Distributed Ensembles for Image Classification

Andrea Burns Gavin Brown Iden Kalemaj

- **Github link:** https://github.com/gavinrbrown1/distributed-ensemble-project
- **Slice name:** Project5

1 Introduction

The general goal of image classification is to determine which of several fixed classes an image belongs to. One common benchmark is the CIFAR-10 dataset [2], which contains 32×32 pixel images of ten different classes: airplanes, automobiles, birds, cats, deer, dogs, frogs, horse, ship, truck. We've included a selection of these images in Figure 1. Image classification is an important machine learning task, as it is an initial step toward visual scene understanding.

Ensembling is a popular way to improve the performance of a machine learning task; an ensemble is a set of different classifiers that are 'combined' to make a collective, better decision on the task. Such models are also sometimes called "committees;" for a reference see Chapter 14 of [1]. In this experiment we distribute 4 independent classifiers across different network nodes (servers) and take a majority vote of their decisions as the final classification result. This distributed system allows us to investigate latency versus accuracy: by only taking the votes from classifiers that have responded within a certain timeout period, we force decisions to be faster but possibly lose accuracy. Lastly, we try hand-crafted caching tools that can act as short-cut decisions, to see the impact on end-to-end delay.

A client wishing to receive classification for an image connects to a manager and sends the image to the manager. The manager then checks if the cache can be utilized before sending copies of the image to the classifiers. If the image already exists in the cache, or a close enough image exists, then it can simply return the cached decision. Otherwise, the manager will start communication with all four classifiers and will aggregate their results into a majority vote. This decision will then be added to the cache and communicated back to the waiting client.

Figure 1: A few images from CIFAR-10.

The learning outcomes of this project include: greater experience with the featured aspects of computer networking like sockets, caching, and coordinating distributed systems. We also hope to improve our ability to collaborate on building moderate-scale computer systems from near-scratch, learning more about how to design our solutions. Finally, we hope to better understand how distributed approaches may benefit machine learning prediction systems.

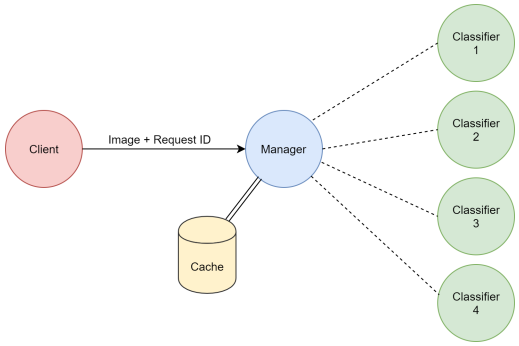

Figure 2: A high-level representation of the GENI architecture setup.

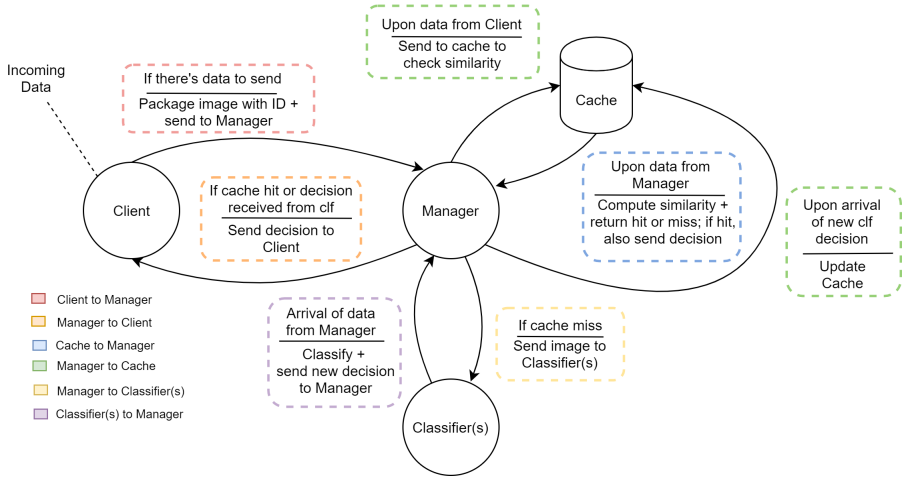

Figure 3: Finite State Machine characterizing the behavior of our system.

