CS167: Machine Learning

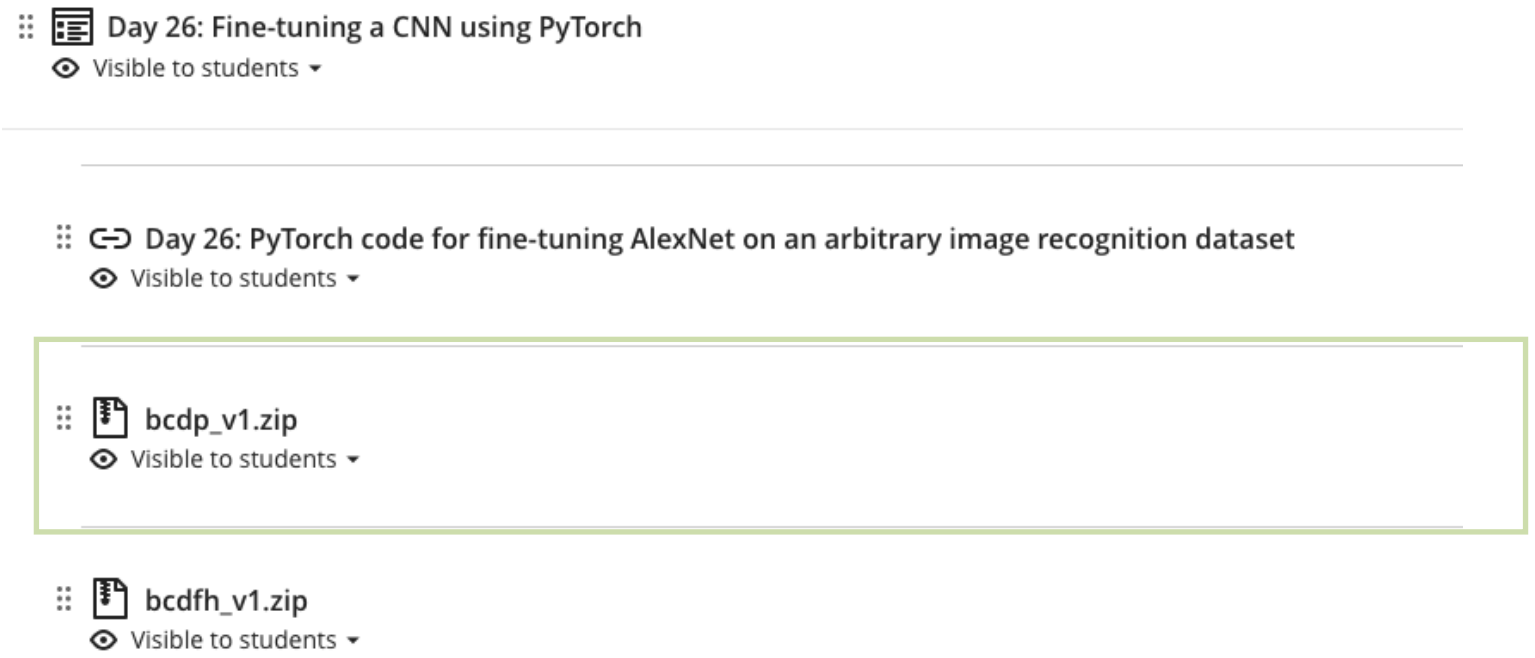
CNN review
Fine-tuning popular CNNs for image recognition

Wednesday, April 29th, 2026

Drake
UNIVERSITY

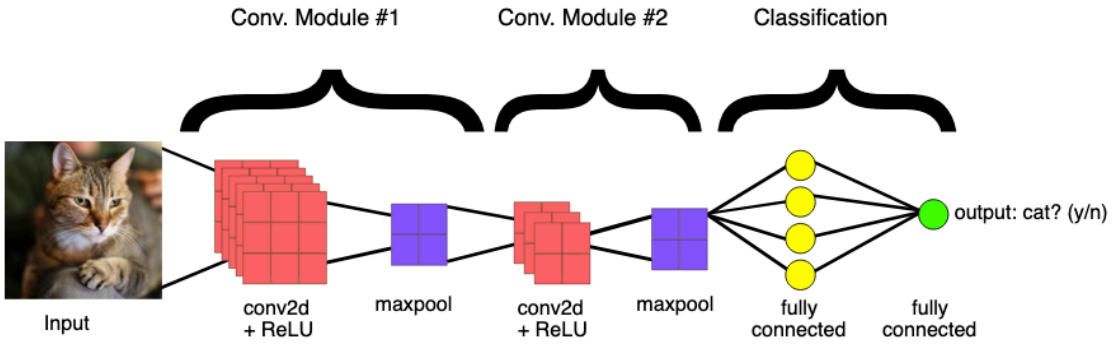
Download the following dataset and upload it to your Google Drive

