CS 181AI
Lecture 16

ML System Resources: Memory

Arthi Padmanabhan

Mar 20 2023

Logistics

- Final group projects and start up info were sent out this morning. Please start a Slack channel (or any form of communication) with your group
- Wednesday: working day for proposals
- Proposals: due Monday 3/27 (template is on course webpage)
- Assignment 4 due Friday

Today

- GPU memory is often a significant bottleneck
  - Demo: learn how to assess model memory usage for loading and running. Compare across models
  - Paper: merging models to lower memory usage

Memory Demo

- Open lec16.ipynb

GPU Memory

- This is often a major bottleneck, both for training faster and for being able to run inference on several models on one GPU
- Even if they're not running at the same time, this is hard.
  - Moving them back and forth between CPU and GPU is very slow, so we'd rather have them stay in GPU
  - We saw that models take memory just to stay in GPU

CUDA Out of Memory

We use the LR Finder to pick a good learning rate.

In [11]:

```python
import gc
import torch
torch.cuda.empty_cache()
gc.collect()
learn.lr_find()
```

```
~/miniconda3/envs/lesson3-planet/lib/python3.7/site-packages/torch/nn/modules/module.py in __call__(self, *input, **kwargs)
    487             result = self._slow_forward(*input, **kwargs)
    488         else:
--> 489             result = self.forward(*input, **kwargs)
    490         for hook in self._forward_hooks.values():
    491             hook_result = hook(self, input, result)

~/miniconda3/envs/lesson3-planet/lib/python3.7/site-packages/torch/nn/modules/conv.py in forward(self, input)
    318     def forward(self, input):
    319         return F.conv2d(input, self.weight, self.bias, self.stride,
--> 320                         self.padding, self.dilation, self.groups)
    321
    322

RuntimeError: CUDA out of memory. Tried to allocate 106.38 MiB (GPU 0; 1.96 GiB total capacity; 959.86 MiB already allocated; 121.25 MiB free; 5.14 MiB cached)
```

Today’s Paper: GEMEL

- Problem: when several models on one GPU, we often run out of GPU memory
- Simple solution: swap models in and out of GPU memory – we know this takes a long time
- Our solution: Can we merge redundant layers across models to lower memory usage of the whole workload?

Memory Usage in GPUs

- Model is a sequence of layers
- Layer = definition + weights

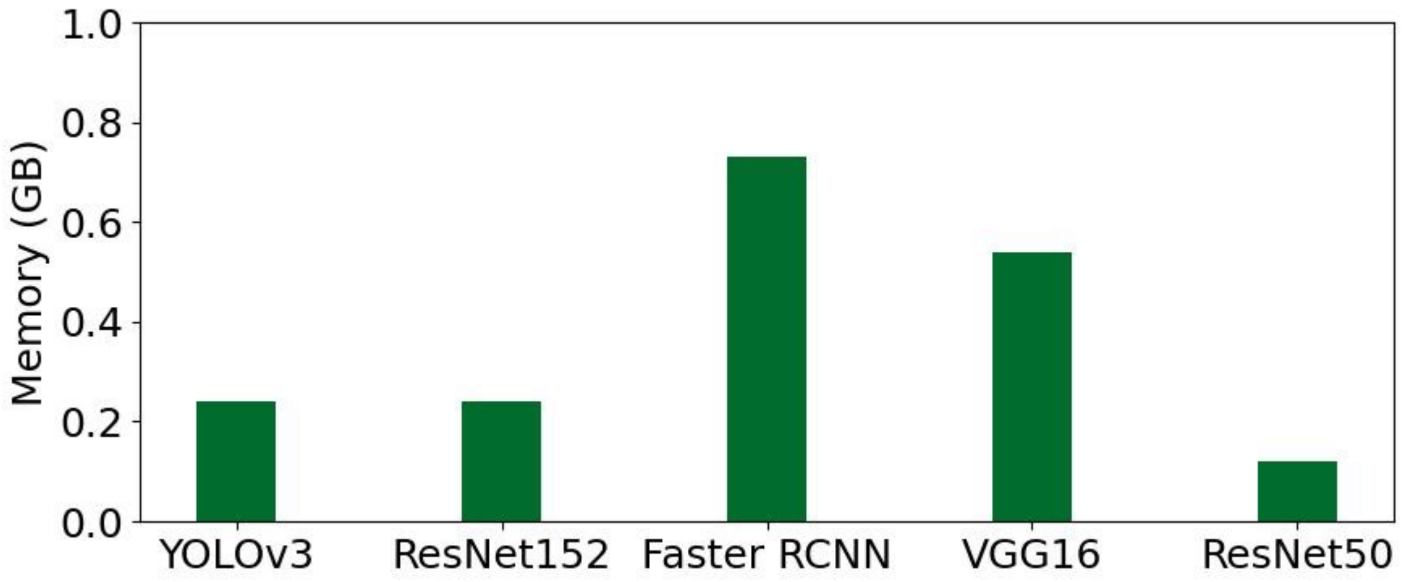
Convolutional Layer
(inputs=256, outputs=512, kernel=(3,3), stride=(0,0), padding=(0,0))

+

```
tensor([[-0.0295,  0.0185, -0.0011,  ..., -0.0160, -0.0060, -0.0183],
        [-0.0190, -0.0122,  0.0016,  ..., -0.0228,  0.0047,  0.0212],
        [ 0.0061,  0.0166,  0.0058,  ...,  0.0174, -0.0241, -0.0285],
        ...,
        [-0.0043, -0.0456, -0.0287,  ..., -0.0237,  0.0192, -0.0271],
        [-0.0344, -0.0279, -0.0188,  ...,  0.0160, -0.0026, -0.0185],
        [-0.0196, -0.0388, -0.0106,  ...,  0.0067,  0.0138,  0.0164]],
       device='cuda:0')
```

