Image Recognition Model (Carmen)

Nicholas Harris, Josh Cooper, Garrett London,
Guy Phelps, James Clabo, Cristian Henriquez

Introduction

Neural Network

- Created an image recognition model, nicknamed Carmen, that identifies various fruits utilizing PyTorch, a python library intended for use in machine learning objectives.
- Estimated the project's feasibility, extracted a potential timeline, and demonstrated the software engineering process through best practices observed in our course.

Demand Analysis

Identifying Market & Utility

- The market for this product would be a set of users who do not know the identification of a fruit.
- This product would most likely be used as a tool for customers or employees of a supermarket.
    - Ex: Camera placed at checkout, identifying and automatically checking out an item based on visual identification.

https://www.theverge.com/2021/6/15/22534570/amazon-fresh-full-size-grocery-store-just-walk-out-cashierless-technology-bellevue-washington

Identifying Competition

- Multiple different variations on programs capable of identifying plants and fruits by image. None have a large name or hold on any market.
- Among these products, none are used on a large or commercial scale, as their usefulness is shadowed by the somewhat poor cost-effectiveness of such a product.
    - Would simplify the job, but it can still be performed efficiently by a human .

Fruit Identifier 4+

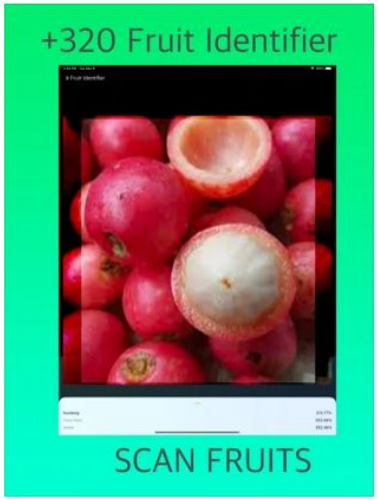
Fruit Scanner Identifies +320

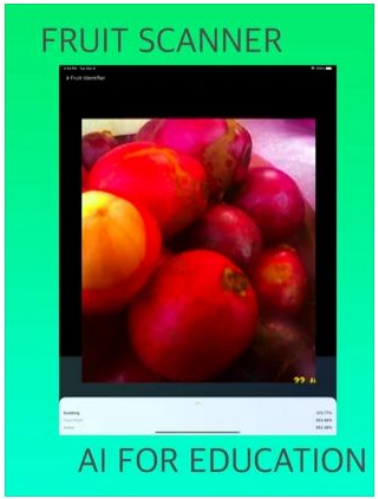
PABLO FABRE

Designed for iPad

$2.99

Screenshots iPad iPhone





https://apps.apple.com/us/app/fruit-identifier/id1543972622

Feasibility Analysis

Technical Feasibility

- Requirements include software, package-managers, language libraries, and sizable dataset of pictures containing a variety of fruits.
    - Each of these are readily available to the participating team members.

https://pytorch.org/

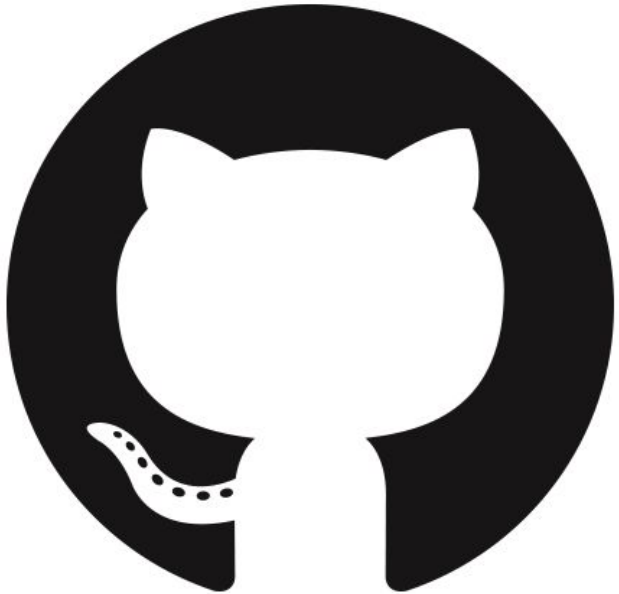
Financial Feasibility

- All the resources for this project were readily available with no cost to the team.
- The optional use of a CUDA capable GPU would accelerate training speeds, allowing more time for a more accurate product.
    - These GPUs are quite costly, ranging from $200 to $10,000 dollars

https://www.bhphotovideo.com/c/product/1046359-REG/nvidia_900_22081_2250_000_tesla_k40_gpu_accelerator.html

Organizational Feasibility

- The team consisted of 6 members with Computer Science backgrounds.
- Each member excelled in different areas pertinent to the project.
- Collectively, the team had experience with GitHub, Python, Java, C/C++, Web Development, Project Management, and Software Engineering Practices.