- Since it will take some time, I'd like you to start the upload process in the background while we discuss the new content.

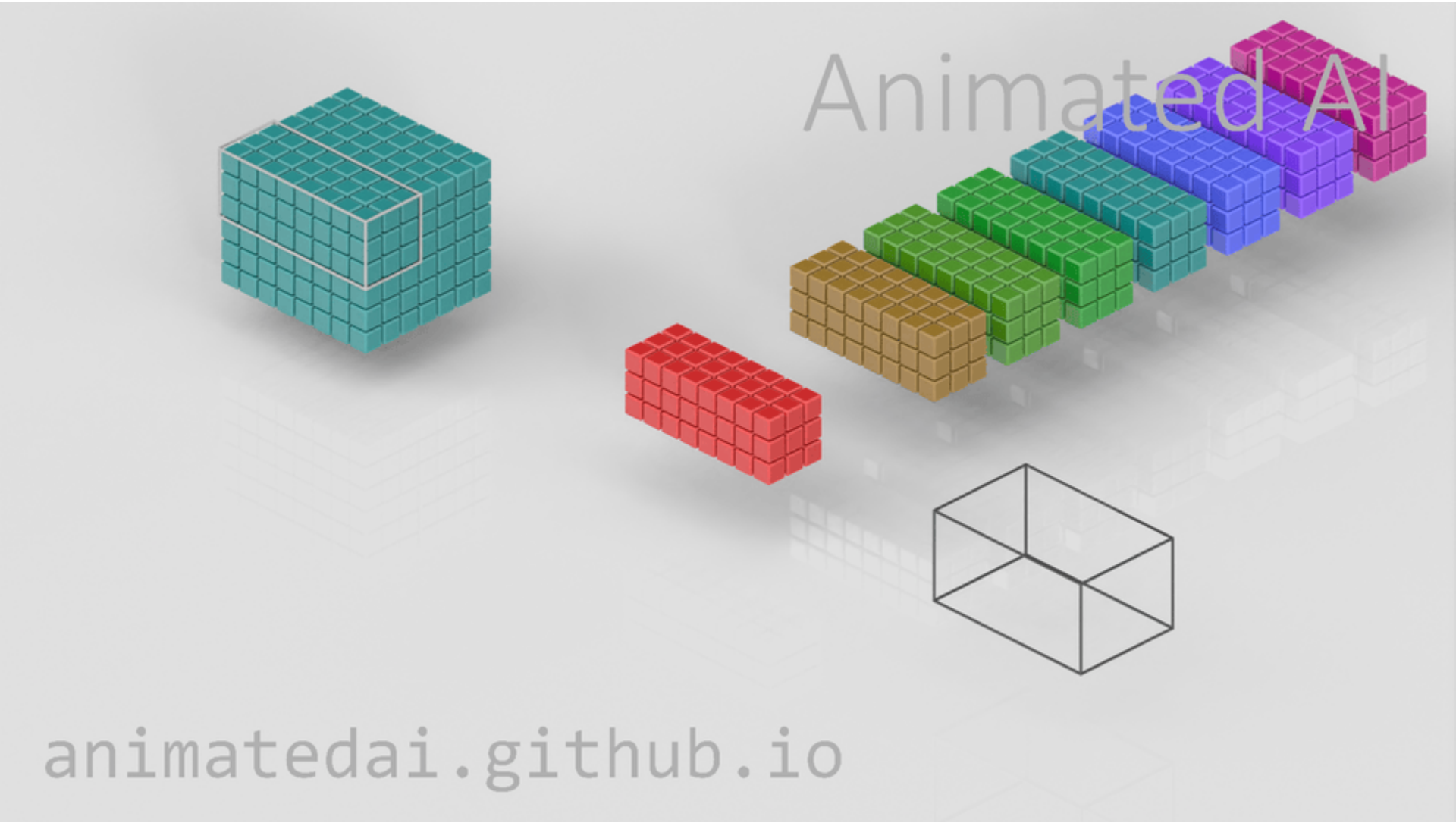


Recap: Convolutional Neural Network (CNN)