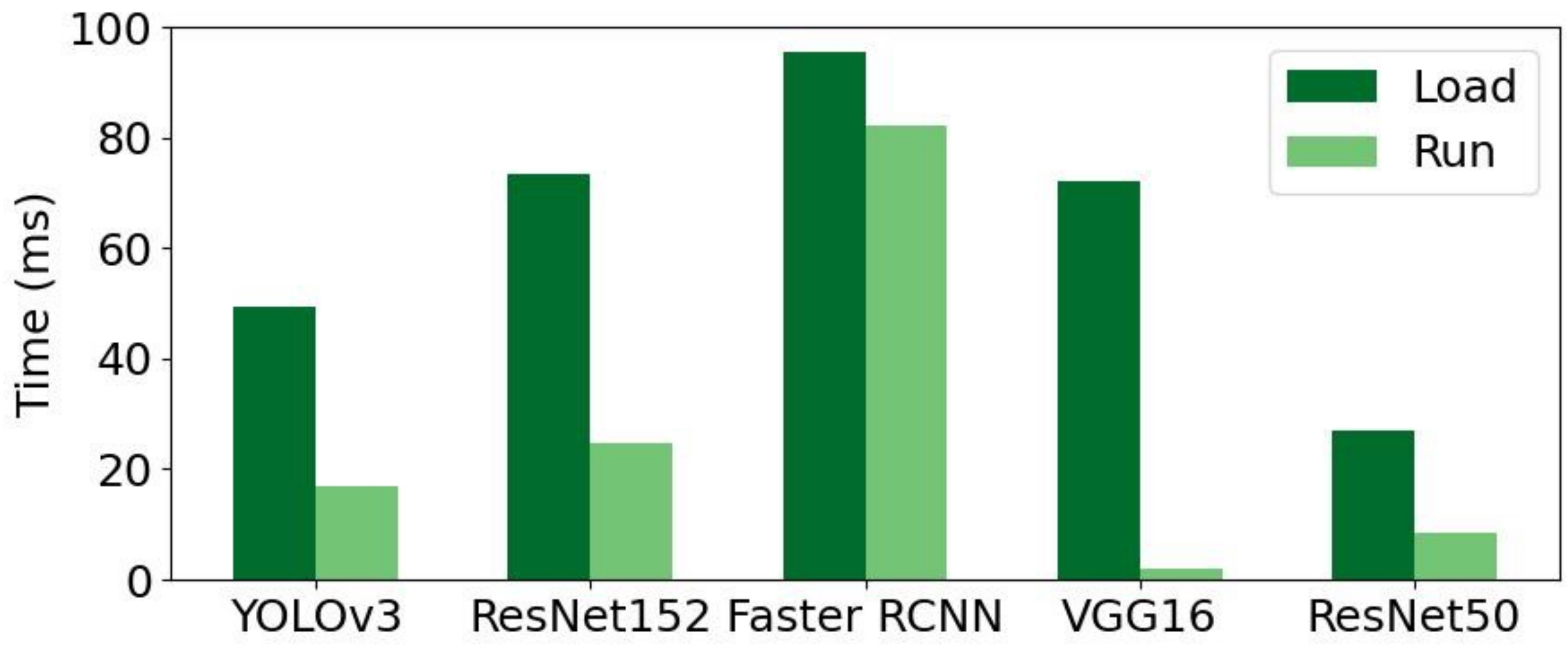
Memory Usage in GPUs

- Weights use GPU memory
- When models are run, GPU memory must also hold intermediates