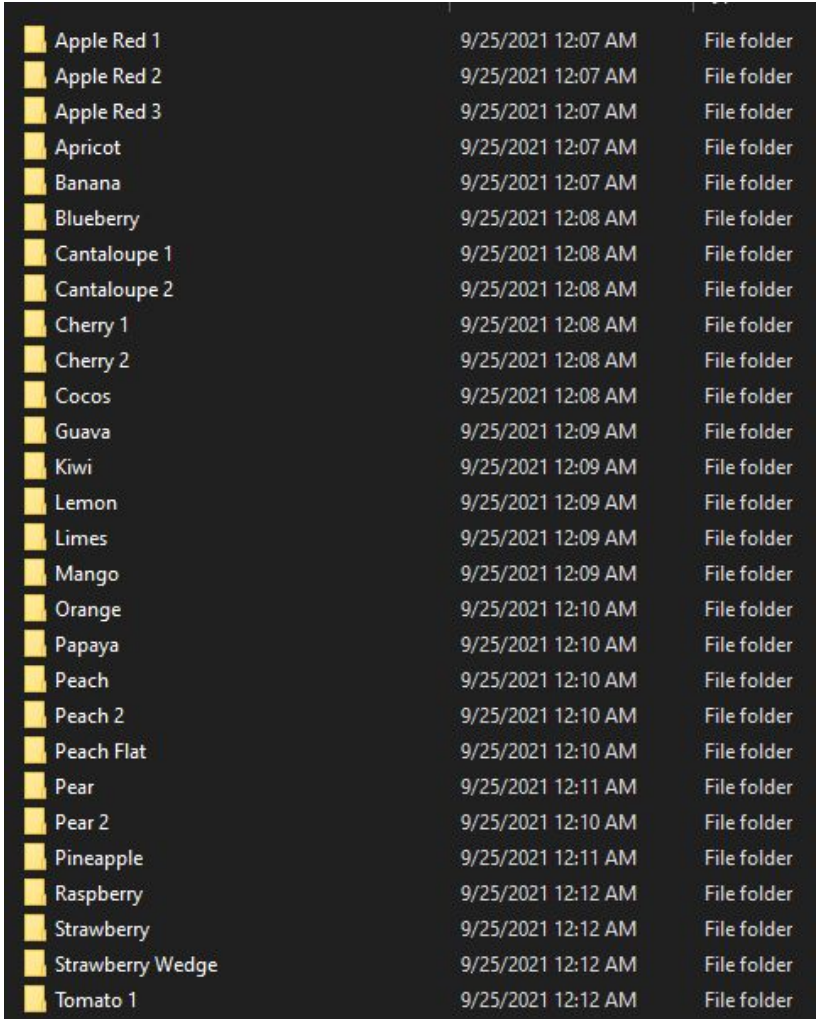


https://www.bhphotovideo.com/c/product/1046359-REG/nvidia_900_22081_2250_000_tesla_k40_gpu_accelerator.html

