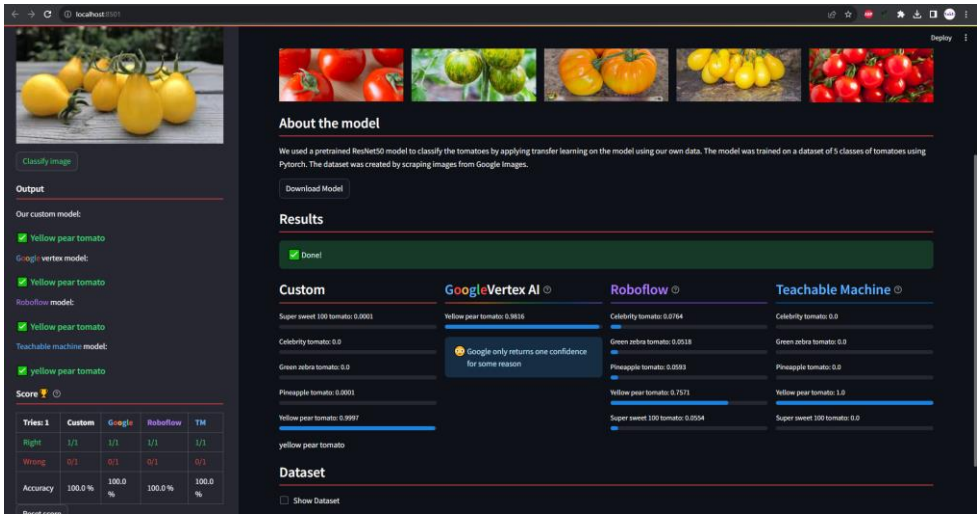
You can also explore the data that was used for the training of the models to see what kind of data was used.

Landing page



Dataset exploration

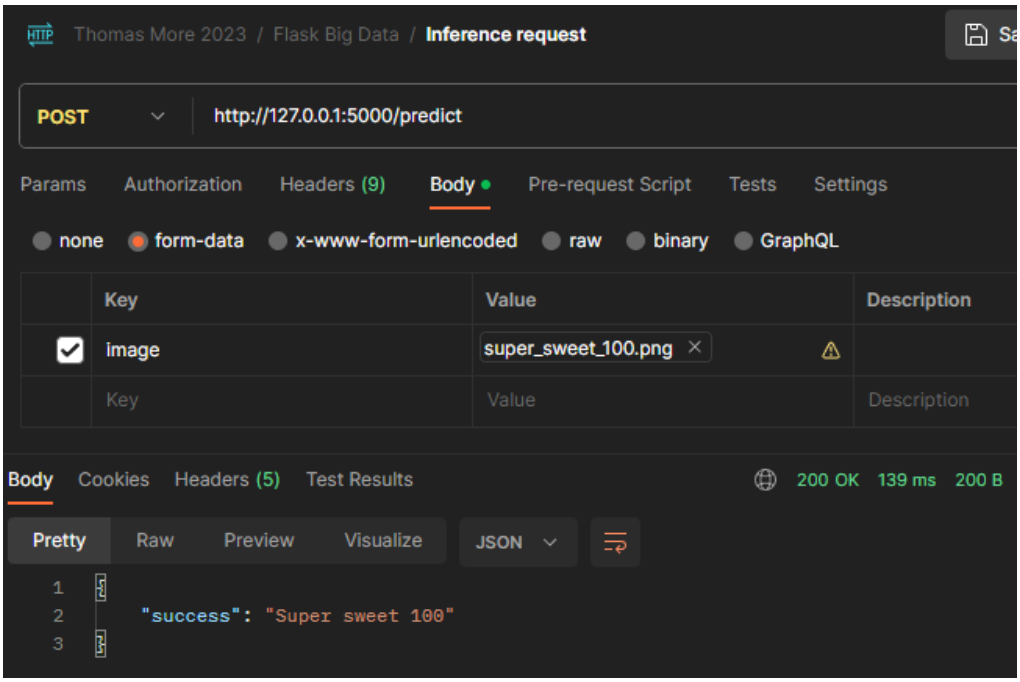
Inferencing result



Note: We tried deploying on Streamlit cloud with just our custom model, but we had trouble loading our model

Flask API

Flask API is a micro web framework written in Python. It is capable of receiving and responding to HTTP requests. When we send one of our images for prediction, the image is processed by our trained model. We previously trained our model to learn the patterns and features of the data. The Flask response helps to determine the prediction generated by the model. For e.g. if we send an image of a "Super Sweet 100", the response would be success:" Super Sweet 100".