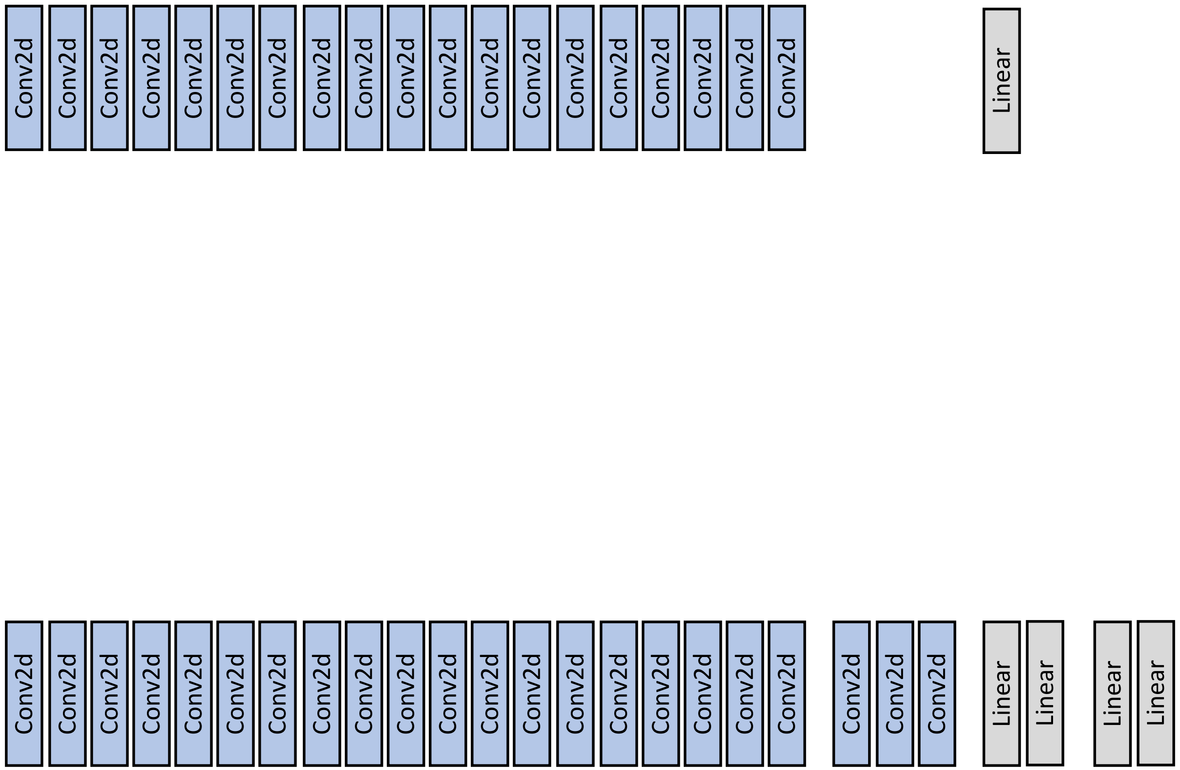
Models Have High Load Time into GPU

- Swapping leads to lower accuracy compared to the case where all models can fit in GPU memory together