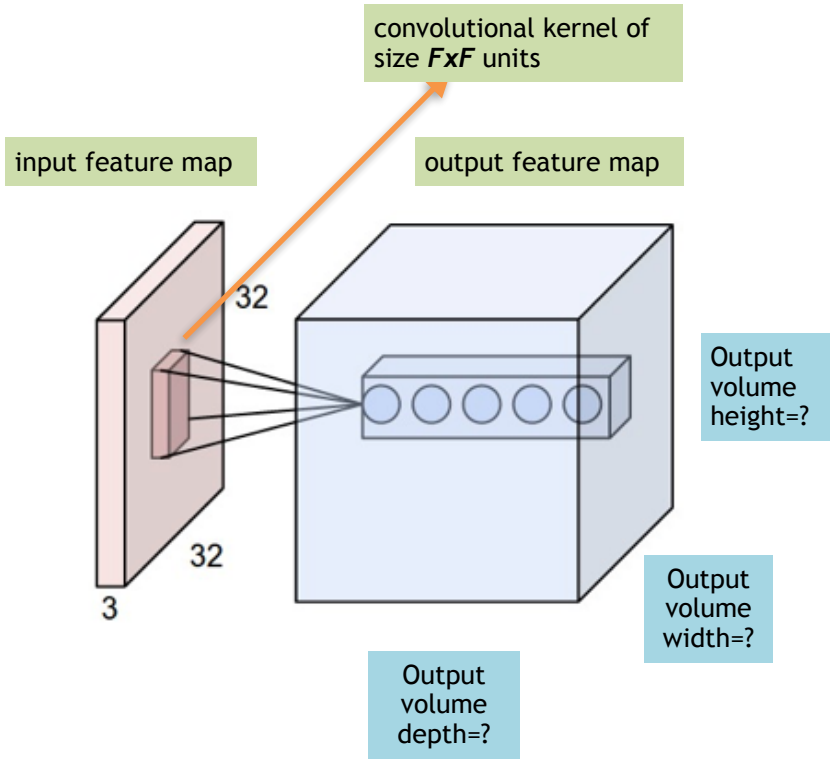
- Convolutional Neural Network (CNN): another type of neural network
  - Convolution operation
  - Nonlinearity
  - Pooling operation
  - CNN: convolutional layer + nonlinearity + pooling layer



Recap: How to calculate the output volume size?

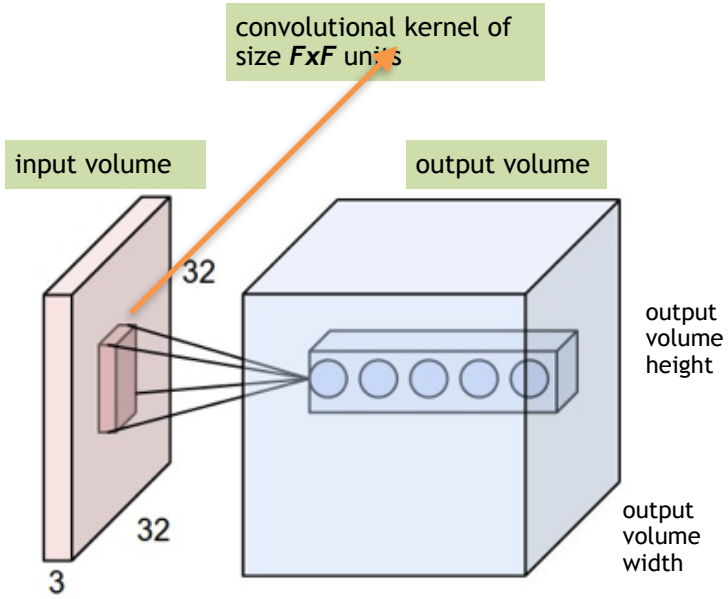


Recap: How to calculate the output volume size?