2 Experimental methodology

2.1 Caching and Sampling

The two key components of the cache are how we compare a new image from the client to the existing cache images, and how we choose to update the cache over time. To compare a new image to the cache images, the Zero-Normalized Cross-Correlation coefficient (ZNCC) is used. This is a image similarity metric that is invariant to brightness differences; its values range $[0, 1]$, where a ZNCC value of one means the two images being compared are the same. We deem two images similar enough to use the cached decision if their ZNCC is greater than 0.9. In the case that this threshold is not reached, we do not use the cache and have to communicate with the classifiers. Any time there is a cache miss, the new image is going to be added to the cache once the ensembled decision arrives. We experiment with cache sizes of 15 and 30 to see the effects on latency.

Once the cache is full, we decide to kick out the **pair** of closest images in terms of L2/Euclidean distance. We can think of this pair as being the least informative, since they are most similar to each other. Pairwise distances are only updated as samples are added or removed to the cache, never redundantly re-computing pairwise distances.

To evaluate the impact of caching on latency, we consider two models for generating image requests from the client. In the first model images are sampled uniformly at random. In a real system, though, some images may occur far more than others, such as security systems evaluating the same fixed scene. To model this we also sample from a "long-tailed" Zipf distribution so some image requests will be very frequent over time and the cache will be more useful. We add a fixed delay to all classifiers to simulate additional delay between machines.

2.2 Corruption and Delay

One of the benefits of ensembling, from a networks perspective, is the resilience to delays and corruption experienced in the communication between servers. Specifically, if one or some of the classifiers timeout, the manager can use the decision from those classifiers that did not timeout and still send a response to the client. Similarly, if one or some of the decisions form the classifiers gets corrupted, the use of the majority vote guarantees that the decision of the manager will not lose too much in accuracy. Our experiment considers the following parameters:

- A random delay experienced by three classifiers – three classifiers draw delays from an exponential distribution with a specified mean, on top of the fixed delay. The fourth classifier does not experience any delay, to guarantee that the manager receives at least one decision before timeout.

```
(base) ikalemaj@client:~/distributed-ensemble-project/client$ python client_image.py 8889
answer = GOT SIZE
Sending ID: 0.0.0
answer = GOT ID
Sending image...
answer = Image is of class 3
answer = GOT SIZE
Sending ID: 0.0.0
answer = GOT ID
Sending image...
answer = Image is of class 0
answer = GOT SIZE
Sending ID: 0.0.0
answer = GOT ID
Sending image...
answer = Image is of class 3
```

Figure 4: Messages printed on client during experiment

- A corruption probability – each classifier experiences corruption, i.e. it sends a flipped decision to the manager, different from the real decision, with some probability. We consider the following corruption probabilities: 0.0, 0.1, 0.2, 0.3.
- A timeout – the manger waits for a response form each classifier for 3 seconds only and closes the connection if there is a timeout.

In our experiment we consider the following relationships:

- How does accuracy of the classification change with the corruption probability?
- How does end-to-end communication time change with random delays?
- How does accuracy change with the random delay introduced between manager and classifier?

3 Results

3.1 Usage Instructions

Our system is designed to be used as part of an automated system, so the process are all started from within scripts. Once the resources are set up on GENI, the user specifies the parameters of the experiments they wish to run in a .csv file residing in the client. The user then starts a script on each machine using the command: python [insert script name] [insert port number]. Once the scripts are running, messages are printed on each node to indicate the communication flow and messages being exchanged, as demonstrated in 4. Refer to the instructions document for a detailed description of how to setup experiments and tweak other parameters.

3.2 Analysis Results

As you can see in Figure 5, our system's behavior changed significantly with different parameters. On the top left, we can see that decreasing the reliability of our individual classifiers reduced the overall accuracy, but by a lower amount: even when each classifier had a 30% chance of answering randomly, our average accuracy was over 80%. On the top right we see how random delays affect communication time: as expected, they increase it! On the bottom left we see how random delays affect accuracy (through the timeout mechanism). This is a little hard to interpret - it's possible more experiments are needed. On the bottom left we visualize how the cache is used over time: while the uniform distribution rarely gets used, the power law linearity means roughly one in three requests hits the cache. Table 1 shows that the uniform sampling utilized the cache at a lower rate than the power law. The smaller cache was faster, possibly because of the overhead of recomputing.

