Patrick Darrow
EE250, SPR2020

End-to-end IoT System

1. **Overview**
2. **Architecture**
   - 2.1. **System**
   - 2.2. **Model**
3. **Performance**
   - 3.1. **Data Transfer**
   - 3.2. **Model Inference**
4. **Improvements**
5. **Conclusion**

---

1. Overview

This project was motivated heavily by past experience with edge devices, where it was imperative to understand tradeoffs between onboard and offboard computing. It imitates a system of at least two devices, one (or more) nodes that fetch data (which will be referred to as the fetching node[s]), and another (which will be referred to as the compute node) with access to significantly greater compute power, and data storage capabilities.

I imagined a setup where the compute node requires an image of a target class, so it makes a request to a number of fetch nodes. It is assumed that there exists a non-negligible degree of uncertainty among the fetch nodes as to whether or not the image sources can be trusted, but they don't have the GPU compute power to perform AI inference for verification. Therefore, they upload the image, and retrieve the URL for the image and return it to the compute node, where AI image classification is used to determine whether or not the retrieval was a success. In this toy case, the images requested and classified by the compute node are all of cats or dogs (Fig. 1).

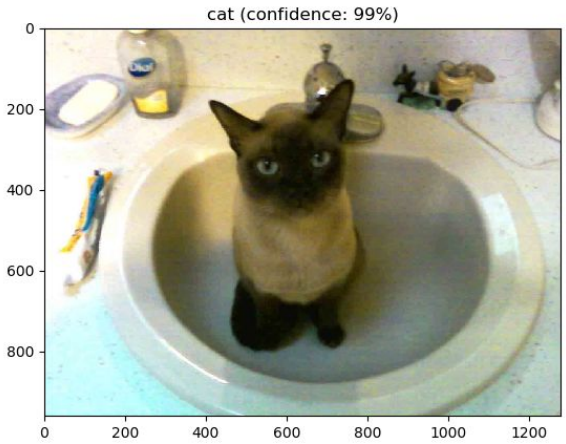

Fig. 1: The compute node classifies a cat with 99% confidence.

2. Architecture

2.1. System

Both nodes communicate via the MQTT publish-subscribe network protocol. In the test case, the default Mosquitto test broker was used. MQTT allows for multiple fetch nodes to retrieve and publish data on independent timelines, reducing the complexity of organizing potentially many fetch nodes. The compute node (shown as "Server pubsub", Fig. 2) publishes string image requests to the "paho/animal" topic, either "cat" or "dog".

The fetch nodes (shown as "IoT/Device pubsub", Fig. 2) are subscribed to "paho/animal" and await valid requests. Upon receiving a valid request, they look to

their sources for images. In practice, this could be a device in the field that takes and uploads a picture using an onboard camera. In the test environment, a fetch node makes a call, using Python's requests package, to one of two APIs: *thecatapi.com* or *dog.ceo*, which return random pictures of cats or dogs respectively. It should be noted that the role of the fetch node would be unnecessary if all nodes were to fetch from the same source(s). This setup presumes that nodes are fetching from unique sources. After a successful image capture, the URL is published by the fetch nodes to "paho/classify".

The compute node is subscribed to "paho/classify" and awaits URLs. Using the public Python libraries requests, PIL, and sci-kit image, the image is downloaded from the URL and the image pixel data is extracted. The image is resized and preprocessed using OpenCV, which, importantly, uses an interpolation scheme that preserves picture quality. The compute node then leverages its GPU compute power to produce predictions on the image class (cat or dog). These predictions and the image itself are then visualized using Matplotlib. In practice, this visualization may be used by a human in the loop to validate low confidence images, but in test it is most useful for debugging and aesthetics.

2.2. Model

The base model used for image classification is the InceptionV3 architecture by Google. The model head, or output layer was removed and replaced with a two neuron output for the cat and dog classes. The model was trained locally with Keras, sampling from 25,000 unique, color images of cats and dogs until the validation loss converged at around 0.02. This was an arbitrarily decided stopping point.