Recap: How to calculate the output volume size?

- An input volume has size **(W_1 x H_1 x D_1)**
  - Filter size/receptive field is ***(FxF)***
  - Spatial stride size ***S***
  - Padding size ***P***
  - Number of filters ***K***



- Spatial sizes of the output volume ***(W_2 x H_2 x D_2)***

$$W_2 = \frac{(W_1 - F + 2P)}{S} + 1$$

$$H_2 = \frac{(H_1 - F + 2P)}{S} + 1$$

$$D_2 = K$$

- Number of filter weight parameters ***= (F x F x D_1) x K***
- Number of bias parameters = ***K***

Consider the first convolutional layer

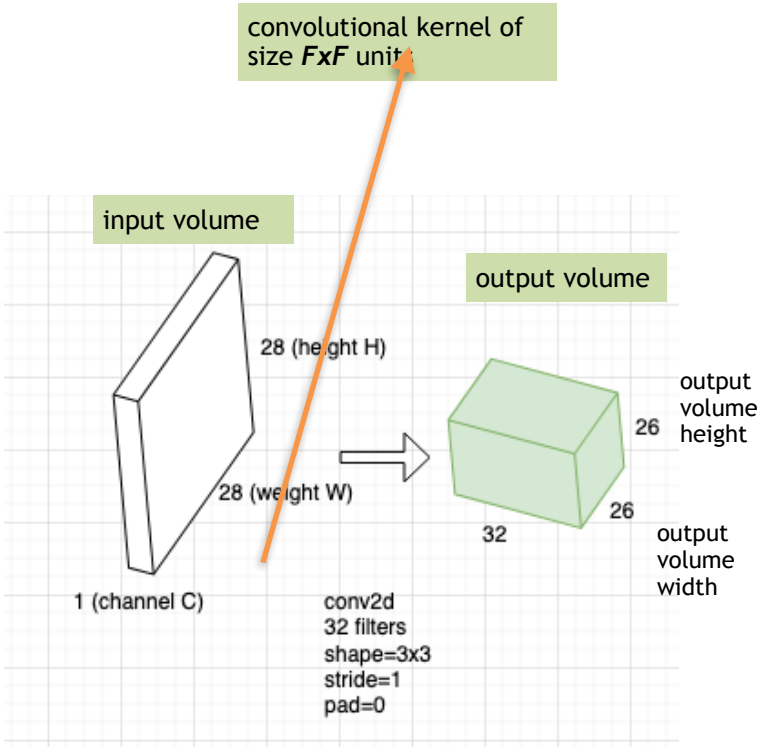
- An input volume has size **(W_1 x H_1 x D_1) = (28 x 28 x 1)**
  - Filter size/receptive field is ***(FxF): 3x3***
  - Spatial stride size ***S: 1***
  - Padding size ***P: 0***
  - Number of filters ***K:***

- Spatial sizes of the output volume
  (W_2 x H_2 x D_2) = ????

$$W_2 = \frac{(W_1 - F + 2P)}{S} + 1 = \frac{(28 - 3 + 2*0)}{1} + 1 = 26$$

$$H_2 = \frac{(H_1 - F + 2P)}{S} + 1 = \frac{(28 - 3 + 2*0)}{1} + 1 = 26$$

$$D_2 = K = 32$$



Your turn? Consider the second convolutional layer

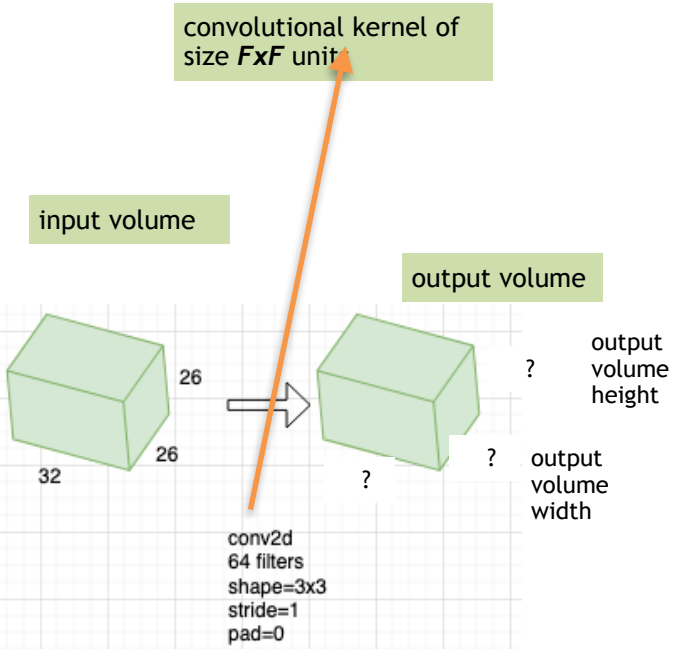
- Now the input volume has size **(W_1 x H_1 x D_1) = (26 x 26 x 32)**
  - Filter size/receptive field is ***(FxF): 3x3***
  - Spatial stride size ***S: 1***
  - Padding size ***P: 0***
  - Number of filters ***K: 64***

