EE613: Lab on finetuning with PyTorch

Olivier Canévet

December 22, 2023

Convolutional Neural Networks

Finetuning a network in PyTorch

Finetuning a network consists of starting from a pre-trained network and continuing the training on a slightly different task. It is useful when training on small data sets.

Some layers may or may not be frozen.

```
model = models.alexnet(weights="IMAGENET1K_V1")

for param in model.features.parameters():
    param.requires_grad_(False)

model.classifier[1] = nn.Linear(9216, 128)
model.classifier[4] = nn.Linear(128, 128)
model.classifier[6] = nn.Linear(128, 10)
```

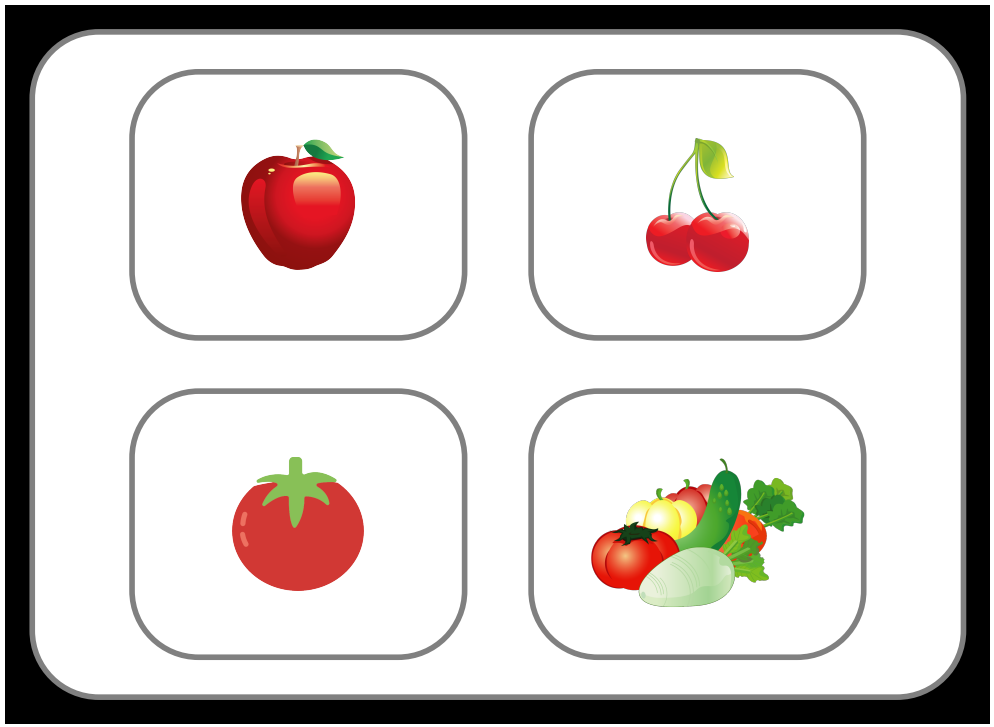
When training from scratch, we do not freeze the layers, because we want them to learn.

Another example with ResNet:

```
model = torchvision.models.resnet34(weights="DEFAULT")
model.fc = nn.Linear(512, num_classes)
```

Smart weigh scale

The user interface would look like this:



- The customer puts the vegetable on the machine,
- The 3 highest confident results are displayed,
- A button to access the full list of items to select the right one if not in top-3.

Lab 6: Train and evaluate a smart weigh scale

Collect realistic data

Add more image distortions.

Same training procedure as in the lab: use pre-trained model, multiple epochs, monitor learning rate, try several architectures, etc.

Evaluate the classifier with the top-k error (in PyTorch, `torch.topk`).

Perform constant data collection and annotation: use the customer choices as labeled data to enrich the training set, and training store-specific classifiers (or even machine-specific within the same store).