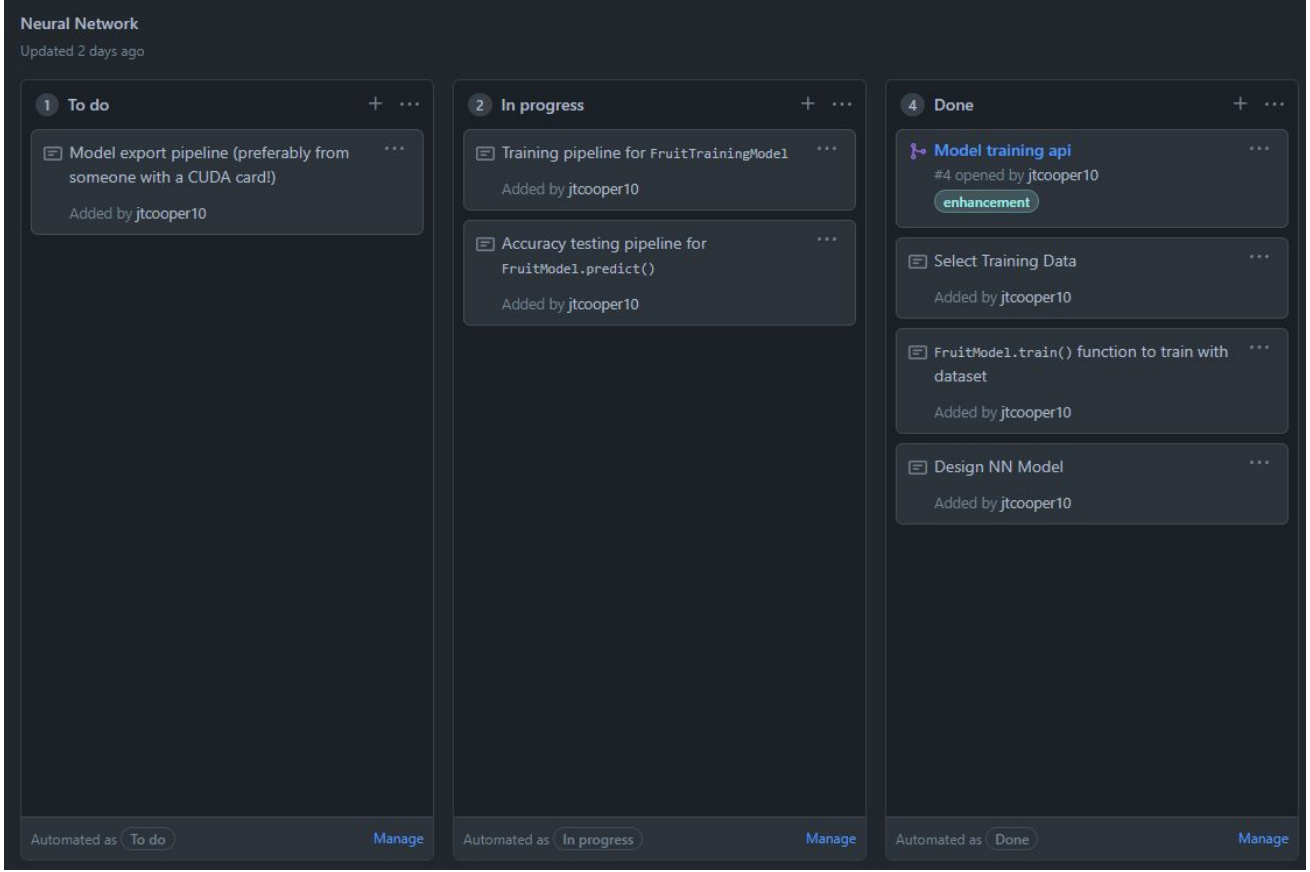
Detailed Design

Fruit Dataset

- Sourced from Kaggle, a website providing data science resources and services to software engineers.
- Directory of over 90,000 images including 131 various fruits
  - 1.3GB
  - Includes outliers
    - Fruits with varying backgrounds
    - Multiple fruits in a single image