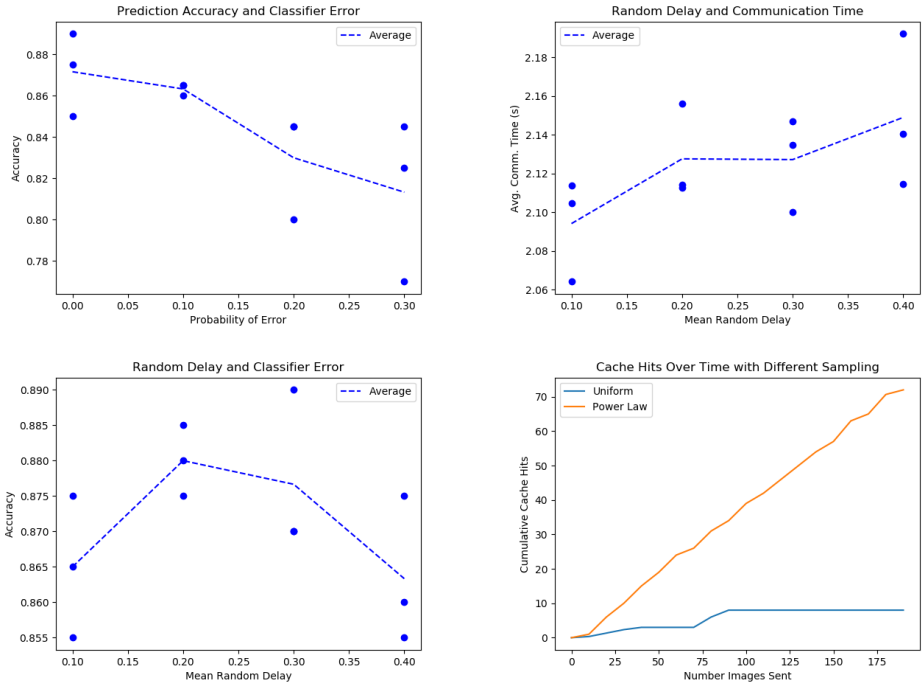

Figure 5: Results from various experiments, as described in the text.

4 Conclusions and Future Extensions

Cache	Unif.	Power
15	1.69	1.27
30	2.11	1.53

Table 1: Average communication times, in seconds.

We found that developing this system and running the experiments was a very rewarding process. It was enjoyable to see the natural way our research interests of vision, machine learning, and probabilistic algorithms mapped neatly onto the concepts we've learned in class. We also learned a lot through the process of pulling together the many components and defining our own protocols to interact across machines. The results mostly matched what we expected, showing clear tradeoffs between accuracy, space requirements, and latency.

Several natural extensions present themselves. Our cache was not efficient enough to speed up the classification: this could be improved by making more efficient decisions or ordering the images. Currently we use a fixed number of classifiers, but we could extend our system to gracefully scale up and down as machines come on and offline. From the point of the view of the user, we used only one client in order to highlight the effects of the cache and unreliable classification. We could extend our manager to handle concurrent requests from multiple clients, and set up the classifiers to process images sequentially instead of one over each connection. Our model also currently sets up a unique TCP connection for each image, but depending on the client's needs it may be more efficient to keep one connection for several images.

5 Division of labor

Andrea, Iden, and Gavin decided on the project topic and wrote the proposal and report together. All high-level decisions were made together: network topology, communication protocols, experiments, etc. Much of the coding was done in pairs, as well. The focus of specific sections were as follows: Andrea created the code to control caching. Iden wrote the bulk of the socket code that allowed the machines to communicate. Gavin wrote the code to handle the prediction as well as modeling error and delay on the part of the classifier.

References

[1] Bishop, Christopher M. Pattern recognition and machine learning. Springer, 2006.

[2] Krizhevsky, Alex, and Geoffrey Hinton. Learning multiple layers of features from tiny images. Vol. 1. No. 4. Technical report, University of Toronto, 2009.