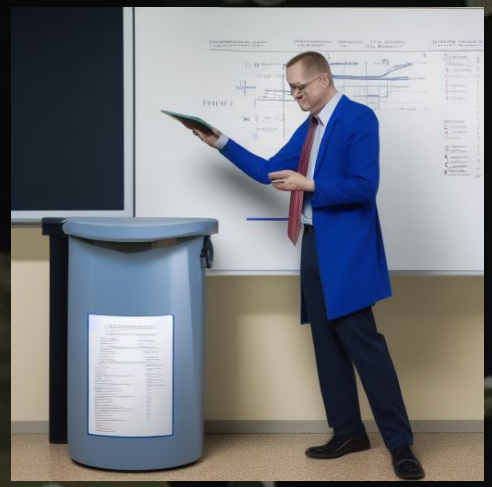
TrashClassifier

Team Big Data Energy

Bryan Burch - Christopher Allen - Daniel Smagly - Jeffrey de Jesus
Julian Hernandez - Kenta Miyahara - Travis Hammond - Santiago Bermudez

Project Overview

Dr. Clark Fitzgerald	Client's Interest	Client's Problem	Proposed Solution
CSUS Professor in Math/Stats Department	Research in statistics & machine learning as well as improving proper trash disposal	Wants a tool to classify trash objects and decide the appropriate trash bin of such	Application that locates and classifies trash in images with a deep neural network models Developed Application A terminal app, terminal script, and a website that connect to a server running EfficentNet and CLIP



Requirements

- **Front-End : React website & Terminal App**
  - Terminal App
    - Menu, Classify, Trash Description, Setting, About
  - React Website
    - Model selection, FileInput(single/multiple), view of the result of classification

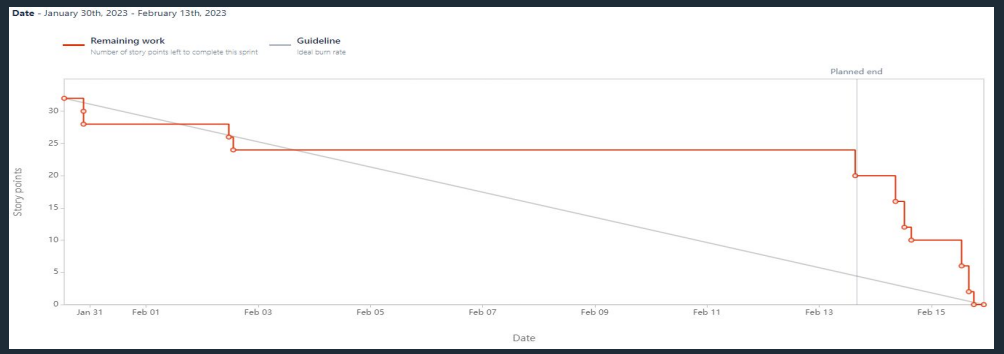
- **Back-End : Flask / Rest API / SQLite**
  - Upload/Download Models
  - Image classification
  - Return Model / metadata
  - Overwrite model or metadata

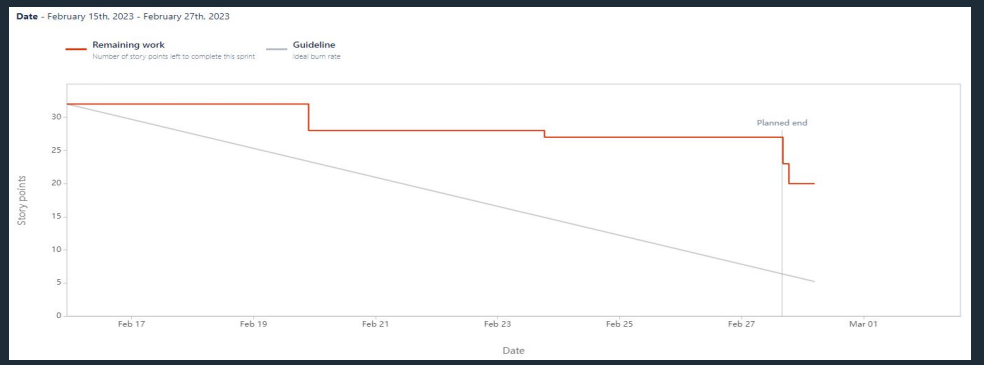
```
{
  "Id": "",
  "annotation" : "",
  "num_annotations": "",
  "dataset" : "",
  "metadata" : "",
}
```