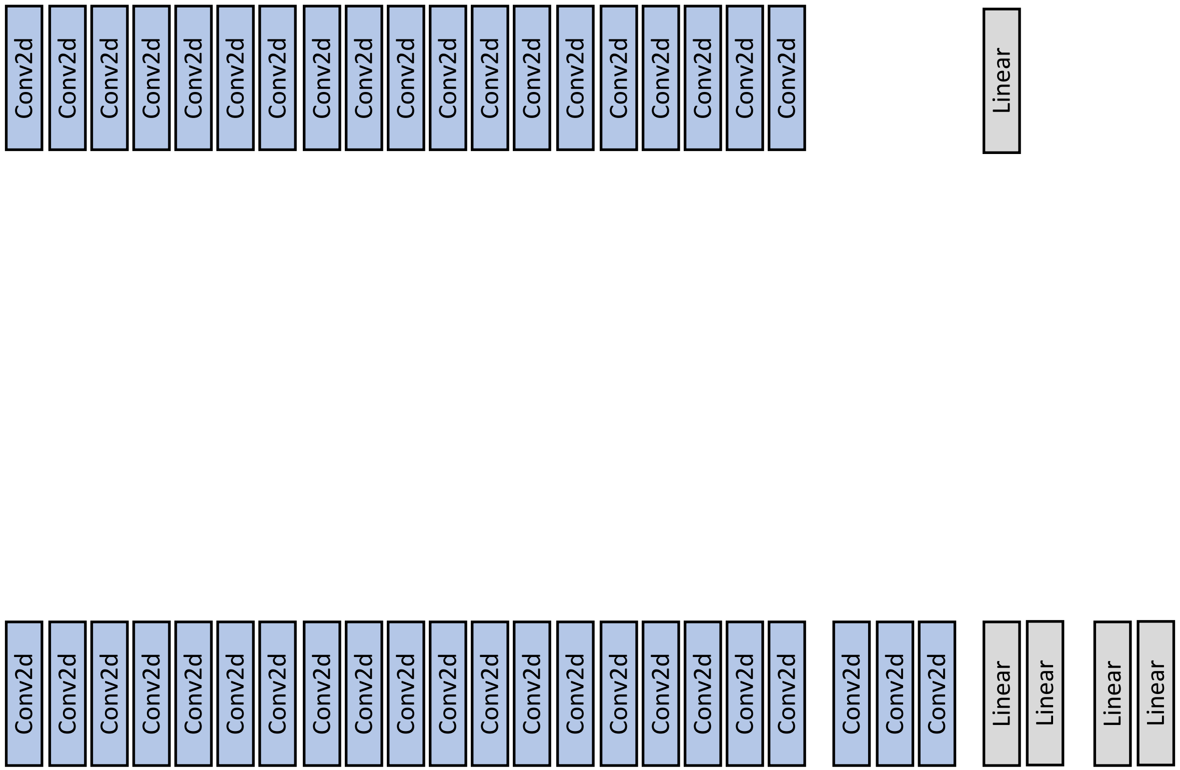
Model Merging

- Observation: some layers are shared between different models



Model Merging

- Observation: some layers are shared between different models



GEMEL

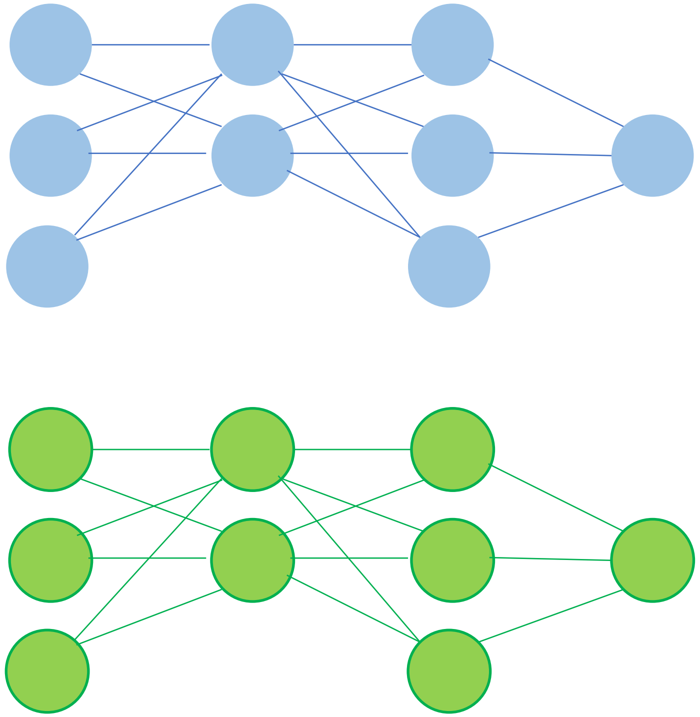
- How much memory could this hypothetically save on realistic workloads?
  - Up to 86% -> could improve accuracy by 17% (once we account for costs of swapping)

