CS167: Machine Learning

Fine-tuning popular CNNs for image recognition

Tuesday, April 23rd, 2024

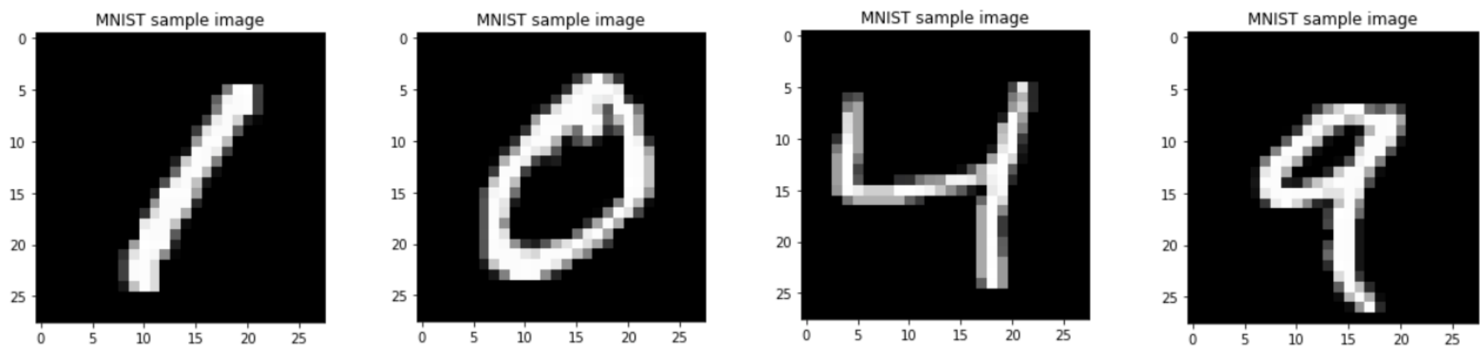
Drake UNIVERSITY

Recap

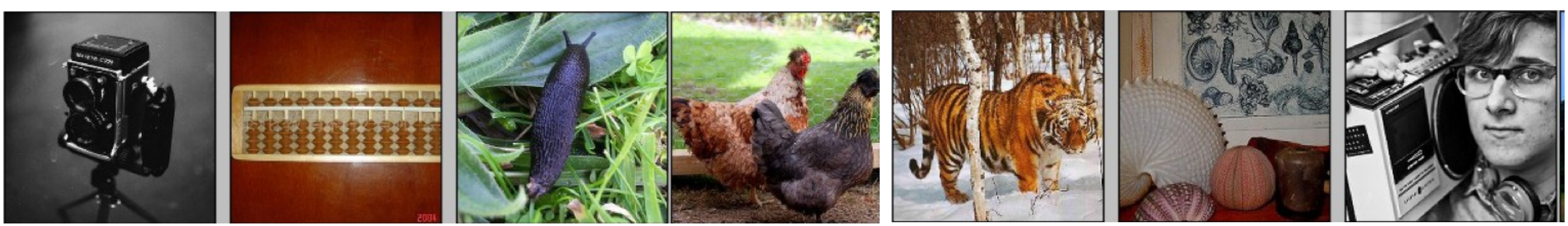
- Popular CNNs
  - LeNet
  - AlexNet
  - VGG
  - ResNet
- Training vs. Fine-tuning
- Fine-tuning a popular CNN (eg, AlexNet) using an arbitrary dataset

Recap: LeNet

- LeNet is a simple CNN architecture suitable for well-structured image
    - e.g., 28x28 pixels image of digits from 0 to 9 in MNIST or our Fashion-MNIST dataset



- Real-world images are much more complicated; pose challenges in classification
    - e.g., high resolution images 600x480 pixels image and contents have a lot more diversity



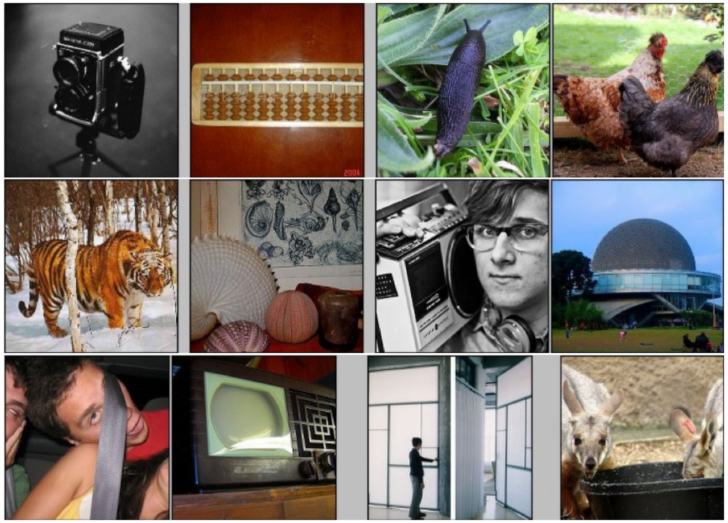
Today's Agenda

- Popular CNNs
  - LeNet
  - AlexNet
  - VGG
  - ResNet
- Training vs. Fine-tuning
- Fine-tuning a popular CNN (eg, AlexNet) using an arbitrary dataset