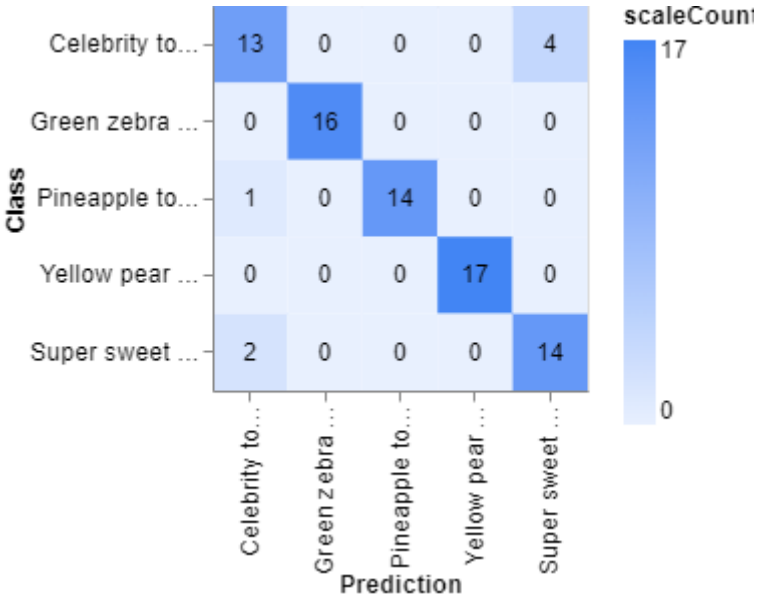
Postman request

Benchmark

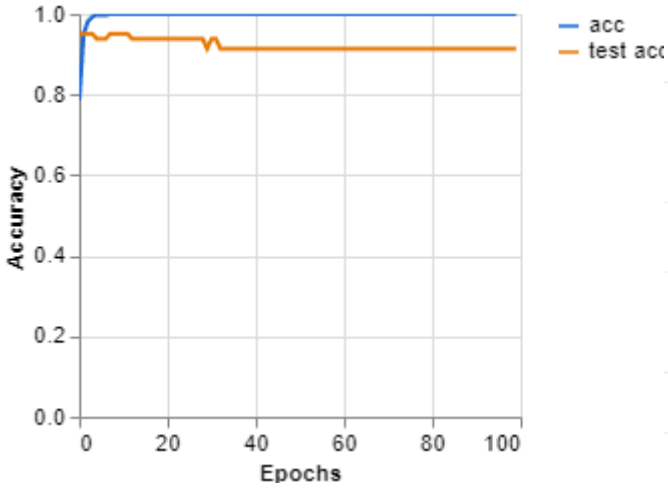
A benchmark is employed to compare or identify the tools and performance of technologies. In our project, we utilized three different types of benchmarks.

Google Teachable Machine Benchmark

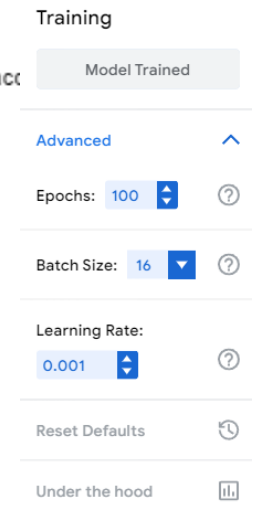
We compared the performance of our model to that of the Teachable Machine. We trained our model using the same images from the custom model.