Apple Red 1	9/25/2021 12:07 AM	File folder
Apple Red 2	9/25/2021 12:07 AM	File folder
Apple Red 3	9/25/2021 12:07 AM	File folder
Apricot	9/25/2021 12:07 AM	File folder
Banana	9/25/2021 12:07 AM	File folder
Blueberry	9/25/2021 12:08 AM	File folder
Cantaloupe 1	9/25/2021 12:08 AM	File folder
Cantaloupe 2	9/25/2021 12:08 AM	File folder
Cherry 1	9/25/2021 12:08 AM	File folder
Cherry 2	9/25/2021 12:08 AM	File folder
Cocos	9/25/2021 12:08 AM	File folder
Guava	9/25/2021 12:09 AM	File folder
Kiwi	9/25/2021 12:09 AM	File folder
Lemon	9/25/2021 12:09 AM	File folder
Limes	9/25/2021 12:09 AM	File folder
Mango	9/25/2021 12:09 AM	File folder
Orange	9/25/2021 12:10 AM	File folder
Papaya	9/25/2021 12:10 AM	File folder
Peach	9/25/2021 12:10 AM	File folder
Peach 2	9/25/2021 12:10 AM	File folder
Peach Flat	9/25/2021 12:10 AM	File folder
Pear	9/25/2021 12:11 AM	File folder
Pear 2	9/25/2021 12:10 AM	File folder
Pineapple	9/25/2021 12:11 AM	File folder
Raspberry	9/25/2021 12:12 AM	File folder
Strawberry	9/25/2021 12:12 AM	File folder
Strawberry Wedge	9/25/2021 12:12 AM	File folder
Tomato 1	9/25/2021 12:12 AM	File folder



Makeshift Kanban Framework

Removing Abstraction

- The detailed design phase was further progressed upon the creation of a github page, where the team made use of a Kanban framework within github's project view.
- Tracked progress of each sub-problem, such as the neural network model and front-end structure, thus decreasing abstraction with detailed descriptions of each project task.
- Consistent communication of the current state of the project, along with discussion of everyone's programming and contribution capabilities clarified each member's current responsibilities and tracked the progress of the program.

Coding & Testing

Coding

- In the first 2 sprints of the project, team members were able to locate an appropriate API, build a functional model, create a front-end user interface, locate a dataset, and familiarize themselves with python.
- We utilized a torchvision API (ResNet-50) sourced from *Deep Residual Learning for Image Recognition*, allowing for quicker overall production

`torchvision.models.resnet50(pretrained: bool = False, progress: bool = True, **kwargs: Any)` → torchvision.models.resnet.ResNet [SOURCE]

ResNet-50 model from "Deep Residual Learning for Image Recognition".

Parameters

- **pretrained** (*bool*) – If True, returns a model pre-trained on ImageNet
- **progress** (*bool*) – If True, displays a progress bar of the download to stderr

- Other team members focused on locating an appropriate dataset and building a front-end web interface.
  - Utilizing the Flask framework for webpage and python integration.

Image Recognition Model

- The team member responsible for generating the code for the neural network, Josh, wrote meticulous documentation for the model.
    - Ensured comprehension
    - Made work easier for other members when contributing.
    - Simplified the process of training & testing

```
@classmethod
def get_data(cls, root: "str",
             transform: "torchvision.transforms.Compose" = None,
             **loader_options) -> "tuple[DataLoader, dict[str, int]]":
    """
    Dataset import helper using PIL images for training/testing purposes.
    The classifications of each dataset must match during training, testing, and inference.

    :param root: Root directory where image classification folders are kept.
    Each sub-folder in the root directory should contain a single image category.
    The classification names are derived from the name of each folder.
    (It is highly recommended to use the EXACT same root folder structure for each model version)
    :type root: str

    :param transform: Optional torchvision transform.
    If not specified, the default transformer is used.
    (Option is only useful if experimenting with different transforms)
    :type transform: torchvision.transforms.Compose

    :param loader_options: Dictionary containing options to pass to the dataloader.

    :return: Pair of type (DataLoader, dict) where the dict maps label names to their corresponding ids.
    """
    if transform is None:
        transform = FruitModel.get_transform()
    img_set = ImageFolder(root=root, transform=transform)
    return DataLoader(dataset=img_set, **loader_options), img_set.class_to_idx
```

Importing the Data Set

The first step of training or testing is to import your dataset. This will make the image set available for use, as well as defining your labels.

Training data must be organized into folders, each of which representing one classification. For example, a folder called 'Apples' which contains only apple pictures, then a folder called 'Bananas', etc.