ImageNet Challenge 2012

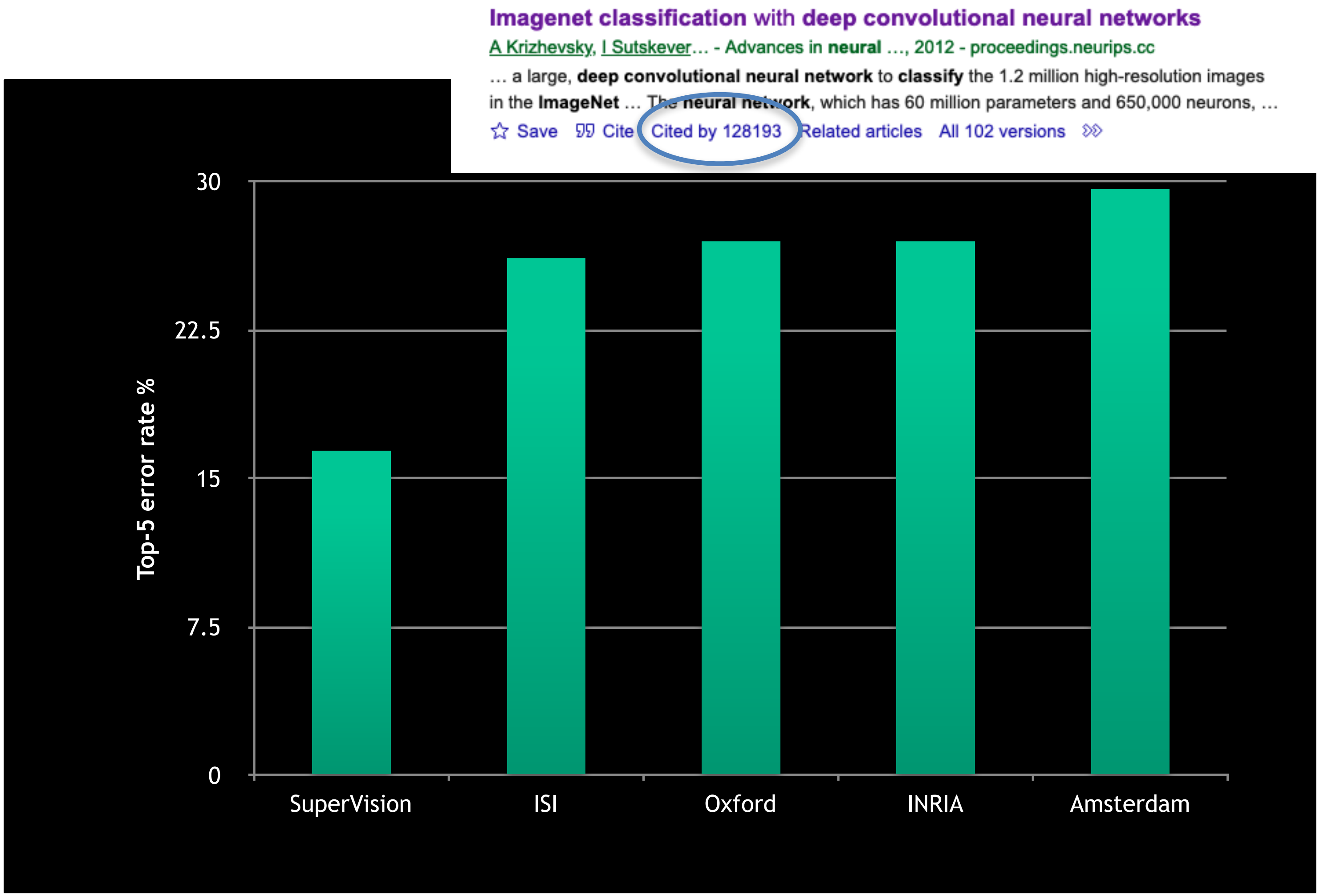
- ~14 million labeled images, 20k classes
- Images gathered from Internet
- Human labels via Amazon Turk
- **Challenge: 1.2 million training images, 1000 classes**

IMAGENET



[Deng et al. CVPR 2009]