- What is the spatial sizes of the output volume ***(W_2 x H_2 x D_2) = ????***

$$W_2 = \frac{(W_1 - F + 2P)}{S} + 1 = \frac{(? - ? + ?)}{?} + 1 = ?$$

$$H_2 = \frac{(H_1 - F + 2P)}{S} + 1 = \frac{(? - ? + ?)}{?} + 1 = ?$$

$$D_2 = K = ?$$



Answer

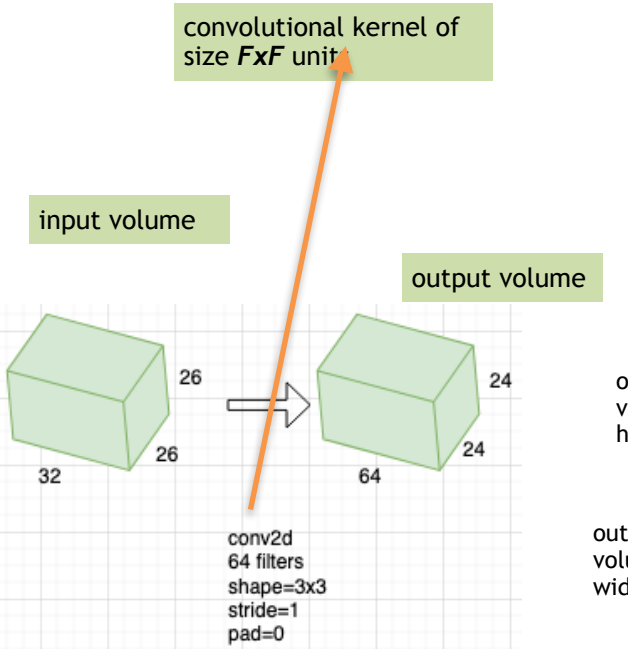
- Now the input volume has size **(W_1 x H_1 x D_1) = (26 x 26 x 32)**
  - Filter size/receptive field is ***(FxF): 3x3***
  - Spatial stride size ***S: 1***
  - Padding size ***P: 0***
  - Number of filters ***K:***

- What is the spatial sizes of the output volume ***(W_2 x H_2 x D_2) = ????***

$$W_2 = \frac{(W_1 - F + 2P)}{S} + 1 = \frac{(26 - 3 + 2*0)}{1} + 1 = 24$$

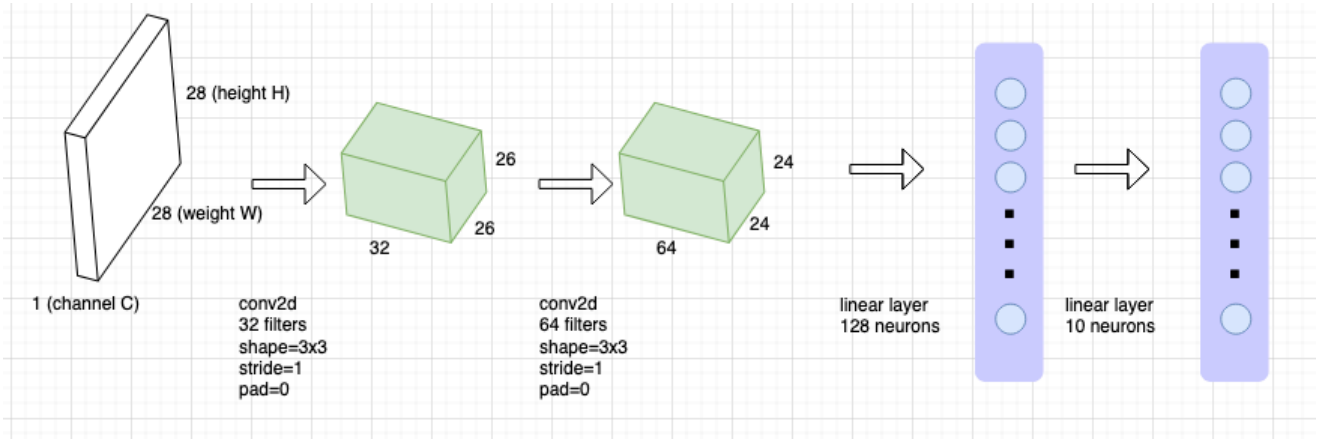
$$H_2 = \frac{(H_1 - F + 2P)}{S} + 1 = \frac{(26 - 3 + 2*0)}{1} + 1 = 24$$