The root path passed into the `get_data()` function should contain all of folders, having the structure:

```
root/
  Apples/
    img1.jpg
    img2.jpg
    ...
    imgN.jpg
  Bananas/
    img1.jpg
    img2.jpg
    ...
    imgN.jpg
  ...
```

Note: importing a dataset is NOT NECESSARY for inference, only for testing and training

In [2]:
```
# To run this example, replace the path below with the path to your dataset.
# It is recommended to import twice, once for training and once for testing (with different data sets, obviously).

training_set, classes = model.fruit.FruitTrainingModel.get_data(root=r"D:\Documents\School\MAIN_FRUITS\PLAY",
                                                                batch_size=4,
                                                                shuffle=True,
                                                                num_workers=4)
# get_data() returns a dict mapping the label name to its id (an integer).
# For training, we need to convert this into an ordered list of strings.
classes = sorted(list(classes.keys()), key=lambda cls: classes[cls])
classes
```

Out[2]:
```
['Apple', 'Banana', 'Lemon', 'Limes']
```

Python Modules

class model.fruit.**FruitModel**(*labels: list[str], model: torch.nn.Module = None, threshold=0.6*)

Wrapper class for PyTorch model prediction.

Using a pre-trained PyTorch model, FruitModel objects can generate predictions based on image data.

The model may additionally be instantiated from a .pth file using from_file().

Once instantiated, it may be used for classifying images.

export(*path: str*)

Save the given model as a .pth file.

Parameters: **path** (*str*) – Absolute location of file to export to.

classmethod **from_file**(*path: str, labels: list[str], threshold=0.6*) → FruitModel

Import a model from a .pth file.

Parameters:
- **path** (*str*) – Absolute location of file to import.
- **threshold** (*float*) – Value on range (0.0, 1.0] where only predictions which meet the confidence threshold are considered valid. Any predictions whose confidence scores do not exceed the threshold are deemed invalid and are discarded.
- **labels** (*list[str]*) – Optional list of indexed label strings representing the names for each classification.

Returns: Model instantiated from the indicated .pth file.

classmethod **get_transform**() → torchvision.transforms.transforms.Compose

Generate the default PyTorch transformation for converting image data to a normalized tensor.

Returns: Instance of default Torchvision transformation object

predict(*img_data: Union[torch.Tensor, Image.Image], limit=True*) → dict[str, float]

Determine what classifications the given image belongs to and their probabilities. Either a pre-processed Tensor or PIL image may be provided.

Parameters:
- **img_data** – Image to generate predictions for. A PyTorch tensor is greatly preferred, but a PIL image can use default transformations to pre-process. An example of this usage: >>> from PIL import Image >>> model = FruitModel(labels[“example1”, “example2”]) >>> model.predict(Image.open(r”/path/to/image”)) <<< dict({ “example1”: 0.7 })
- **limit** (*bool*) – Indicate whether or not low-scoring predictions should be

classmethod **transform**(*data: PIL.Image.Image*) → None._VariableFunctionsClass.tensor

Generate the default PyTorch transformation and apply it against the given image data. Useful for one-off transformations, but if repeated transformations are needed, it is preferable to use get_transform to get a reusable transform function.

Parameters: **data** (*PIL.Image.Image*) – Image data to apply the default transformation against.

Returns: Transformed and normalized tensor data based on the provided image.

class model.fruit.**FruitTrainingModel**(*model: torch.nn.Module = None, threshold=0.6, labels: list[str] = None, optimizer=None, loss=None*)

detach() → model.fruit.FruitModel

Generate a new trained model suitable for predictions from the current model. All training methods will be disabled in the generated model, and is optimized for inference.

Returns: Non-trainable copy of the training model.

classmethod **get_data**(*root: str, transform: torchvision.transforms.Compose = None, **loader_options*) → tuple[DataLoader, dict[str, int]]

Dataset import helper using PIL images for training/testing purposes. The classifications of each dataset must match during training, testing, and inference.

Parameters:
- **root** (*str*) – Root directory where image classification folders are kept. Each sub-folder in the root directory should contain a single image category. The classification names are derived from the name of each folder. (It is highly recommended to use the EXACT same root folder structure for each model version)
- **transform** (*torchvision.transforms.Compose*) – Optional torchvision transform. If not specified, the default transformer is used. (Option is only useful if experimenting with different transforms)
- **loader_options** – Dictionary containing options to pass to the dataloader.