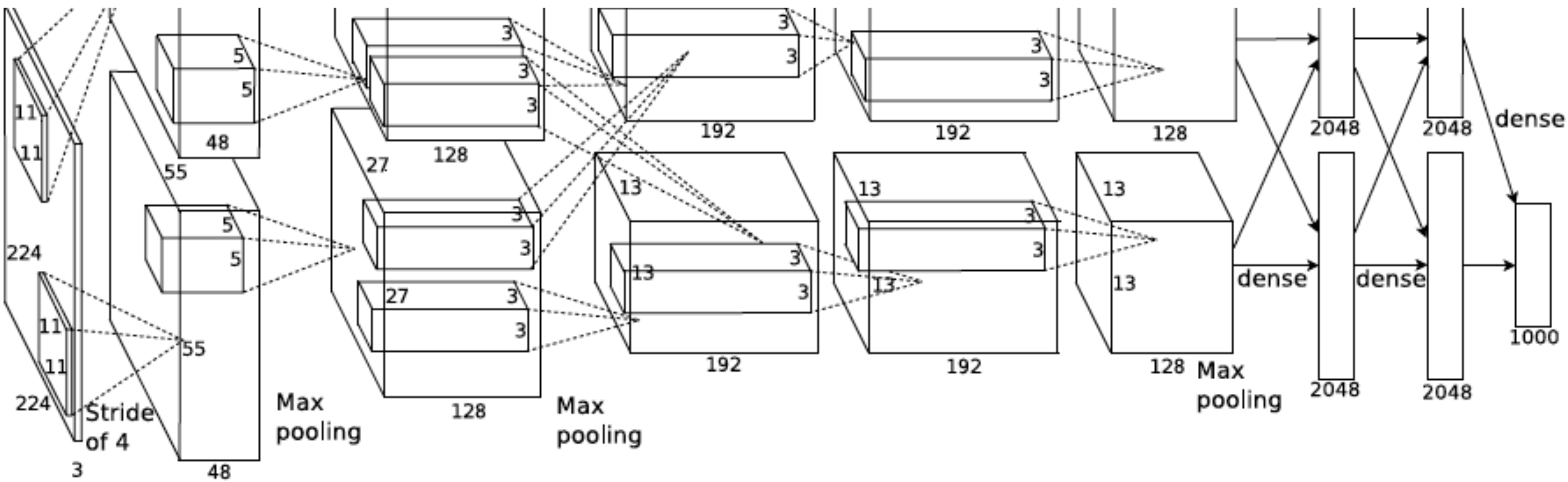
Recap: ImageNet Challenge 2012



- AlexNet (Krizhevsky et al.) -- **16.4% error** (top-5)
- Next best (non-convnet) – **26.2% error**

Popular CNN: AlexNet

- Similar framework to LeCun'98 but:
  - Bigger model (7 hidden layers, 650,000 units, 60,000,000 params)
  - More data (10^6 vs. 10^3 images)
  - GPU implementation (50x speedup over CPU)
    - Trained on two GPUs for a week
  - Better regularization for training (DropOut)



A. Krizhevsky, I. Sutskever, and G. Hinton, ImageNet Classification with Deep Convolutional Neural Networks, NIPS 2012

Popular CNN: AlexNet

Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2

} self.features

[6x6x256] **ADAPTIVE AVG POOL**: filters with output size 6x6

} self.avgpool

[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)

} self.classifier

A. Krizhevsky, I. Sutskever, and G. Hinton, ImageNet Classification with Deep Convolutional Neural Networks, NIPS 2012

Today's Agenda

- Popular CNNs
  - LeNet
  - AlexNet
  - VGG
  - ResNet
- Training vs. Fine-tuning
- Fine-tuning a popular CNN (eg, AlexNet) using an arbitrary dataset

Popular CNN: VGG

- VGG was the winner of ImageNet (1000-class image classification) challenge in 2014
  - proposed by Andrew Zisserman's group in Oxford University