$$D_2 = K = 64$$



Full Network

```
class SimpleCNNv1(nn.Module):
  def __init__(self):
    super().__init__()
    # your network layer construction should take place here

    input_volume_channel_first            = 1
    output_volume_channel_first           = 32
    input_volume_channel_second           = 32
    output_volume_channel_second          = 64
    input_volume_channel_third            = 64
    number_of_neurons_before_output_layer = 128
    num_of_neurons_output_layer           = 10 # usually this number should be equal to the total number of classes in your classification task

    # Beginning layers: a series of 2D convolutional layers (useful for feature map learning from the grid layouts of an image)
    self.first_conv_2d              = nn.Conv2d(input_volume_channel_first, output_volume_channel_first, 3)  # 2D convolutional transformation module
    self.relu_activation_1st        = nn.ReLU()
    self.second_conv_2d             = nn.Conv2d(input_volume_channel_second, output_volume_channel_second, 3) # 2D convolutional transformation module
    self.relu_activation_2nd        = nn.ReLU()

    # ------------------------------------------------------------------------------------------------------------------------------------
    # -------                                                heads up!                                                          --------
    # you need to calculate the total_size_of_the_output_volume of your self.second_conv_2d layer,
    # as it will be needed by the upcoming nn.Linear(). This number will be used as the first argument for the next nn.Linear().
    # I pre-calculated this number, and it is 64*24*24 = 36864. I will plug this number in the next layer
    # ------------------------------------------------------------------------------------------------------------------------------------

    # End layers: a series of dense linear layers (almost like an MLP for final classification)
    self.first_linear_layer         = nn.Linear(64*24*24, number_of_neurons_before_output_layer)  # linear transformation module (input=64*24*24, outpu
    self.relu_activation_3rd        = nn.ReLU()
    self.output_layer               = nn.Linear(number_of_neurons_before_output_layer, num_of_neurons_output_layer) # linear transformation module (inp
```