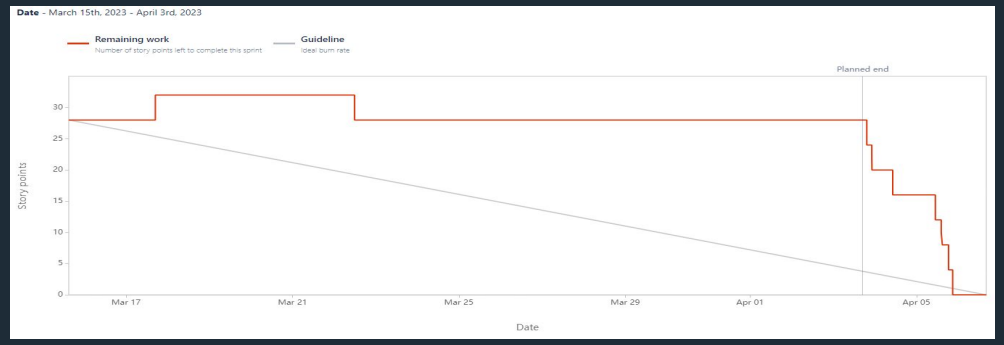
- **ML : PyTorch / OpenCLIP / EfficientNet**
  - Accuracy around 75%
  - Bin classification

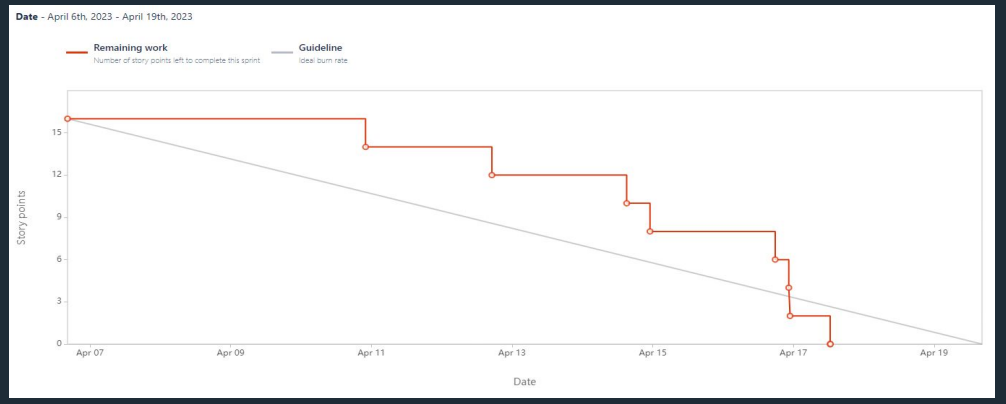
Sprints 05-09 Burndown Charts

Current Presenter:
Kenta Miyahara



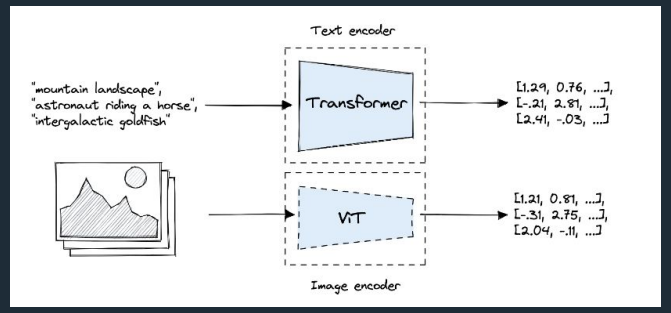






Design

Create multiple models, and using zero shot transfer learning, detect trash types



Search localized recycling information using the City of Sacramento's Waste Wizard API