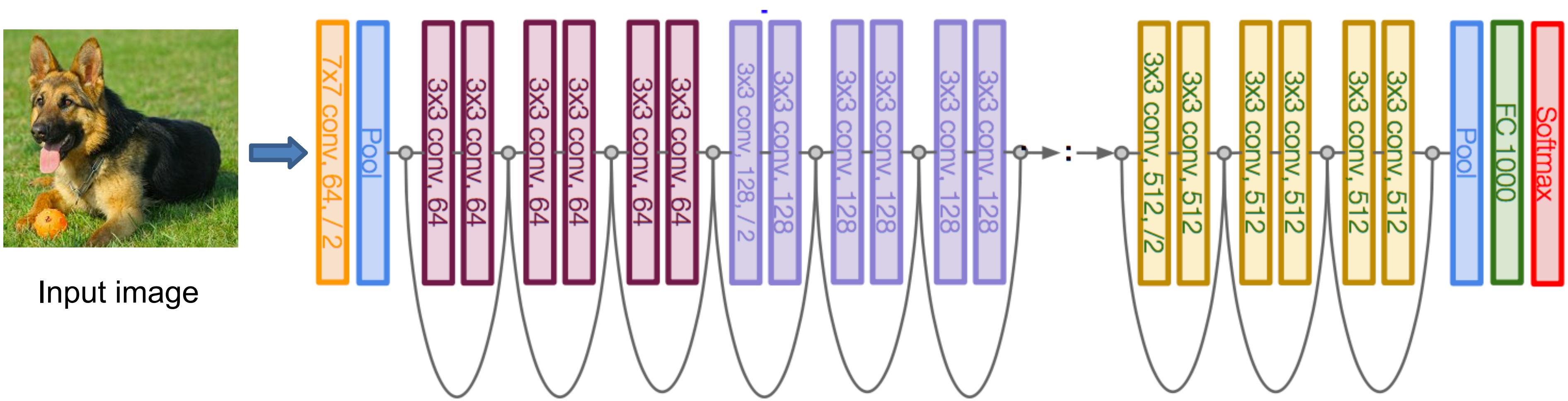
Very Deep Convolutional Networks for Large-Scale Image Recognition - **Karen Simonyan and Andrew Zisserman**

Today's Agenda

- Popular CNNs
  - LeNet
  - AlexNet
  - VGG
  - ResNet
- Training vs. Fine-tuning
- Fine-tuning a popular CNN (eg, AlexNet) using an arbitrary dataset

Popular CNN: ResNet

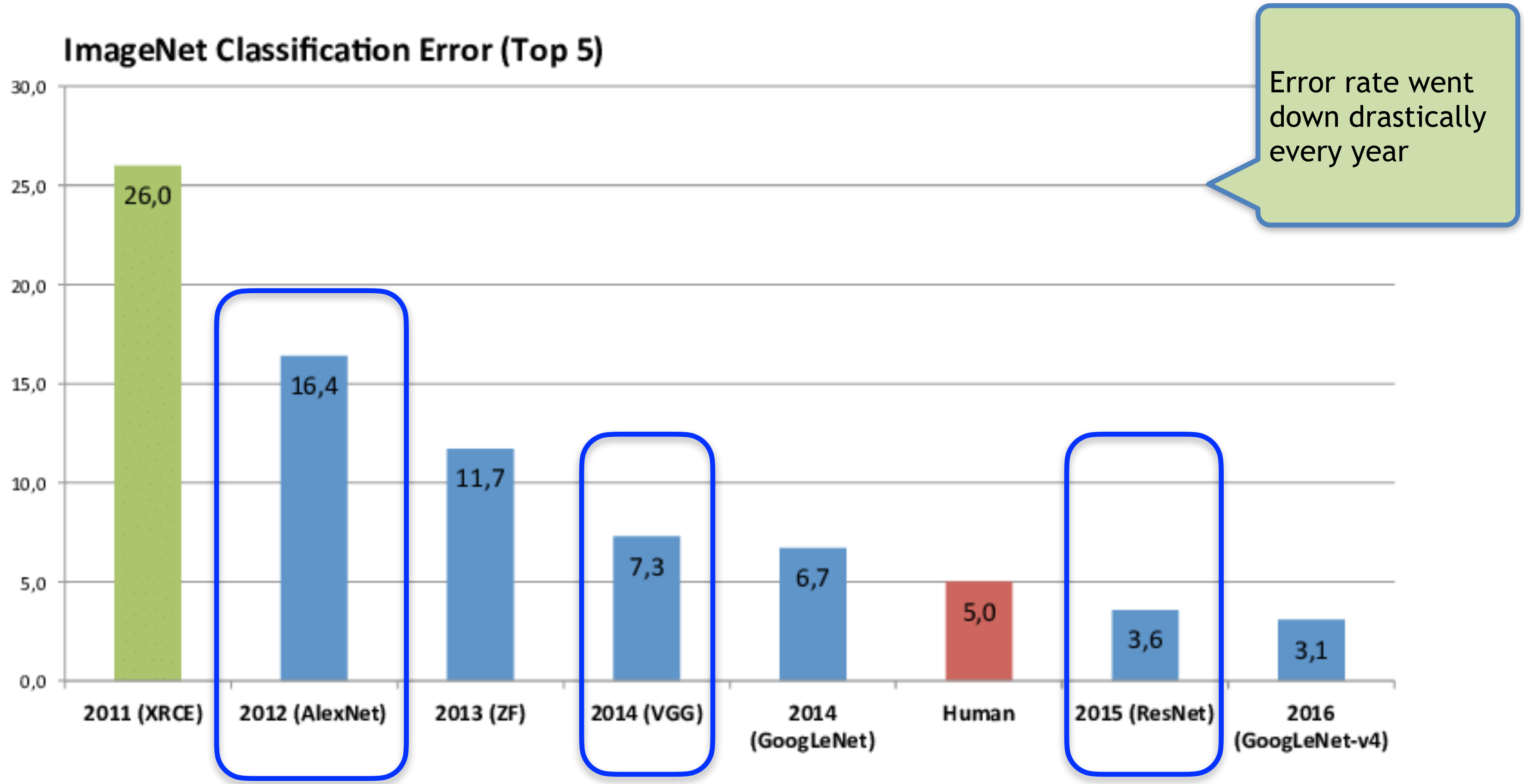
- ResNet was the winner of ImageNet challenge in 2015