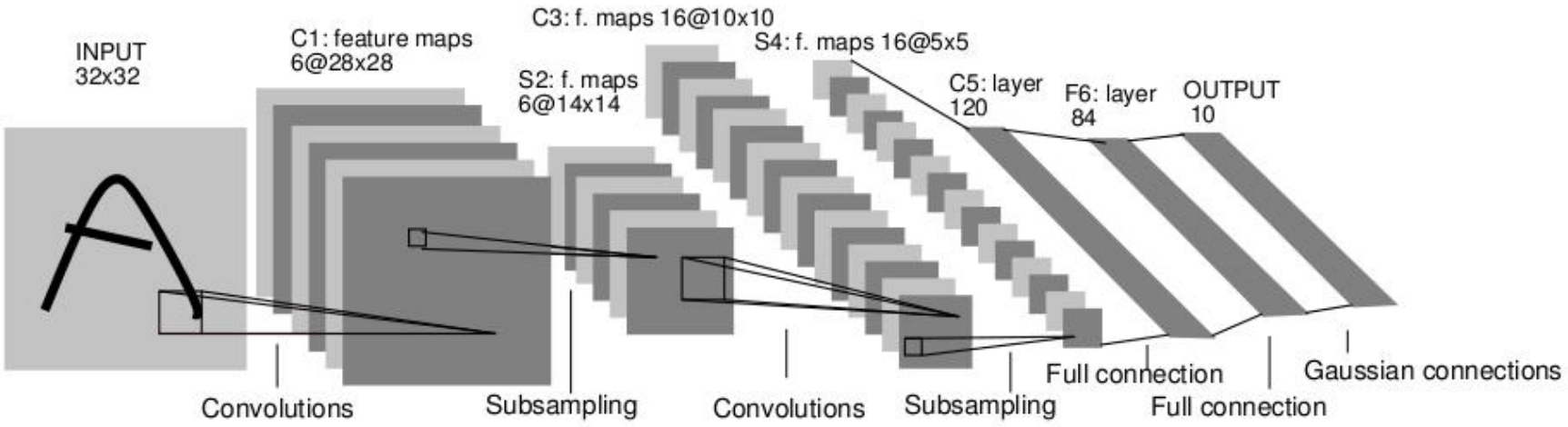


New Content!

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

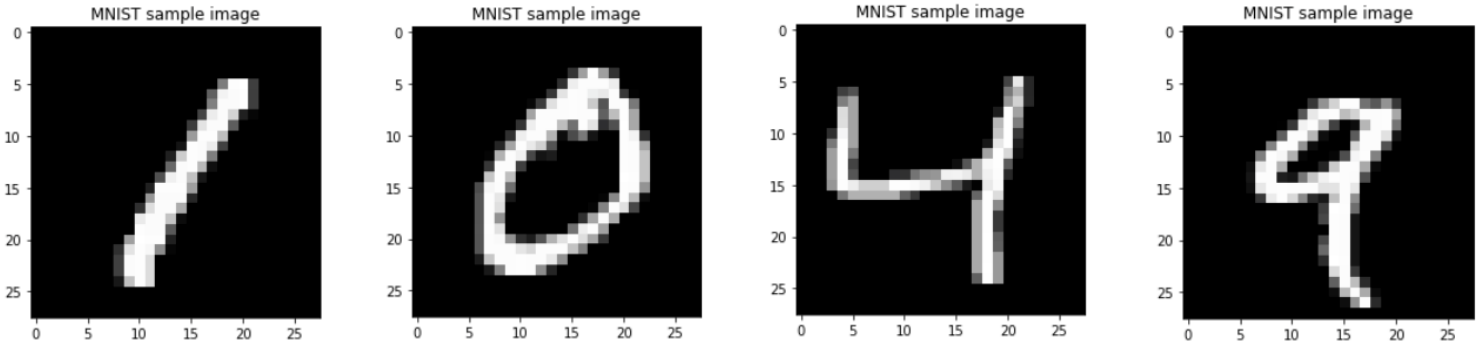
Network Structure of LeNet

- LeNet (introduced back in 1998) is a simple CNN architecture suitable for well-structured image
  - e.g., 32x32 pixels image of digits from 0 to 9 in MNIST or our Fashion-MNIST dataset



LeNet

- LeNet is a simple CNN architecture suitable for well-structured image
  - e.g., 32x32 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

- **ImageNet dataset**
  - 14 million labeled images
  - 20k classes

- **Data source**
  - Images gathered from Internet

- **Image annotators**
  - Human labels via Amazon Turk

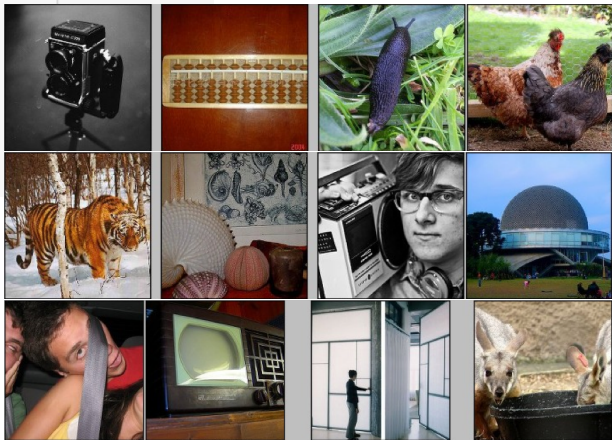
Project led by Fei-Fei Li at Stanford
CVPR 2009