Merging Layers

- What might be an issue if we simply took all layers with the same structure and made them use a single set of weights?
- Accuracy would take a hit! All layers in a model are trained together to perform a task.
- We can, however, retrain all the models together with the constraint that the shared layers must have the same weights

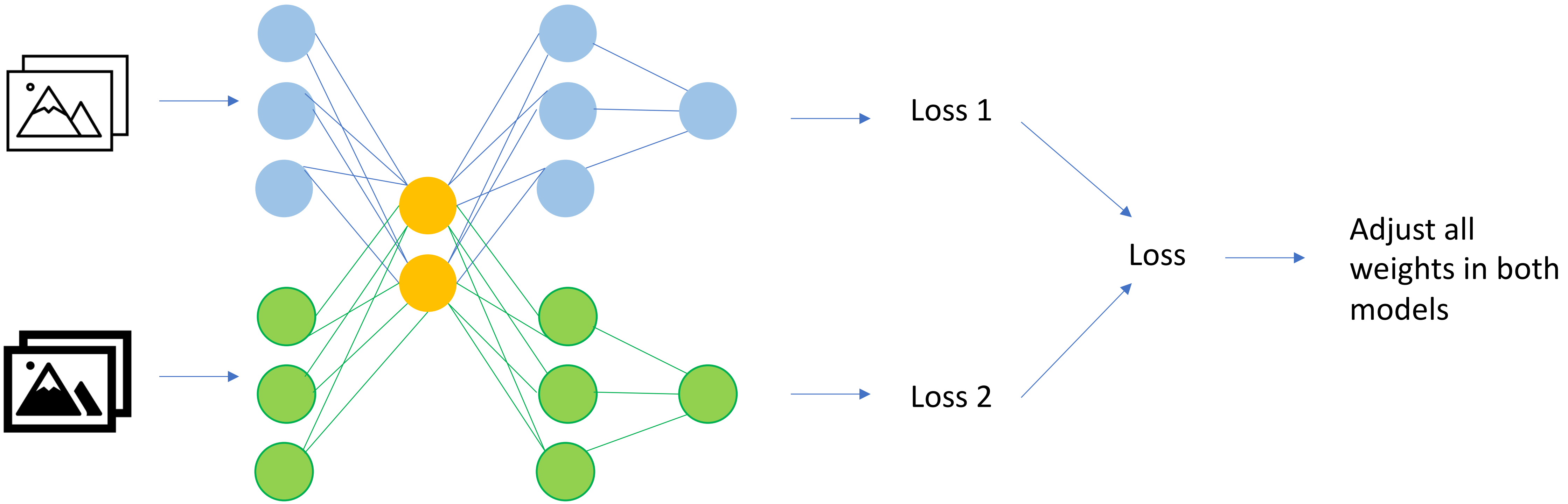
Joint Training

- The shared layer is fixed by creating a single object for the layer. Both models reference this layer