[Deep Residual Learning for Image Recognition]() - **Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun**

ImageNet Winners by the Popular CNNs

- AlexNet (2012) —> VGG (2014)—> ResNet (2015)



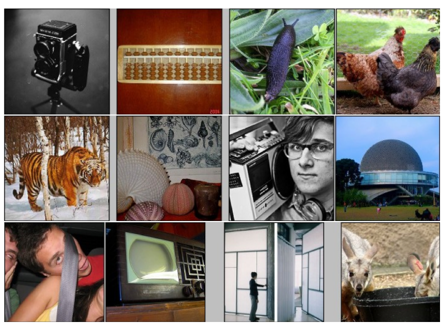
Today's Agenda

- Popular CNNs
  - LeNet
  - AlexNet
  - VGG
  - ResNet
- Training vs. Fine-tuning
- Fine-tuning a popular CNN (eg, AlexNet) using an arbitrary dataset

Training a Model

- **Training** refers to the process of training a model from scratch, often on a large and general dataset (e.g., ImageNet for image classification).

IMAGENET



A dataset with over 1 million images

Feed to a model for training

Useless model

A deep neural network with randomly initialized weights

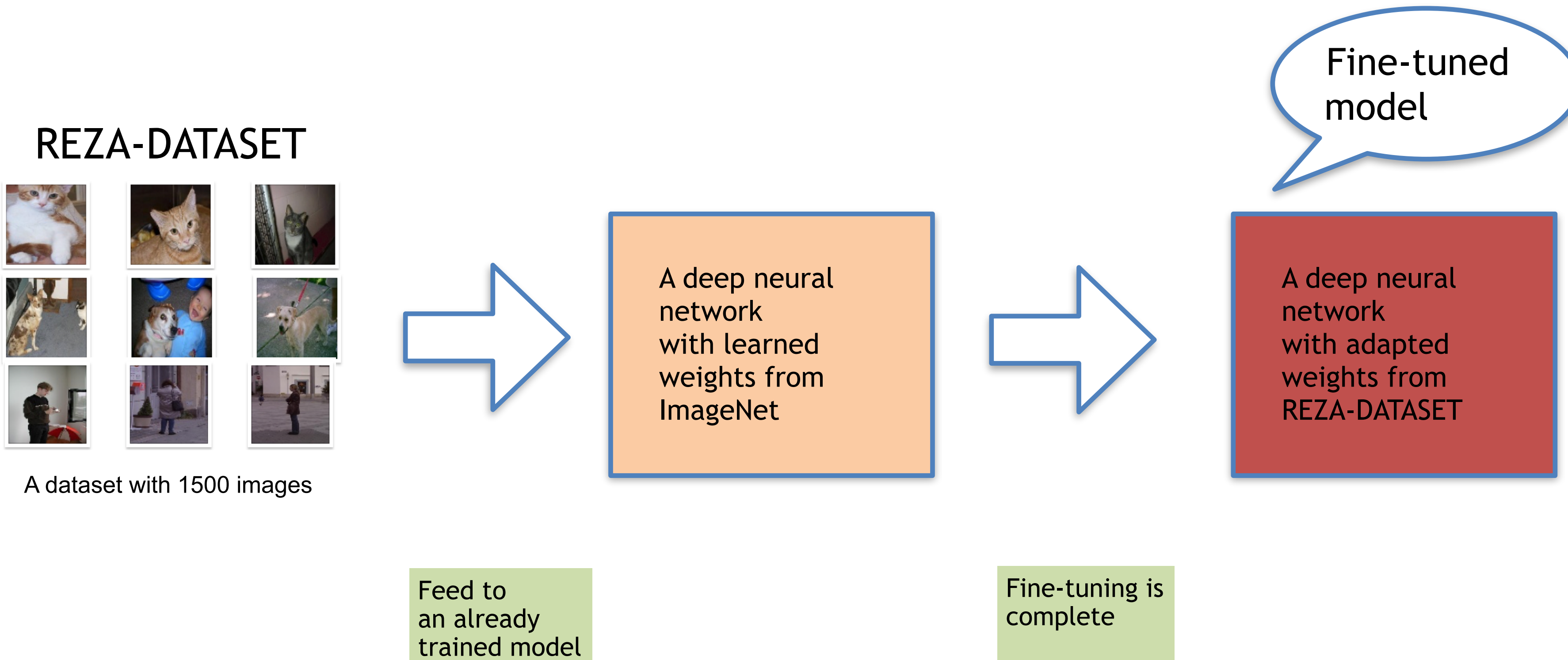
Training is Complete

Trained model

A deep neural network with learned weights

Fine-tuning a Model

- **Fine-tuning** refers to the process of taking a pre-trained model and further training it on a new or specific dataset. The initial model is often trained on a large and general dataset, e.g., ImageNet, and fine-tuning adapts the model to perform well on a more specific task or dataset.