Tell the user what item they are holding and how to dispose of it

EfficentNetV2

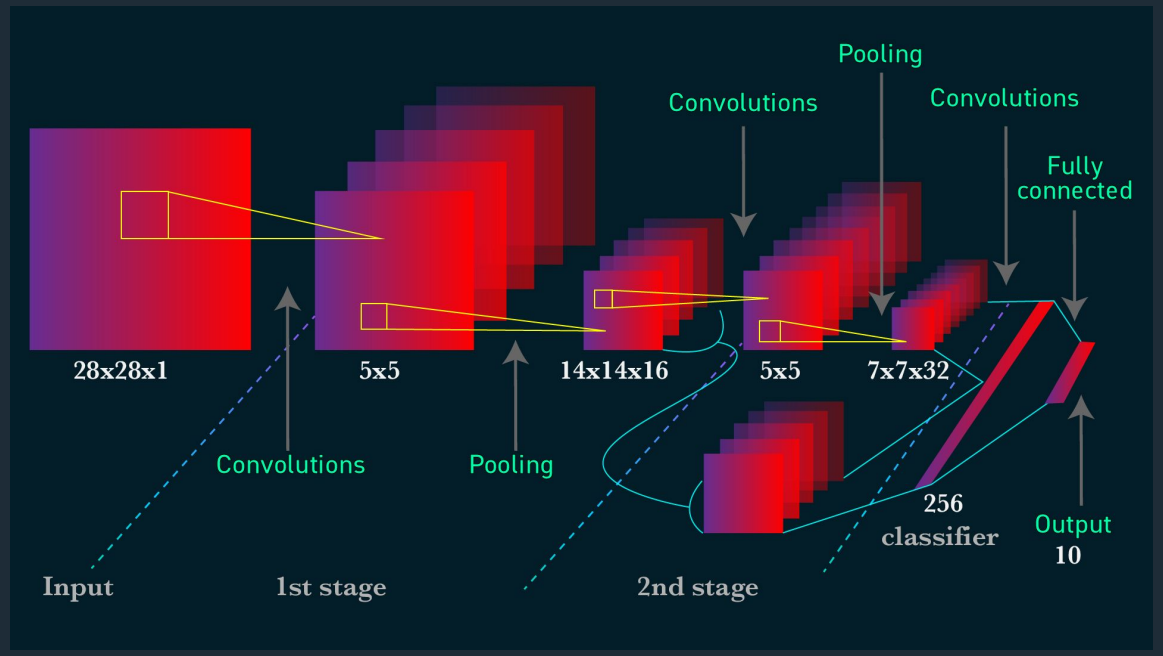
One of the latest CNN image classification models

Predict *Object Class* using predefined categories

(1000 unique categories)

Pass through conversion map to convert *Object Class* to trash category

Collect top K *Object Classes* from previous prediction and their respective trash category to calculate the overall likely *Trash Class*



OpenCLIP

An Image-to-Text **Vi**sion **T**ransformer (ViT) model

Predict *Object Class* using Natural Language

(Ex. “A photo of a water bottle”)

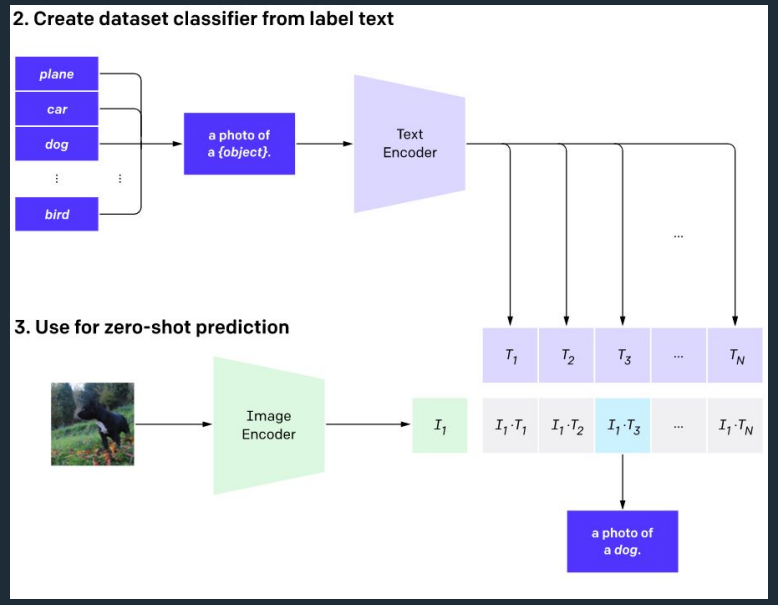
Pass through conversion map to convert *Object Class* to trash category