Confusion matrix



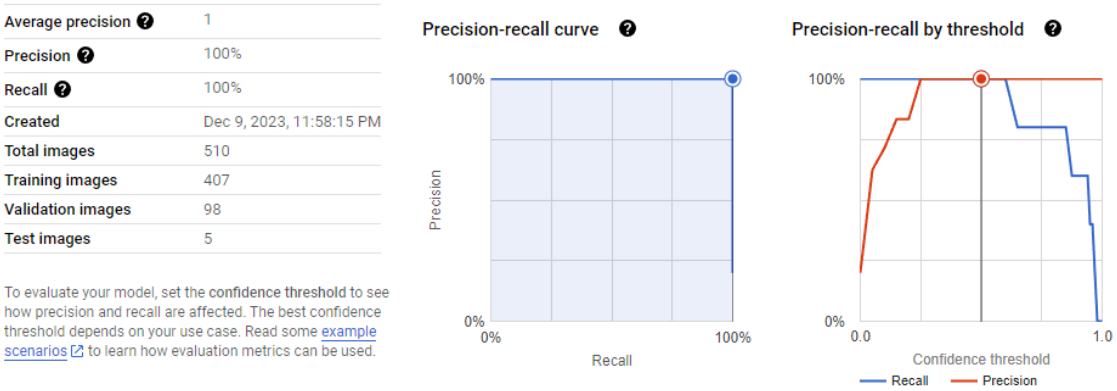
Accuracy per epoch



Settings

Google Vertex AI Model Benchmark

This benchmark was employed to observe the change in the performance of a model. As the model was created, we can access it by sending a request to the endpoint where the model is deployed.



Training matrix

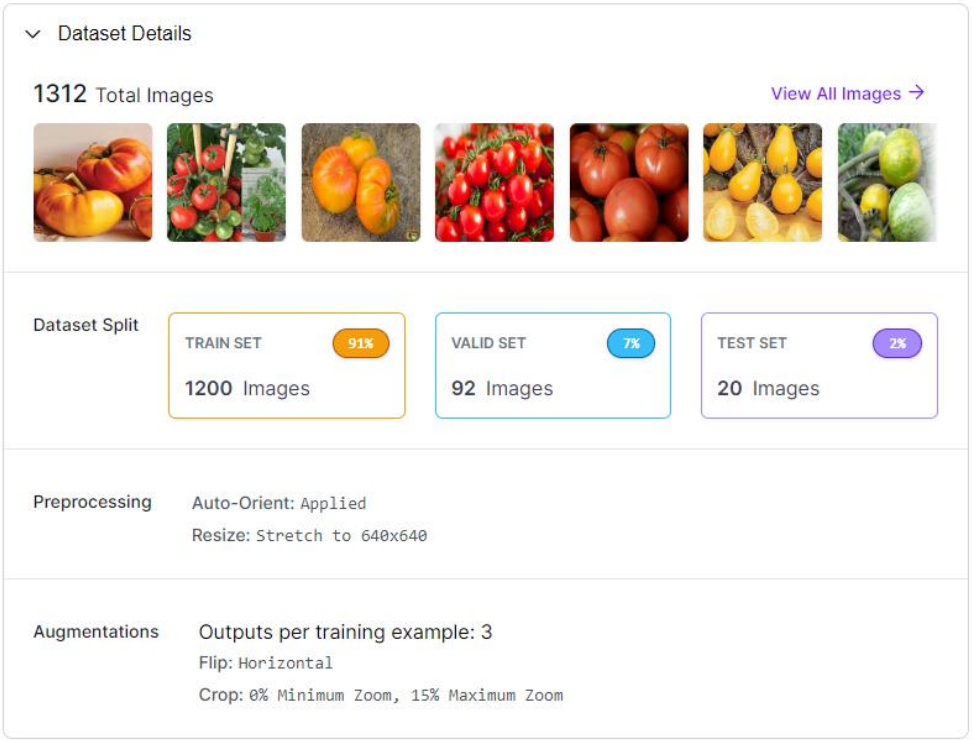
True label \ Predicted label	Pineapple_tomato	Green_zebra_tomato	Yellow_pear_tomato	Celebrity_tomato	Super_sweet_100_tomato
Pineapple_tomato	**100%**	0%	0%	0%	0%
Green_zebra_tomato	0%	**100%**	0%	0%	0%
Yellow_pear_tomato	0%	0%	**100%**	0%	0%
Celebrity_tomato	0%	0%	0%	**100%**	0%
Super_sweet_100_tomato	0%	0%	0%	0%	**100%**