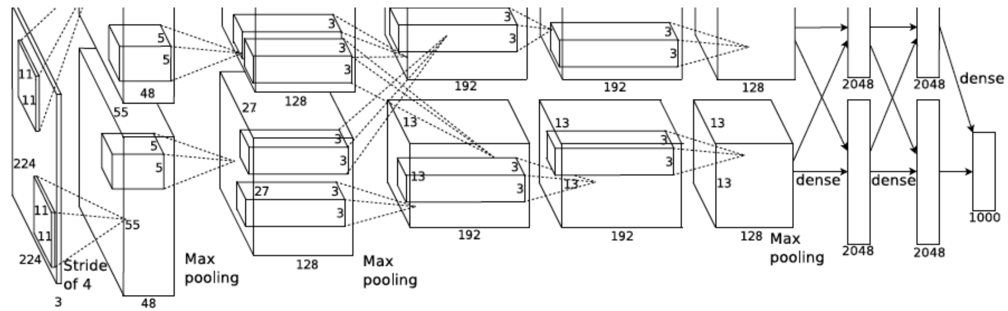


Today's Agenda

- Popular CNNs
  - LeNet
  - AlexNet
  - VGG
  - ResNet
- Training vs. Fine-tuning
- Fine-tuning a popular CNN (eg, AlexNet) using an arbitrary dataset

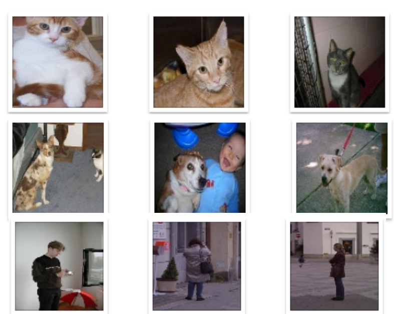
Fine-tuning AlexNet on an Arbitrary Dataset

- Let’s use one of the popular CNNs
  - LeNet
  - AlexNet
  - VGG
  - ResNet



- Let’s fine-tune AlexNet with a new dataset eg, REZA-DATASET

REZA-DATASET



A dataset with 1500 images

Existing Dataset in PyTorch

- Notice these are some of the datasets provided by PyTorch.

Image classification

Caltech101(root[, target_type, transform, …])	Caltech 101 Dataset.
Caltech256(root[, transform, …])	Caltech 256 Dataset.
CelebA(root[, split, target_type, …])	Large-scale CelebFaces Attributes (CelebA) Dataset Dataset.
CIFAR10(root[, train, transform, …])	CIFAR10 Dataset.

Fine-tuning AlexNet on an Arbitrary Dataset

- Download the following dataset and put it into your Google Drive

- Bike-Cat-Dog-Person Dataset
  - Each image size: **100x100x3**
    - Note that these are color images
  - Each image is associated with a label from **4 classes**
  - Training set of **1500** examples and test set of **300** examples