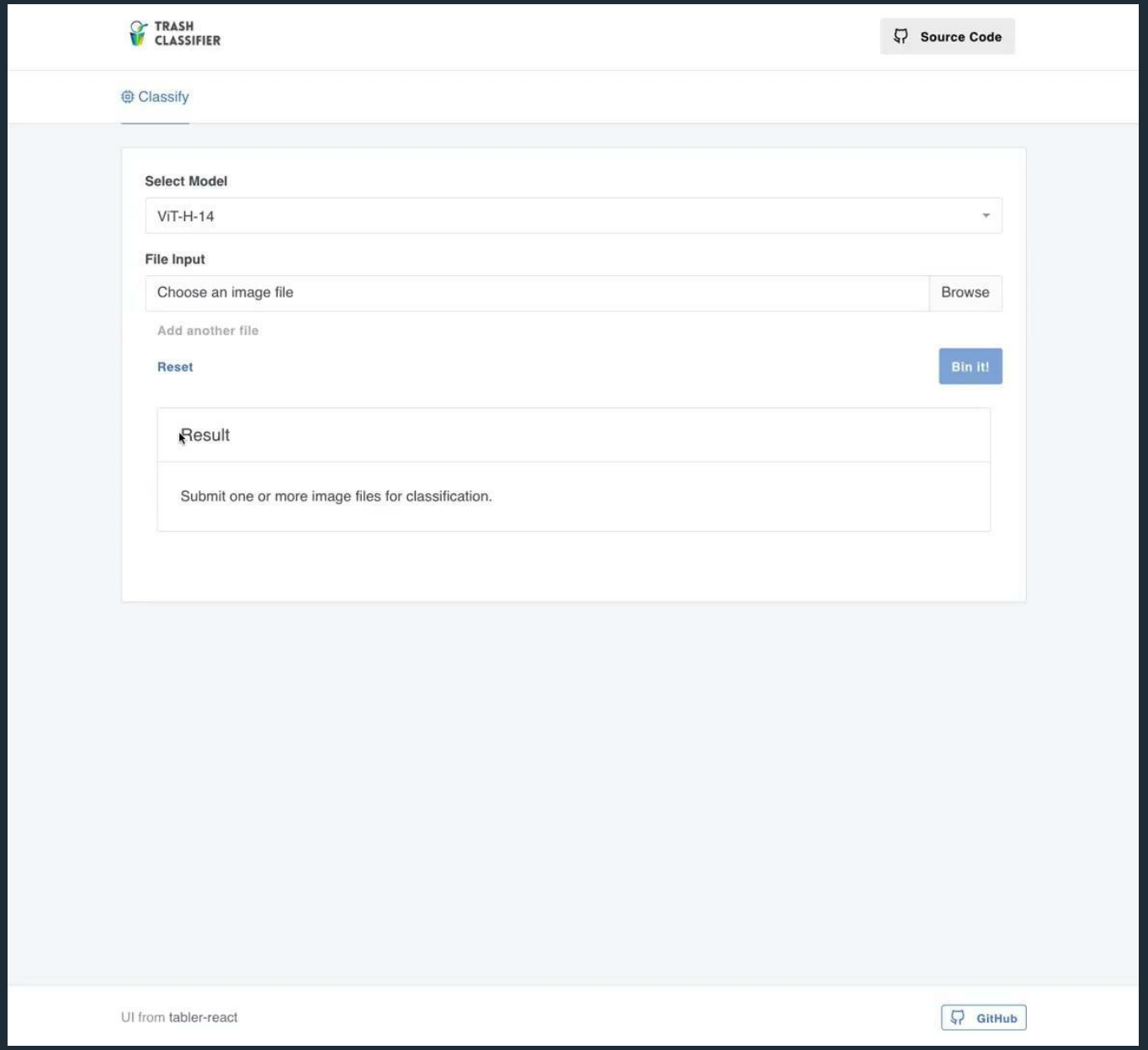
(60 unique categories mapped to one of 4 trash categories)

&

Predict *Trash Class* using Natural Language

(Ex. “A photo of garbage”)




TRASH CLASSIFIER
Source Code
Classify
Select Model
ViT-H-14
File Input
Choose an image file
Browse
Add another file
Reset
Bin it!
Result
Submit one or more image files for classification.
UI from tabler-react
GitHub

Live Demo Time!!!