Returns: Pair of type (DataLoader, dict) where the dict maps label names to their corresponding ids.

run_epoch(*data: torch.utils.data.dataset.Dataset*) → float

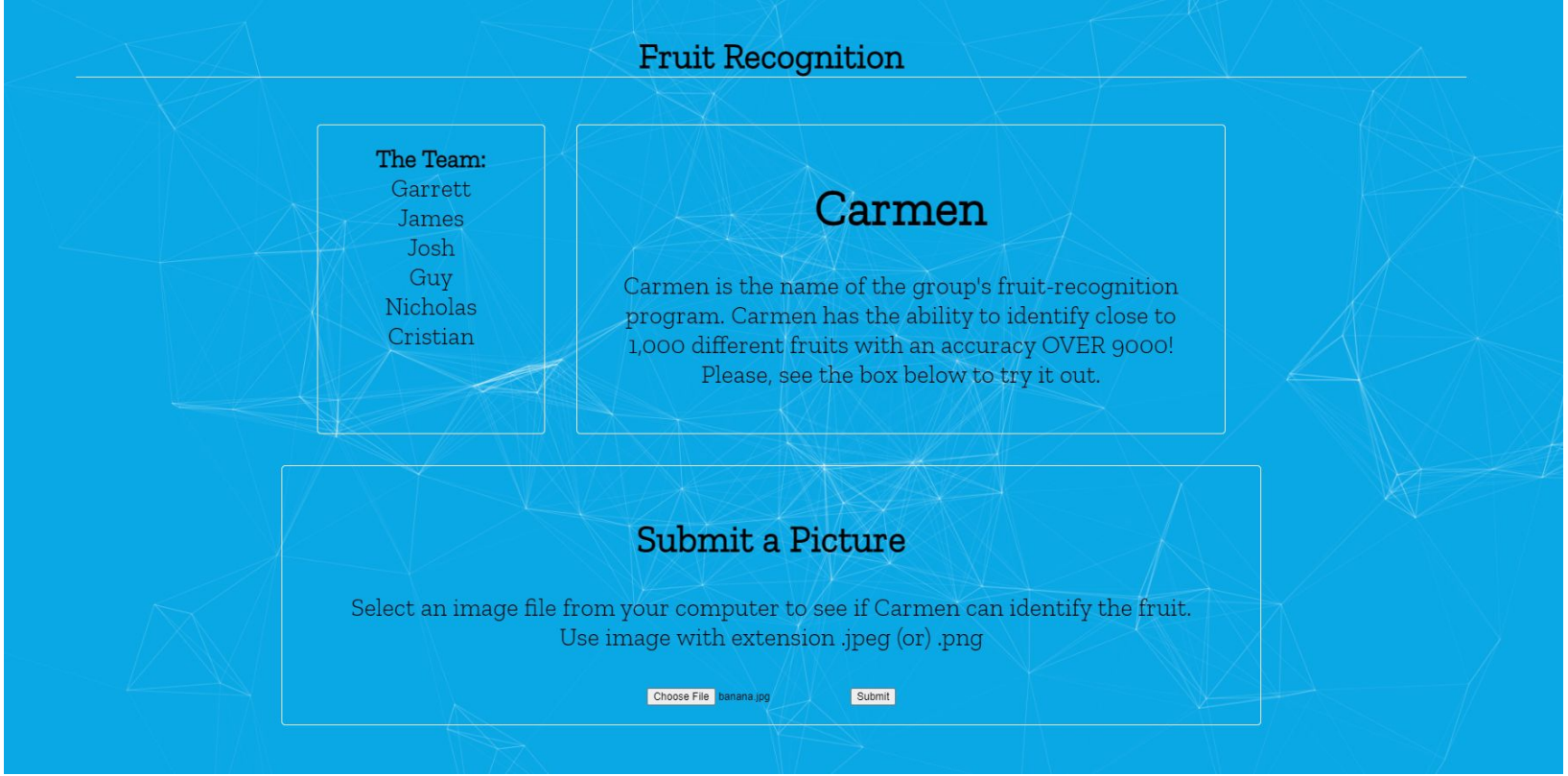
Performs a single training iteration on the model with the given PyTorch dataset. The labels provided by the dataset MUST match the instantiated model.

Parameters: **data** (*torch.utils.data.Dataset*) – PyTorch dataset with labels matching the model.