https://analytics.drake.edu/~reza/teaching/cs167_sp24/dataset/bcdp_v1.zip

Fine-tuning AlexNet on an Arbitrary Dataset

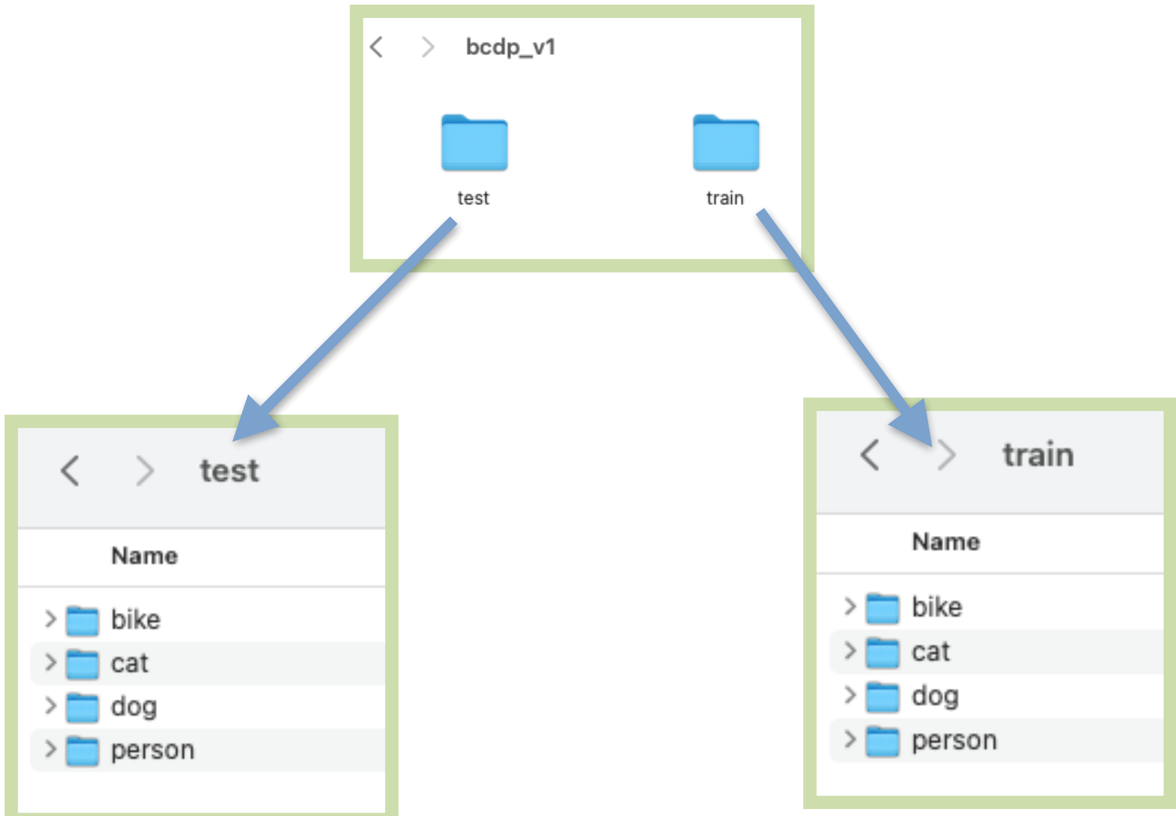
- This is a random dataset of images, unlike the datasets provided by PyTorch.

Image classification

Caltech101(root[, target_type, transform, ...])	Caltech 101 Dataset.
Caltech256(root[, transform, ...])	Caltech 256 Dataset.
CelebA(root[, split, target_type, ...])	Large-scale CelebFaces Attributes (CelebA) Dataset Dataset.
CIFAR10(root[, train, transform, ...])	CIFAR10 Dataset.

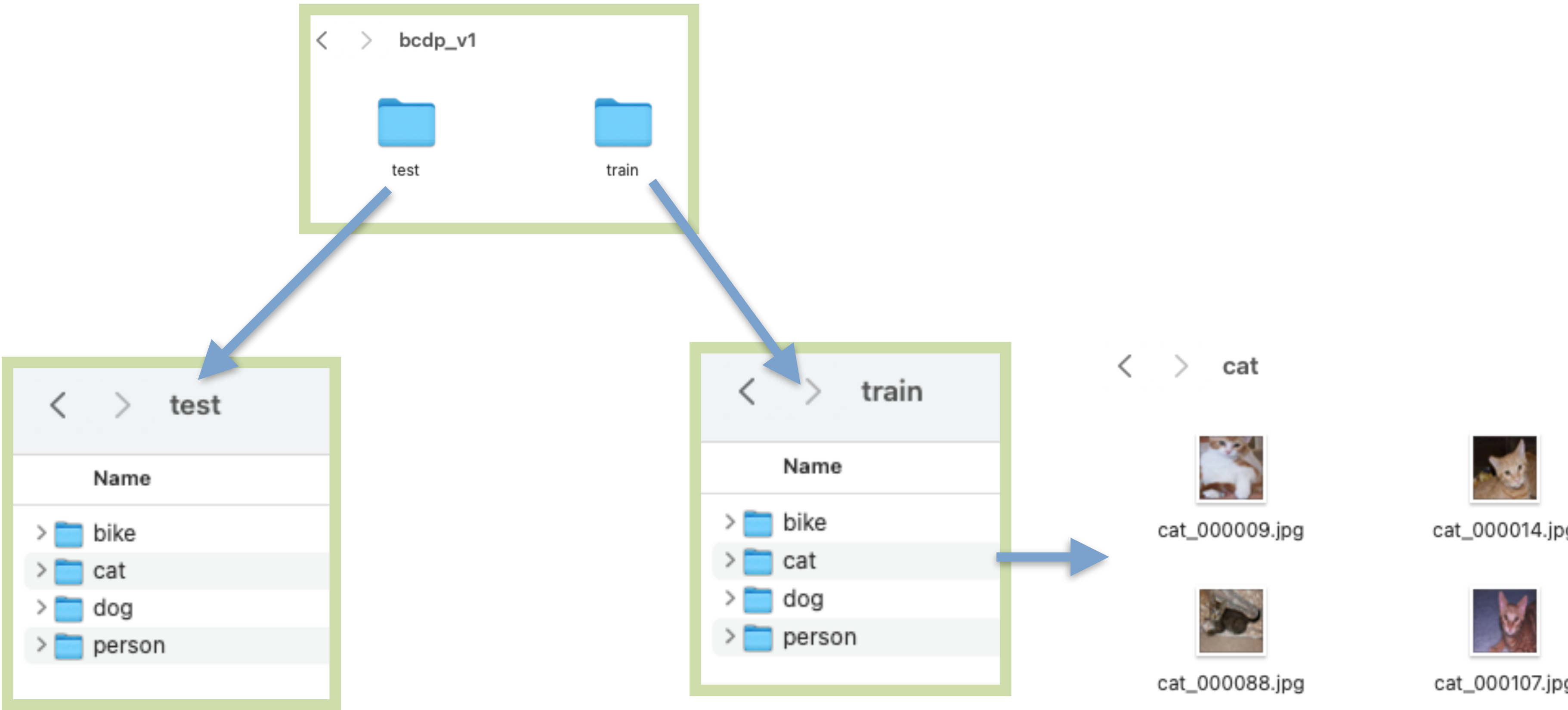
Fine-tuning AlexNet on an Arbitrary Dataset