Confusion matrix

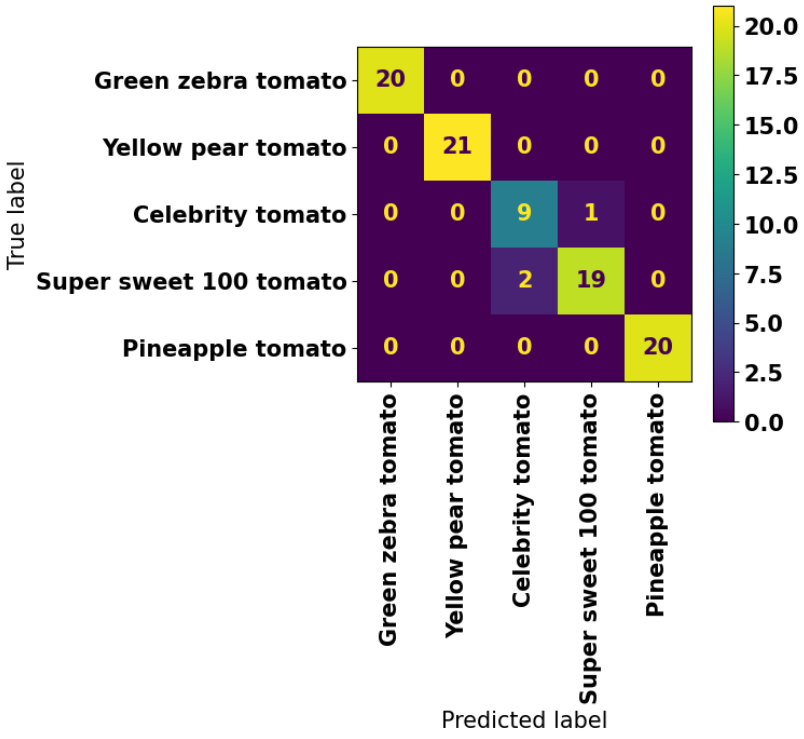
Note: These values are not representative because the model only used 1 image for testing per class.

Roboflow Model Benchmark

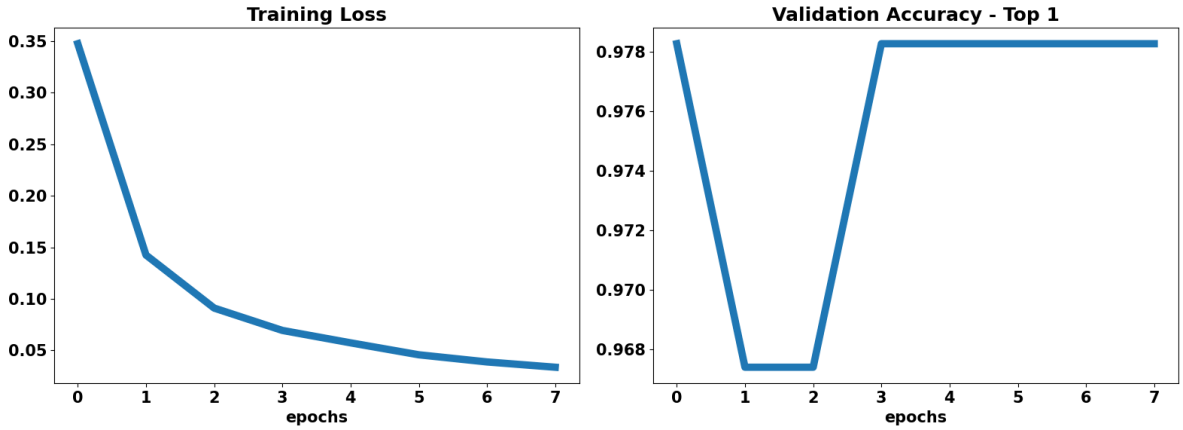
We created a model using Roboflow to compare its performance with our own model. This can be achieved by sending a request with an image to the deployed model on Roboflow.



Dataset



Confusion matrix



Training graphs

Problems

The problems we faced during this project are as follows:

- When we scraped images from Google and Bing, we don't always get relevant images such an advertisement for tomatoes or a drawing of tomatoes.
- We could not consistently beat Google's Teachable machine model, the reason for this could be that we let Teachable machine learn for 100 epochs when our model only did a couple with discriminative learning rates.

Instruction on how to install required dependencies to run programs.

Note: Before installing the packages, we recommend creating a virtual environment of Python 3.11 or lower. (python -m venv venv)
Run "./venv/Scripts/Activate.ps1" in terminal to activate the environment.

Streamlit:

If you go to the requirements.txt file, then you can see all the libraries that are used in the application. But if you use this command pip -r then it opens the requirements.txt and then install the packages inside of it automatically.

To run the application, you have to use the command “streamlit run app.py” in the root folder.

Flask API:

The necessary packages are already installed using the requirements.txt file.

To run the API locally you must run this command: “python app.py” in the root/Flask/ directory.