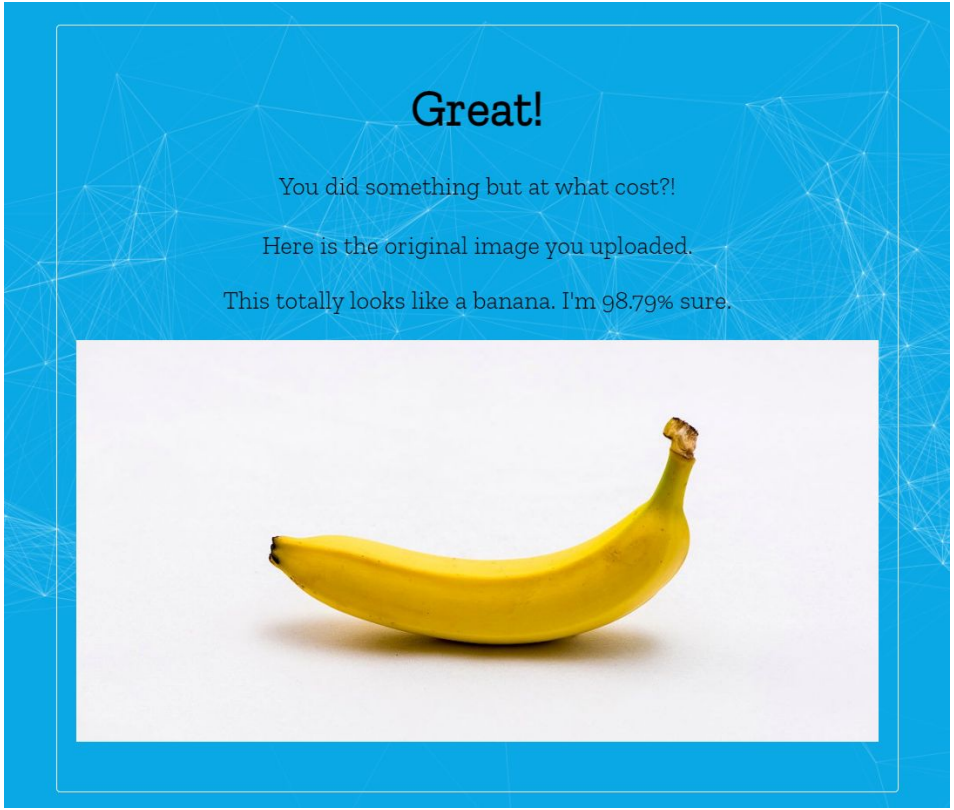
Returns: Float representing loss value of the iteration.

Testing

- Initial testing of the image recognition product displayed poor accuracy with poorly displayed results.
  - Result of poor structuring in the dataset, separate folders containing the same fruits, some images displaying flawed/outlying fruits.
  - Revisited the directory of images, merged and renamed folders of each labelled set of images.
  - The resulting product after this fix showed better performance in accuracy and display of results.
- Not enough time was remaining due to the lengthy process of training and testing to improve accuracy after end-user testing. No major issues were identified in end-user testing other than less-than-desired accuracy.



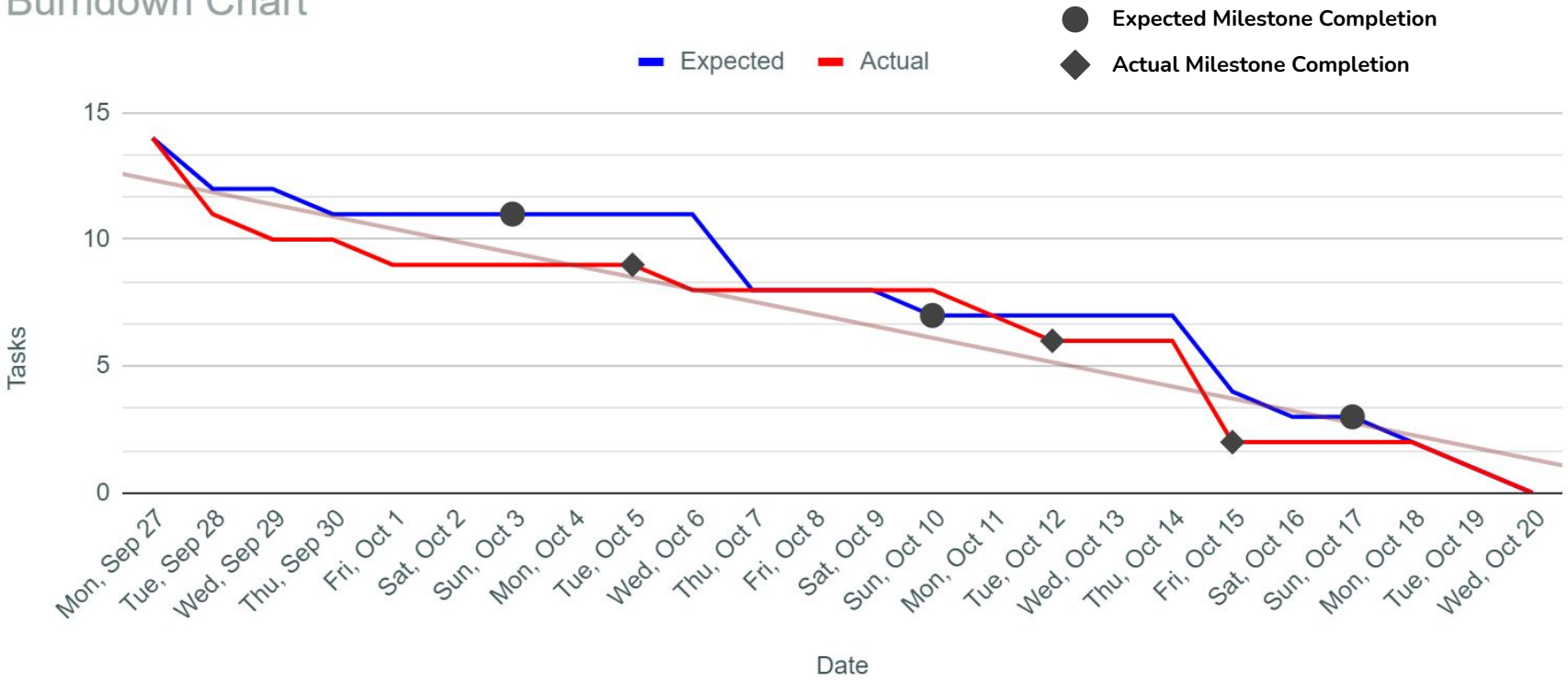
Webpage interface



Webpage interface

Burndown

Burndown Chart



Estimated timeline closely mirrors the actual timeline.
Product was functional on October 15th.

Burndown Analysis

- Most of the effort and focus on production took place after the second sprint ended on October 10th, where the team began working on creation & integration of the webpage and python script once the webpage was ready and the image recognition program gained basic functionality.
- The ‘final’ product of the image recognition program was officially established on october 15th.

- Translation from the Gantt chart to the burndown chart was not perfect, as some tasks named prior to production proved either redundant and/or unnecessary
- Some tasks took a much longer duration to accomplish than expected, such as generating the image recognition script and training/testing the neural network.

Burndown Analysis Continued

- Found that the training process, though it needed to be performed only once per version, took a very long time resulting in an inability to further refine the model in the time remaining before the deadline.
- The lengthy process of each task involved in production in addition to the student's total workloads outside of the project resulted in a product that functioned but displayed poorer accuracy than anticipated.
- Found that multiple member's lack of experience in python and other skills resulted in an uneven workload that bottlenecked possible production speeds.

Maintenance

Maintenance of the Fruit Identification Model

- Consists of
  - Updating the code to function with language libraries (Python and PyTorch)
  - Upkeeping web page functionality and hosting platforms
  - Accommodating newer libraries that provide more efficient resources for image processing.
- These maintenance considerations, however, are minimal.
  - Requires relatively little funding and effort.

In an officially implemented & commercial setting, maintenance of the program would also consist of hardware maintenance including cameras and computing equipment such as graphics cards, which would wear from sustained used over a long period of time.

Distribution of work between group members

Nicholas: Webpage design, Integration, sourced dataset.

Josh: Code documentation, Generated model, Sourced API, README.md.

Garrett: Documentation, Presentation, Generated webpage.

Guy: Training pipeline.

James: Gantt chart, Burndown Chart.

Cristian: Training pipeline.