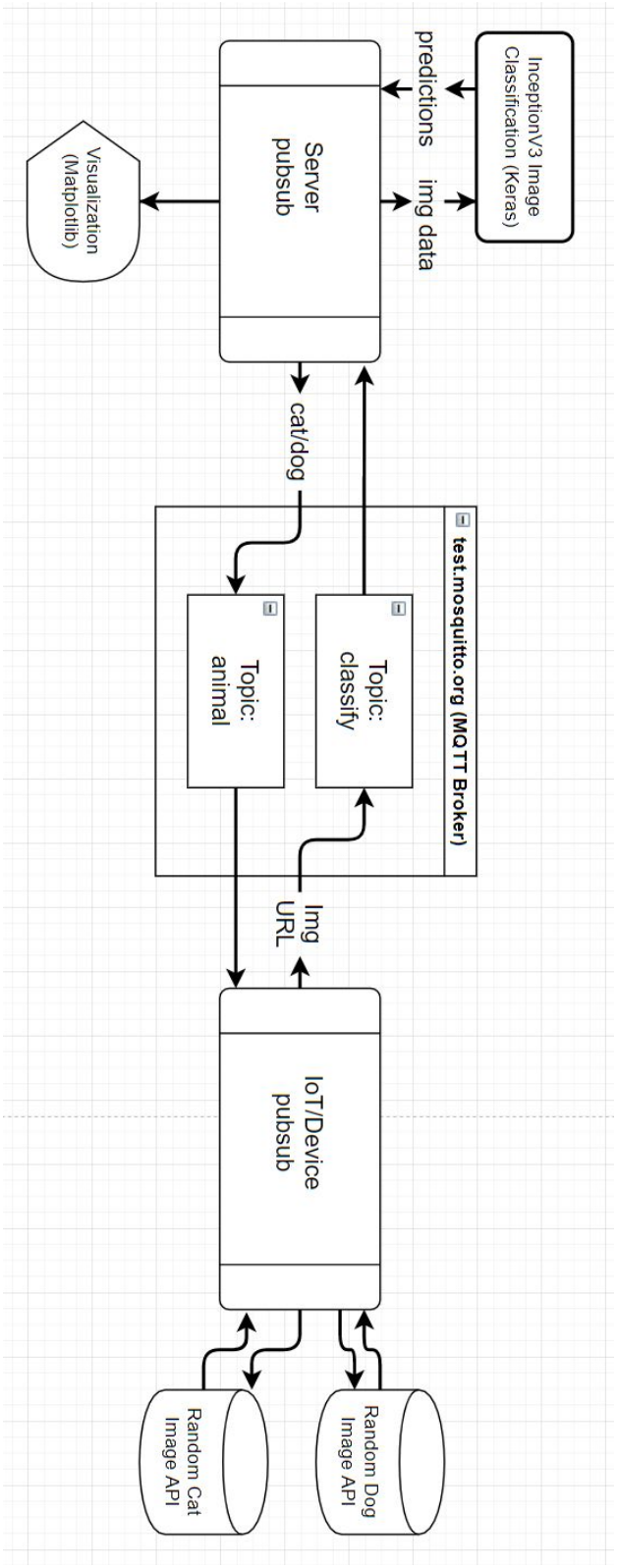

Fig. 2: High level system architecture overview

3. Performance

In the test environment, significant sources of latency include: 1) data transmission between the nodes and APIs, the MQTT broker, the images' host, and the GPU, 2) model inference. In practice, the system would handle best with sparse requests, as loading/resizing images is expensive. In the current test environment, latency varies widely, from 100ms to 1.5s. Sources of variance include GPU preparedness (there is overhead upon startup, depending on the hardware and ML framework) and API/broker interactions.

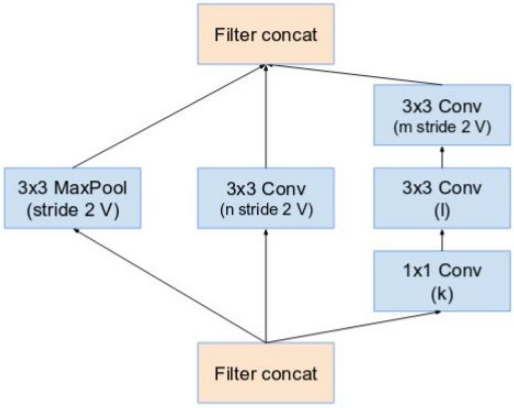

Fig. 3: Inception blocks concatenate features from multiple convolution blocks, trading latency for improved accuracy

The model itself performs well, with 96% validation accuracy after only a few epochs. At the time of its invention, the model achieved state of the art performance with the use of so-called inception blocks (Fig. 3). Inception blocks improved accuracy, but contributed significantly to the over 23 million parameters, making it far from the fastest in terms of computation; however for non-real-time applications, the latency is not likely to be a bottleneck, particularly when using modern GPUs.

4. Improvements

Currently, the model supports only single image prediction batches. In higher volume environments, it would be imperative that the inference occurs on larger batches. A queue could be used to hold incoming images until a number of frames has been reached, or an amount of time has passed. Furthermore, to improve latency, model pruning would be extremely effective, considering the large number of starting parameters.

The model performed well with minimal training. It should however be stated that, with common techniques, accuracy for such a model can improve. Using more data, a learning rate scheduler, more sophisticated image augmentation (only horizontal reflections were used), and transfer learning with Imagenet weights, would all contribute to improved accuracy.

This framework scales well with MQTT as the underlying network protocol, particularly as the number of topics and/or nodes increases. However, alternative frameworks may be worth exploring for a single topic, single compute node setup.

5. Conclusion

The proposed framework performs well for sparse, non-real-time, asynchronous data retrieval and classification tasks, in terms of both latency and accuracy. Overall, the project was great in terms of exploration and as a proof of concept, and is something that will continue to be worked on.