There are 2 options:

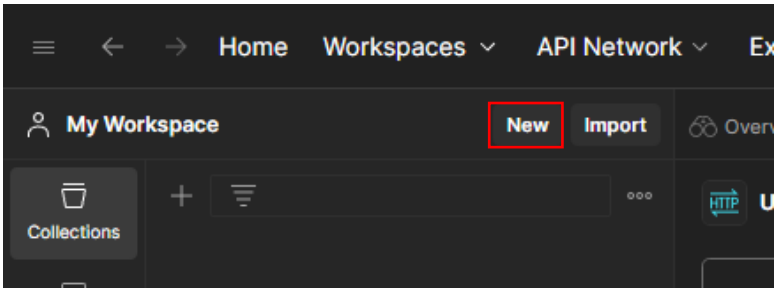
1. Curl command:

   curl -F "image=@{absolute_path_of_image.png} " localhost:5000/predict

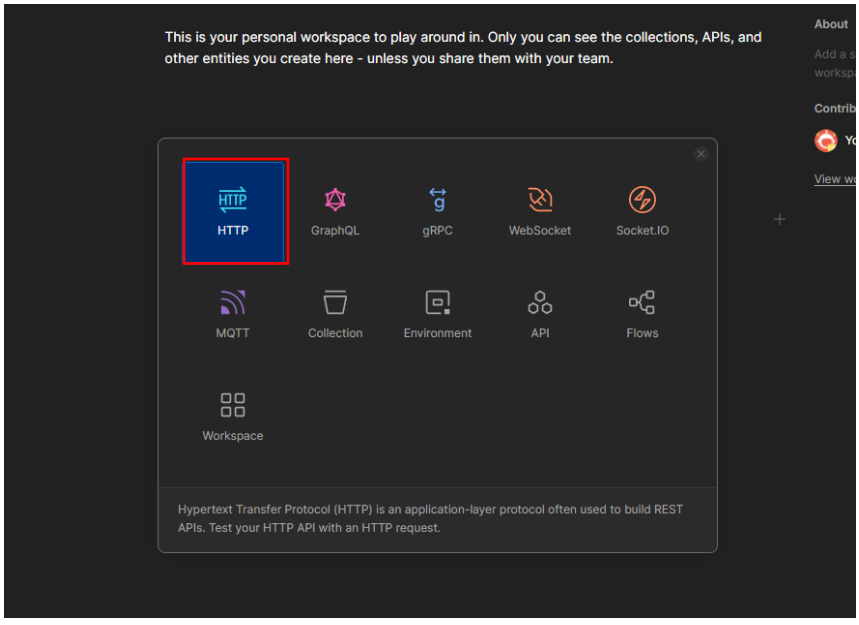
   Replace “{absolute_path_of_image.png}” with the absolute path of the image on your device.

2. Postman request:

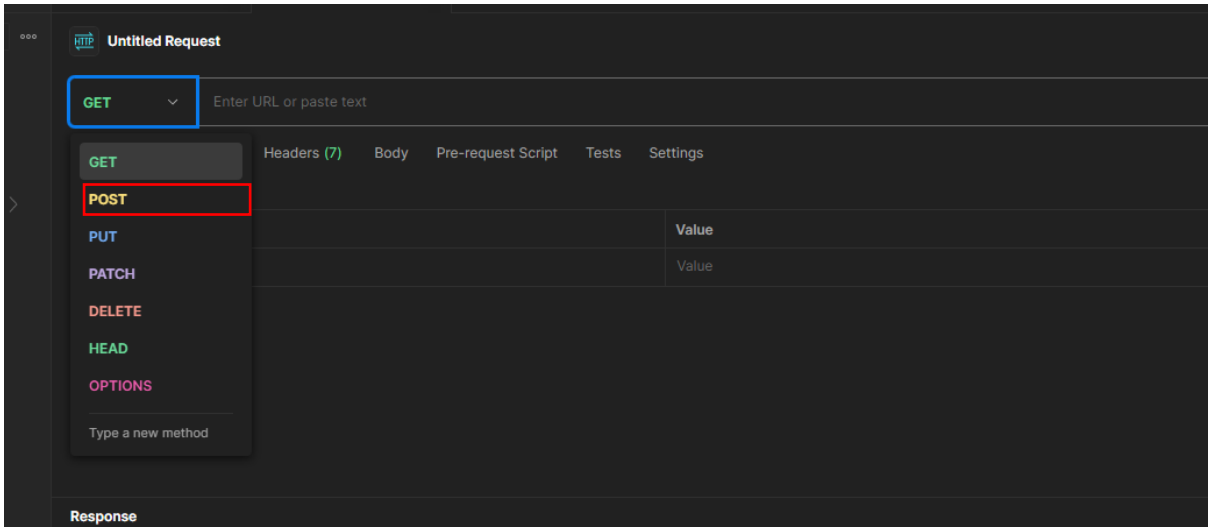
   Step 1: create a new request at the top right of the page



