Flower Identifier

Kevin Sakowicz: ksakowicz10@gmail.com
Margaret Evarts: maggie.evarts@yahoo.com

4/27/2024

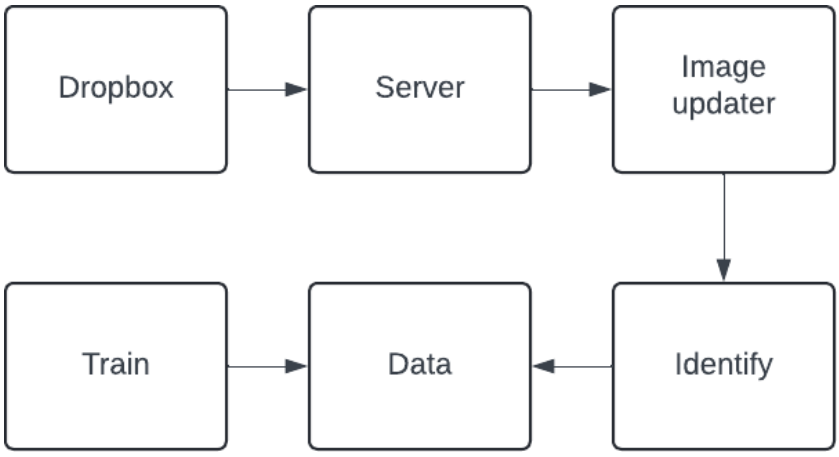
Report Specification

Overview:
Flower Identifier is a machine learning program for identifying flowers. By using drag and drop functionality with React, Python, and Flask, a concurrency system was implemented to streamline image processing.

Areas of Focus:
Kevin Sakowicz
- Handled majority of backend functionality
- Aided in frontend functionality
- Threading for POST Requests
- Researched TensorFlow's library
- Explored Keras with models, layers, and preprocessing images
- Implemented train data and identify image
- Tested different implementations to find max efficiency

Margaret Evarts
- Handled majority of frontend functionality
- Implemented drag and drop with a submit option
- Helped with backend functionality
- Threading for POST Requests
- Developed test plan
- Ran and recorded results for accuracy/evaluation

Analysis / Design:
Once the setup was completed with the base Flask framework, the frontend and backend were worked on simultaneously. The front end began with a simple drag and drop method to upload images from a file system. This led to the creation of a submit button to have the photo results displayed under a "Submitted files:" sub-heading. Before the results are shown, the image is sent to the backend via post requests through the server. For the backend, the TensorFlow and Keras libraries were used to process and categorize the images. These results are what are shown next to each image in the front end.



Test Plan:

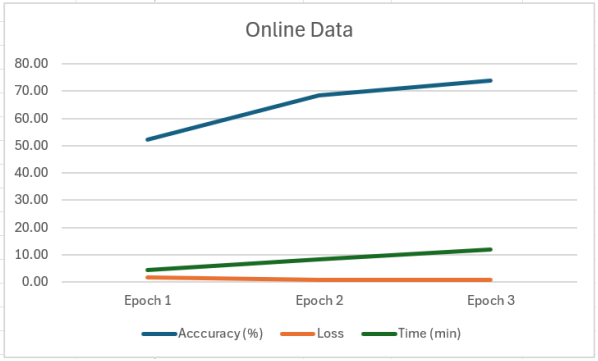
- Create base for Flask framework
- Develop a drag and drop for the front end
- Implement a submit button for the POST Request
- Include POST Request so each image is a separate thread

- Run the training model with keras.models, keras.applications, keras.layers, keras.preprocessing.image
    - Run for one epoch
    - Observe and record the accuracy and loss for the one epoch, and total amount of time
    - Run for two epochs
    - Observe and record the accuracy and loss for the second epoch, and total amount of time for all two
    - Run with three epochs
    - Observe and record the accuracy and loss in the third epoch, and total amount of time for all three
- Repeat this three times and record the average for each variable (accuracy, loss, time)
    - Accuracy and time should increase with each epoch
    - Loss should stay consistent
- Repeat this, but instead use TensorFlow's online dataset
    - Similar results, but with better accuracy, less loss, and more time

- Use the more efficient trained data to develop an identifying model for sent-in images
- With a baseline of 3 epochs from the training model, run the identifying file with keras.models and PIL
    - Send in two images (different flowers) through the front end submit button
    - Observe and record if they are given the correct result (flower name matches flower images)
    - Measure and record the total amount of time from hitting submit to seeing results
- Repeat this four times with the same two images and record the average result accuracy and time for each
    - Accuracy should be between 60-70%
    - Time should be between 2-3 seconds

Project Results:

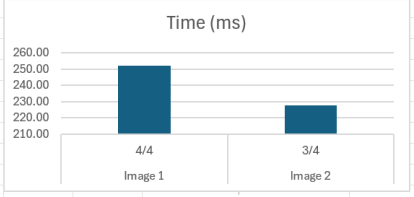
	Train	Personal	Data			Train	Online	Data	
Trial	Epoch	Accuracy (%)	Loss	Time (min)	Trial	Epoch	Accuracy (%)	Loss	Time (min)
1	1	17	9.48	0.35	1	1	52	1.61	4.3
	2	18	8.97	0.53		2	71	0.8	8.47
	3	21	9.2	0.95		3	77	0.6	12.13
2	1	18	8.34	0.4	2	1	53	1.59	4.1
	2	20	9.89	0.63		2	69	0.82	8.05
	3	22	9.76	1.02		3	73	0.63	12.01
3	1	19	9.22	0.38	3	1	52	1.63	4.4
	2	20	9.16	0.7		2	65	0.79	8.51
	3	21	8.95	0.97		3	72	0.59	11.95
	Average	Acccuracy (%)	Loss	Time (min)		Average	Acccuracy (%)	Loss	Time (min)
	Epoch 1	18	9.01	0.38		Epoch 1	52.33	1.61	4.27
	Epoch 2	19.3	9.34	0.62		Epoch 2	68.33	0.8	8.34
	Epoch 3	21.67	9.3	0.98		Epoch 3	74	0.61	12.03



The results were about as expected. For the personal data set, there was an increase in time and accuracy when increasing the epochs while the loss stayed consistent.

Similarly, the online data set also yielded an increase in accuracy and time as more epochs were added with the loss staying consistent. However, this data set was much larger, which significantly increased the extent of our training. This led to a notable increase in time from the personal data set, but also led to a huge difference in accuracy and loss, as well. Accuracy was much higher, and the loss was much lower than before.

	Identify		
Trial	Image	Correct	Time (ms)
1	1	Yes	261
	2	Yes	198
2	1	Yes	235
	2	No	283
3	1	Yes	296
	2	Yes	237
4	1	Yes	217
	2	Yes	193
	Average	Correct	Time (ms)
	Image 1	4/4	252.25
	Image 2	3/4	227.75



The results were as expected, with the time being a little lower than expected. The average amount of time it took for each image was around 235 milliseconds. The accuracy was a little higher than originally calculated, with the first image having the correct result 100% of the time and the second image having the correct result 75% of the time. With a larger scale of tests, these numbers might fluctuate to be a bit more accurate but should still be somewhere in the range of 75%-100%.

Lessons Learned:

When dealing with machine learning, a lot must be considered while taking the end goal into perspective. The benefit of each trade-off is important to consider when testing implementation. Is it cost-efficient to have less time but less accuracy, or better to have more accuracy with a longer running time? These are the kinds of questions we had to ask ourselves while implementing the training side of our model.

We learned that, because we want to have more accuracy and less loss with our build, it would be best to have the trainer take a few minutes to categorize the online data set with a few epochs. This was chosen over our personal data set which, although had a significantly less amount of time, also had a significantly less amount of accuracy and much more loss.

By doing this, we learned that threads can greatly help to streamline the process when dealing with machine learning projects or activities. Without threading, image processing would be sequential, and each image would have to be processed one after another. By utilizing threads, multiple images can be processed concurrently, which reduces the overall processing time.

In this case, multiple flower images can be simultaneously processed with the results of each sent back much faster. Also, by offloading image processing to separate threads, the Flask server remains responsive. Without threads, processing multiple images sequentially could result in the server appearing unresponsive to users until all images are processed.