Current Presenter:
Jeffrey de Jesus

Implementation

		Sprint03	Sprint04	Sprint05	Sprint06	Sprint07	Sprint08	Sprint09
Frontend	Interactive Menu	Create Seperate Menus	Integrate With REST API	Create Tests		update features		
	Command Line App				Create App	Finalize App		
	React Website				Create Prototype	Implement Features	Create Tests	
Backend	Annotation Database Server	Create REST server	Finalize REST API	Create Tests				
	Model Inference Server	Create REST server	Finalize REST API	Create Tests	Fix Bugs			
	Website Server			Create prototype	Integrate with frontend			
	Consolidated Server				Combine Servers	Deprecate old servers		
Model	Model Creation	Collect Training Data	Model Architecture	Model Training	Switch to Pytorch	Model Training		Model Downloading
		Research Models				Integrate with Servers		
	Model Validation						Model Testing	Validation Training
Misc.	Combining Code			Fix Bugs				
	Finalizing Code						Refactor code delete old code	Finalize codebase

Later sprints had less and less work:

- Adjusting our approach to the model and implementing pytorch
- Adjustments to codebase to work with model

Implementation

Languages
- Python
- Javascript

SQLite

Technologies
- Flask
- Rest API
- React (web frontend)
- PyTorch/OpenCLIP (Neural Networks)
  - Efficient Net: Convolution Layers
  - CLIP: Transformer Layers

IDEs
- Jupyter Notebook
- Visual Studio Code

Project Statistics

278 hrs of development time

4,886 lines of Python
10,646 lines of Javascript
10,049 lines of JSON

210 Files
- 102 Images (JPG/JPEG/PNG)
- 36 Python
- 19 Javascript
- 13 JSON
- 4 Jupyter Notebooks

Current Presenter:
Bryan Burch

Testing

1. Unit Testing
2. Model Testing
3. Functional and Integration Testing

Unit Testing

- Pytest: terminal app and Flask servers
  - setup database & server states using fixtures
  - encapsulate tests for a feature in a class
- Enzyme: React website
  - mock model selection & verify state update
  - shallow render components

```
@pytest.fixture()
def app():
    app = create_app()
    app.config.update({
        "TESTING": True,
    })

    yield app
```

```
describe("handleDropdownChange", () => {
  it("should set selectedModel state", () => {
    const wrapper = shallow(<ClassifyForm />);
    wrapper.instance().handleDropdownChange({
      target: { value: "model1" },
    });
    expect(wrapper.state("selectedModel")).toBe("model1
  });
});
```

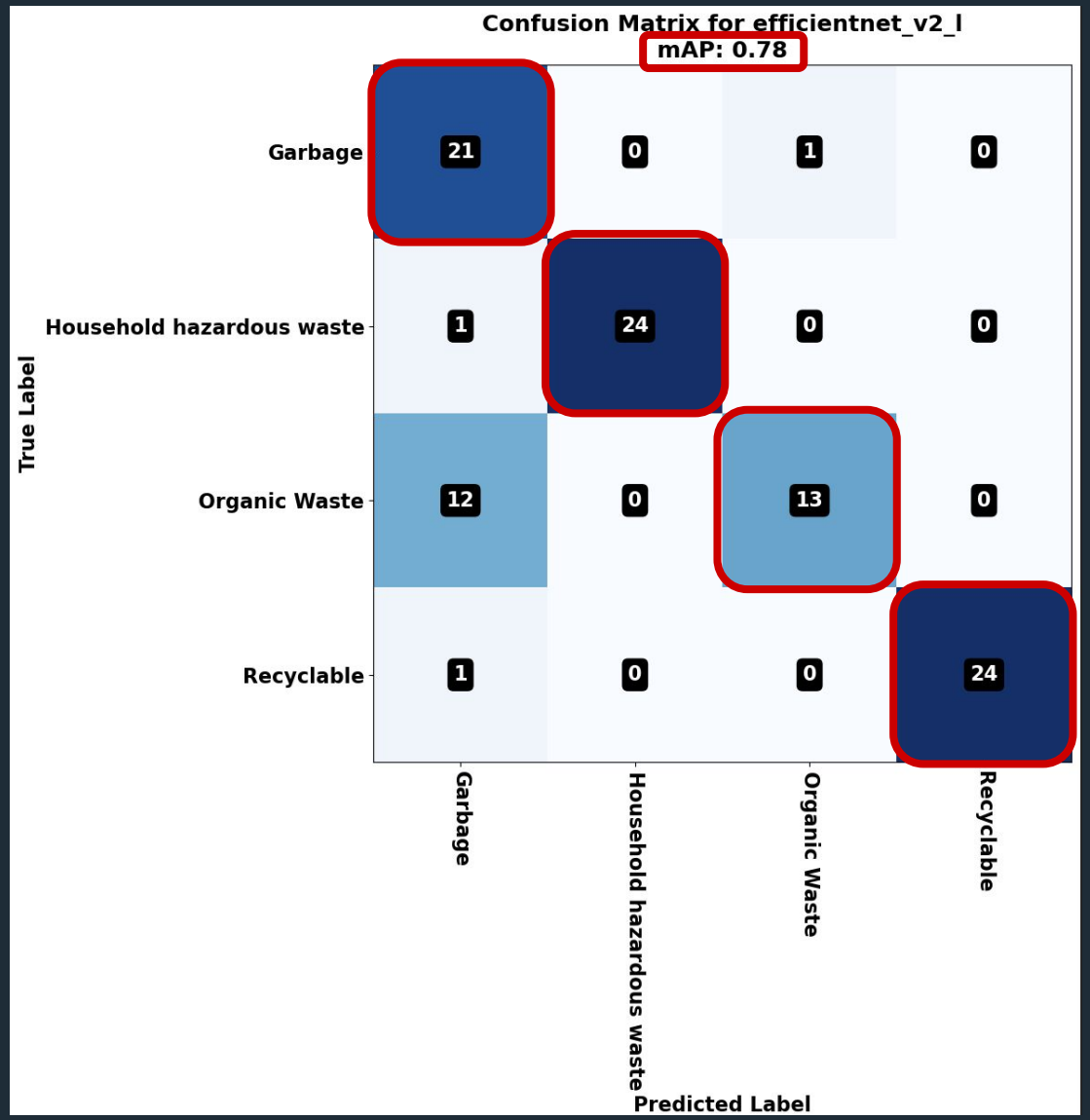
```
class TestReadInference:
    def test_no_image(self, client):
        res = client.post(
            'http://localhost:5000/read/inference/test')
        assert res.status_code == 200
        assert res.get_json() == {'error_code': 3,
                                  'error_msg': 'Missing image
                                  'model_name': 'test',
                                  'predictions': None}

    def test_success(self, client, tmp_img):
        res = client.post(
            'http://localhost:5000/read/inference/test', data=
```