   Step 2: Select HTTP

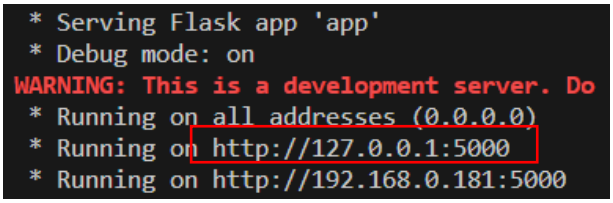


Step 3: Select post request.

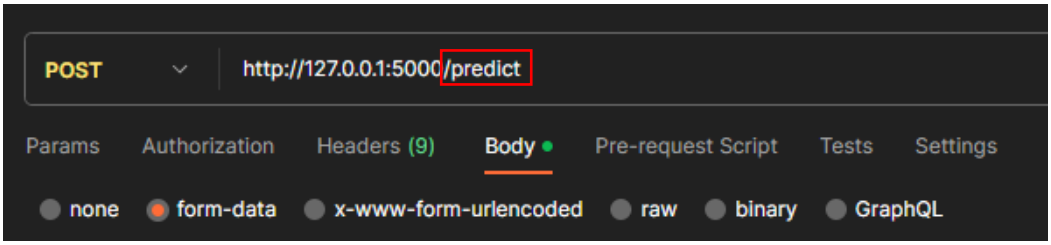


Click the dropdown where it says GET and select POST

Step 4: copy the URL from your terminal that says 127.0.0.1 and paste in postman address bar.

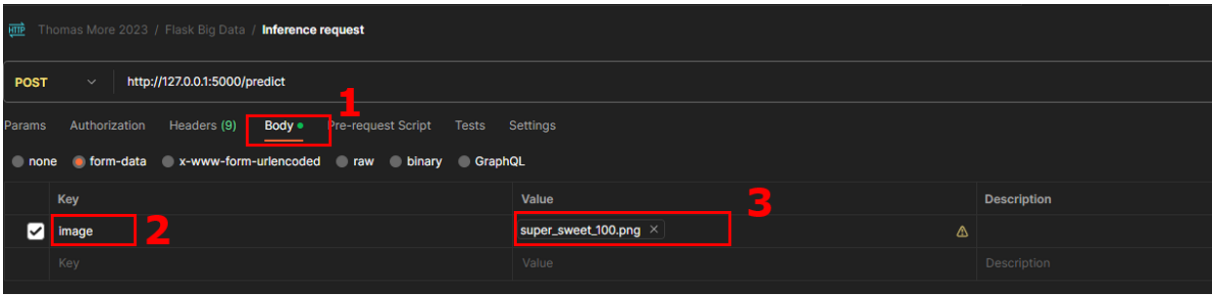


Paste it and add /predict. That is the endpoint where the model is running.

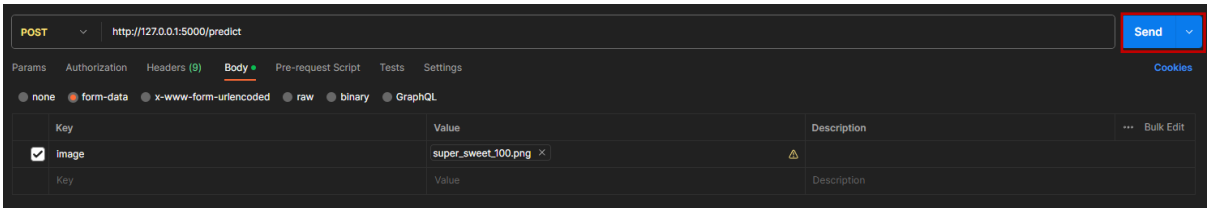


Step 5: Add image to request body.