Joint Training

- A single loss function that combines the loss functions of each is used

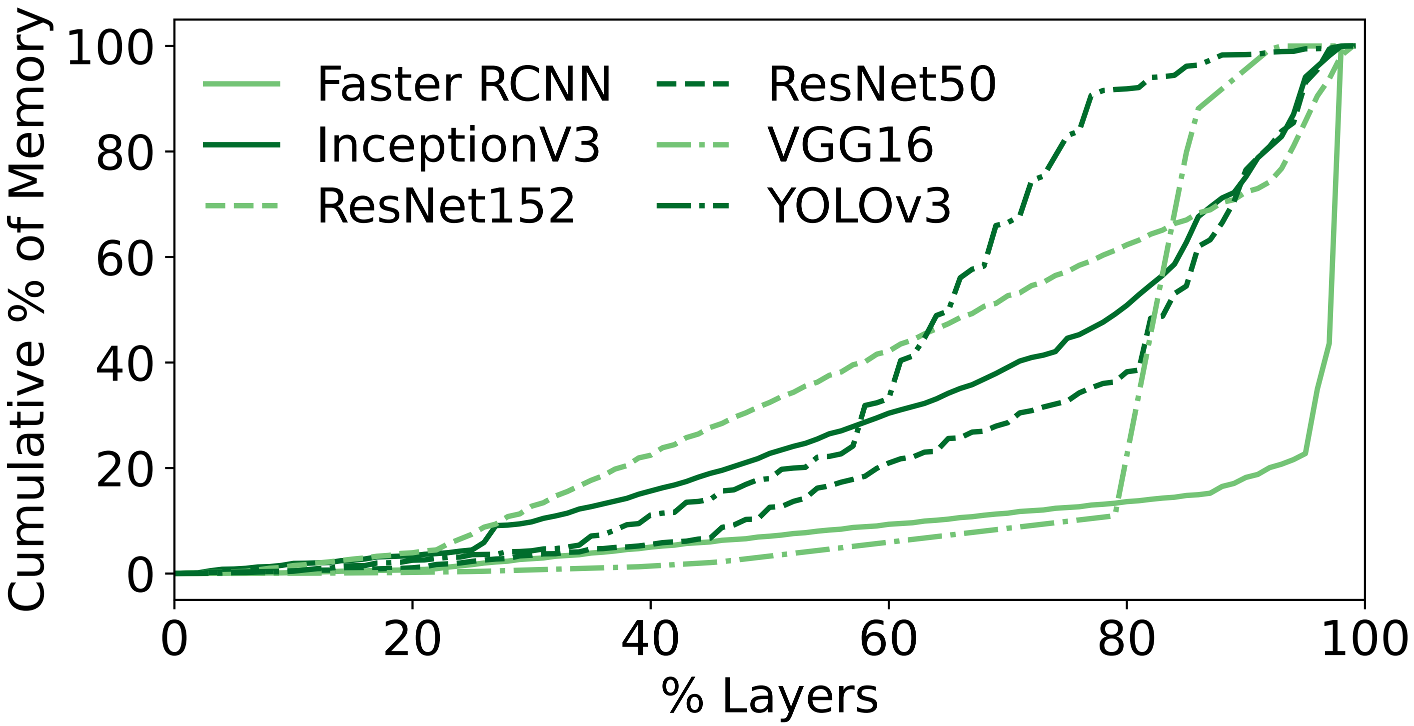


Mainstream?

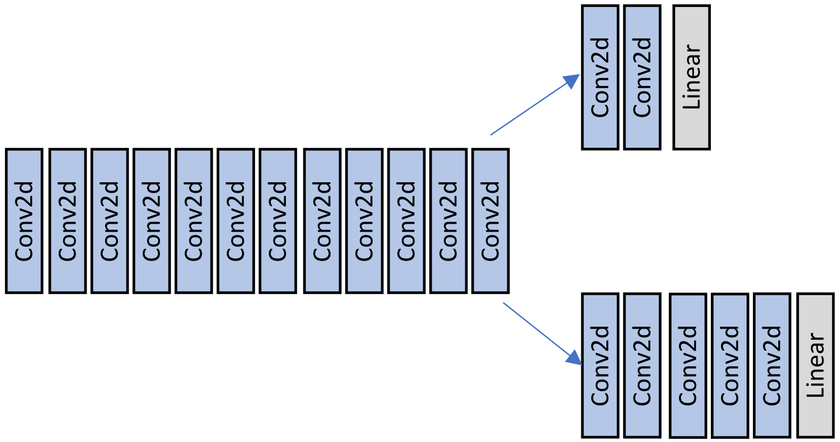
- Mainstream also shared layers in the same way!
- However, Mainstream shared only the earliest layers of each model
- Why does that matter?

Observation: Power-Law Distribution

- Memory usage within model follows power law distribution



Shared stem: doesn't save most memory-heavy layers



Solution

- Merging can be done but we need to be careful about whether accuracy will meet the requirements
- Merging heuristic: find the layers that would save the most memory if shared and try training (greedy algorithm)
  - Added optimizations to make this run faster

Memory Takeaways

- GPU memory can be a bottleneck when running several models on a GPU
- It can also be a bottleneck when training (can Chat GPT fit on a single GPU?)
  - In a couple weeks, we'll look at how training works if the model is too big (occupies too much memory) for one GPU
- The person deploying models needs to be aware of memory when allocating models to GPUs and choosing batch sizes