Model Testing

- Gathered real world images of trash
- mAP: metric measuring accuracy which should be maximized
- confusion matrix: performance visualization where actual & predicted labels should coincide

```
efficientnet_v2_l 0.7825558977105369
ViT-g-14 0.7217932677288348
ViT-L-14 0.6734351010998718
ViT-B-16 0.5012932274188851
```



Functional & Integration Testing

- non-automated & informal
  - manually navigating UI, checking for expected behavior
  - inputting images from terminal/website apps, checking inference result

Deployment

Front end server deployment:

- Application is deployed on a Flask server for remote inference

REST API deployment:

- Training dataset API is designed to handle image annotations, image tags as well as manage database contents.
- Model inference API designed to handle model features as well as retrieve new models.
- Our API design is useful for the existing dataset design, and its simple modularity allows for easy portability to other projects with similar functionality.

Download Models

- Various pre-trained models are available for download with options for both GPU and non-GPU enabled systems

```
1. GPU model: ViT-g-14 (5.47 GB)
2. GPU model: ViT-H-14 (3.94 GB)
3. CPU model: ViT-L-14 (1.71 GB)
4. CPU model: ViT-B-16 (599 MB)
M. Return to Menu
2
Downloading model...
Downloading (…)ip_pytorch_model.bin:  27%|▌| 1.08G/3.94G [01:04<03:59, 12.0MB/s]
```

Project Expenses

Funds were received from Sac State Sustainability Student Grant

Trash Grabbers	$74.99
Camera Module	$31.25
Raspberry Pi 3B	$35.00
Total	$141.24

Lessons Learned

New Technologies/Tools

- Flask
- REST API
- SQLite
- React (Web Frontend)
- PyTorch/OpenCLIP(Neural Networks)
- Jupyter Notebook
- React and Js

Soft Skills

- Needed to switch tools/models halfway
  - Learned how to adjust and make changes for the new and superior tools
- Learned how to communicate effectively and work well together remotely

Insights/Wisdom

- Definitely have good communication.
- Don’t be afraid to ask for guidance or clarification when needed.
- When changes arrive, be able to plan ahead for that and be able facilitate a smooth transition for new goals.

Conclusion

Client's feedback

- Very satisfied with the project.

- Ready to sign product delivery document.

Insights

- Communication is key.

- Working together through any issues.

Possible Improvements

- Making a custom fine tuned model.

- Annotation options to the website.

Q&A