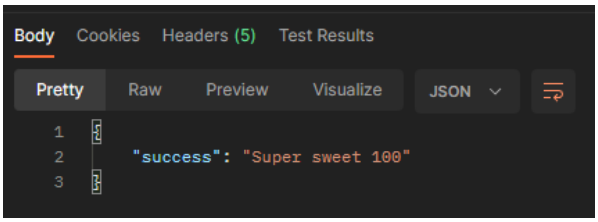
First select body. Then add a key called image and finally upload the image.



Step 6: Send the request.



If the request is successful, the output should show something like this:



Part 2: Natural Language Processing

EDA

We looked at the data and looked at what is contained in the data. These are the data that were their id, comment_text, toxic, severe_toxic, obscene, threat, insult, identity_hate.

Value counts

toxic	severe_toxic	obscene	threat	insult	identity_hate	
0	0	0	0	0	0	143346
1	0	0	0	0	0	5666
		1	0	1	0	3800
				0	0	1758
		0	0	1	0	1215
	1	1	0	1	0	989
	0	1	0	1	1	618
0	0	1	0	0	0	317
		0	0	1	0	301
1	1	1	0	1	1	265
0	0	1	0	1	0	181
1	1	1	0	0	0	158
	0	0	0	0	1	136
				1	1	134
		1	1	1	0	131
		0	1	0	0	113
	1	1	1	1	0	64
	0	1	1	1	1	56
0	0	0	0	0	1	54
1	1	0	0	0	0	41
	0	1	0	0	1	35
	1	1	1	1	1	31
0	0	0	0	1	1	28
			1	0	0	22
		1	0	1	1	18
1	0	0	1	1	0	16
	1	0	0	1	0	14
			1	0	0	11
	0	1	1	0	0	11
		0	1	0	1	7
	1	0	0	1	1	7
		1	0	0	1	6
			1	0	0	4
		0	0	0	1	3
	0	0	1	1	1	3
0	0	1	0	0	1	3
		0	1	1	0	3

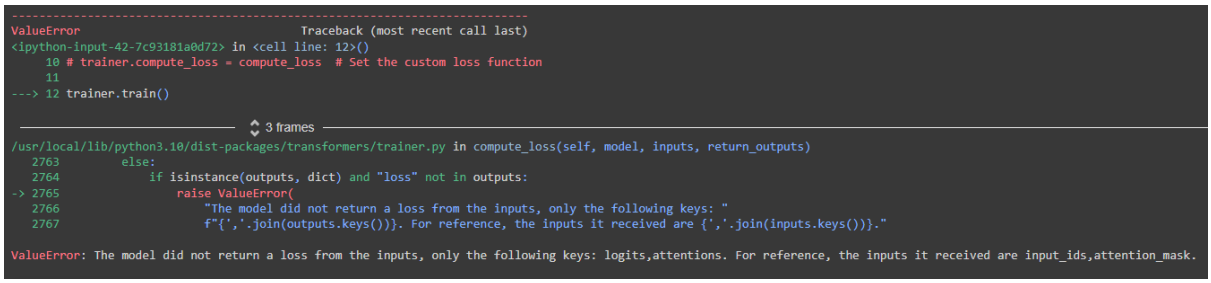
Preprocessing

We decided to narrow down the classes to is_toxic or not.

```
is_toxic
0    143346
1     16225
Name: count, dtype: int64
```

Training

For the training we tried using the code we got from our notebook with the example of the emails. But we kept on getting errors while training like this:



This was the case for Colab and Kaggle so we stopped working on it because of time constraints.

Conclusion

In the first part of our project, we focused on creating a custom image classifier for tomatoes. We used our own dataset and started by ensuring it was well-balanced through Exploratory Data Analysis (EDA). Using PyTorch and transfer learning, we built a model. We took on additional challenges by implementing a Flask API and benchmarking our model against Google Teachable Machine, Google Vertex AI, and Roboflow. We evaluated performance using ROC/AUC metrics. In our Streamlit application, you can locally explore the dataset, view images for each class by checking a checkbox, adjust the displayed number of images with a slider, and even classify tomato images by dragging and dropping them. The app also shows accuracy metrics for custom, Google Vertex AI, Roboflow, and Teachable Machine models, and provides an option to download the model.

For the second part of our project, we aimed to build an NLP classification model using Kaggle's 'toxic comment' dataset. The goal was to categorize comments as either toxic or non-toxic. Unfortunately, due to time

constraints with our two-person team, we couldn’t complete this part of the project.

Note

We know that our API keys are exposed in here but we don’t know another way around it for now.