- This dataset is organized into 'train' and 'test' folders as follows:



Fine-tuning AlexNet on an Arbitrary Dataset

- Each folder (‘train' and ‘test) contains a set of images that will be used by our model during fine-tuning and testing, respectively



Existing Dataset in PyTorch

- If we need to use PyTorch's existing datasets, we can use the following module from PyTorch to easily download and prepare the data loader for training and testing.

https://pytorch.org/vision/stable/datasets.html#

Docs > Datasets

```
imagenet_data = torchvision.datasets.ImageNet('path/to/imagenet_root/')
data_loader = torch.utils.data.DataLoader(imagenet_data,
                                          batch_size=4,
                                          shuffle=True,
                                          num_workers=args.nThreads)
```

- This is what we used in our previous experiment when training our own CNN from scratch using the CIFAR-10 dataset or Fashion-MNIST dataset.

Using Arbitrary Dataset

- Instead, when we need to use an arbitrary dataset, we can use the following module from PyTorch to prepare the data loader for training and testing.

```
from torch.utils.data import DataLoader
from torchvision import datasets
from torchvision import transforms


# For fine-tuning with an AlexNet/VGG/ResNet architecture that has been
# pre-trained using the ImageNet dataset, you need to normalize
# each image with the given mean and standard deviation.
transform = transforms.Compose([
    transforms.Resize((227, 227)),
    transforms.ToTensor(),
    transforms.Normalize((.229, .224, .225), (.485, .456, .406)) # ImageNet: mea
])


train_dir       = '/content/drive/MyDrive/cs167_fall23/datasets/bcdp_v1/train'
test_dir        = '/content/drive/MyDrive/cs167_fall23/datasets/bcdp_v1/test'

train_dataset   = datasets.ImageFolder(train_dir, transform=transform)
test_dataset    = datasets.ImageFolder(test_dir,  transform=transform)

train_dataloader  = DataLoader(train_dataset, batch_size=batch_size_val, shuffle=True)
test_dataloader   = DataLoader(test_dataset, batch_size=batch_size_val,  shuffle=False)
```

Loading a Pre-trained AlexNet Model in PyTorch

- Import a pre-trained instance of AlexNet inside our Network class and make any other necessary changes as follows:

```
class AlexNet(nn.Module):
    def __init__(self, num_classes, pretrained=True):
        super(AlexNet, self).__init__()
        net = models.alexnet(pretrained=True)

        # retained earlier convolutional and pooling layers from AlexNet
        self.features   = net.features
        self.avgpool    = net.avgpool

        # added new fully connected layers
        self.classifier = nn.Sequential(
            nn.Linear(256 * 6 * 6, 4096),
            nn.ReLU(True),
            nn.Dropout(),
            nn.Linear(4096, 512),
            nn.ReLU(True),
            nn.Dropout(),
            nn.Linear(512, num_classes)
        )

    def forward(self, x):
        #print("shape of input: ", x.shape)
        x = self.features(x)
        #print("output shape (self.features): ", x.shape)
        x = self.avgpool(x)
        #print("output shape (self.avgpool): ", x.shape)
        x = torch.flatten(x, 1)
        x = self.classifier(x)
        #print("output shape (self.classifier): ", x.shape)
        return x
```

Fine-tuning AlexNet on an Arbitrary Dataset

- Go to Blackboard and follow the notebook as shown below: