FlexServe: Deployment of PyTorch Models as Flexible REST Endpoints

Edward Verenich\textsuperscript{1,2}, Alvaro Velasquez\textsuperscript{2}, M.G. Sarwar Murshed\textsuperscript{1}, and Faraz Hussain\textsuperscript{1}

\textsuperscript{1}Clarkson University, Potsdam, NY
\textsuperscript{2}Air Force Research Laboratory, Rome, NY
Scenario

• Need on-demand image classification / object detection from sensor imagery.
• Inference capability needs to ‘plug-in’ into existing systems.
• Rapidly changing target classes with possible geometric and photometric variations.
• Need for dynamic classification sensitivity threshold to control number of false negatives.
Why PyTorch Models?

• Large collection of pretrained PyTorch CNNs used as base models for convolutional feature extraction and transfer learning.

• PyTorch’s dynamic graph architecture makes the network easier to debug.
Frequent Model Retraining and Deployment

- Target classes change frequently -> use transfer learning for novel classes with limited training samples.
- Images from sensors may contain geometric and photometric variations (noise, angle, lighting, etc.) -> train several architectures for the same target class to take advantage of different inductive biases.
- Inference service must be decoupled from the larger system -> deploy trained models as REST services.
Options for Deployment

• TensorFlow Serving – requires a two-phase conversion process to an intermediate format (ONNX) and then a static computational graph. Not all PyTorch features are properly translated (nn.Module subclassing for ensemble models).

• KFServing for Kubernetes – requires that Kubernetes be installed, depends on a separate ingress gateway.

• TorchServe (April 2020) – released after FlexServe by AWS and Facebook, requires Java 8 (not always permitted)
FlexServe

• Minimal dependencies outside of PyTorch due to restrictive and heterogeneous host computational environments.
• Deployment of multiple models behind a single endpoint.
• Multiple models share GPU resource for inference.
• Accept flexible data batch sizes during inference.
• Adjust inference endpoint sensitivity dynamically.
FlexServe Model Ensembles in Shared GPU Memory

- Multiple architectures trained for target class.
- Shared GPU memory for ensemble inference.
Flexible Batch Size Inference Endpoint

- Pool 1 to N samples in a single request, each model in the ensemble returns batched inference results.

```
HTTP POST

HTTP POST

FlexServe Model Ensemble as a Service

JSON

Requesting Application

{ 'model 1': ['ALPHA', 'ALPHA', 'BETA'], 
  'model 2': ['ALPHA', 'BETA', 'ALPHA'], 
  'model N': ['ALPHA', 'ALPHA', 'BETA'] }
```
Ensemble: Mixture of Experts and Sensitivity

• Consuming applications have different detection sensitivity policies that depend on target classes, surveillance area, etc.

• Ensemble inference allows the consuming application to employ its own sensitivity policy by combining output from FlexServe ensemble responses as needed (vote, average, probability, threshold, etc.)

App Client 1 (vote policy)
App Client 2 (threshold policy)
App Client 3 (average policy)

FlexServe Ensemble Models as a Service
Takeaways

• Existing PyTorch model deployment solutions did not meet our requirements for rapidly fielding RESTful inference services.
• FlexServe was designed to be a lightweight, flexible, and open deployment module for the growing PyTorch user community.
• Major contributors (Amazon, Facebook) to the PyTorch ecosystem recognized some of these limitations and released a scalable model deployment library (TorchServe).
• FlexServe is still the simplest solution with minimal requirements that is suitable for a wide range of computational environments.
Questions

- Edward Verenich – verenie@clarkson.edu, edward.verenich.2@us.af.mil
- Alvaro Velasquez – alvaro.velasquez.1@us.af.mil
- M.G. Sarwar Murshed – murshem@clarkson.edu
- Faraz Hussain – fhussain@clarkson.edu

https://github.com/verenie/flexserve