- **What is ImageNet challenge**
  - Train your network with a subset of 1.2 million training images from ImageNet
  - Test your network using another testing subset where you need to classify each image to one of the 1000 classes

ImageNet Challenge

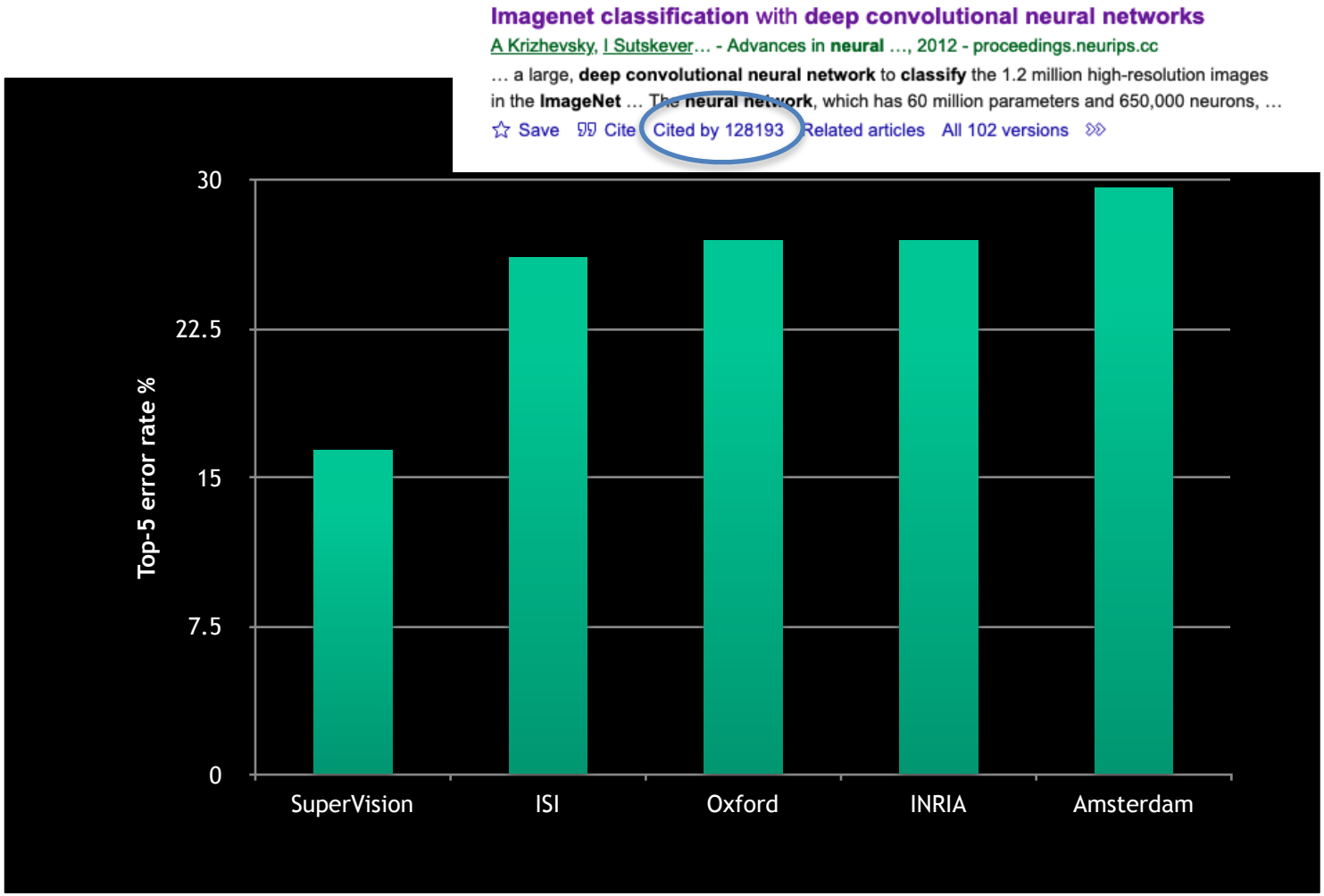
```
imagenet_1000_classes_label_map = \

{0: 'tench, Tinca tinca',
 1: 'goldfish, Carassius auratus',
 2: 'great white shark, white shark, man-eater, man-eating shark, Carcharodon carcharias',
 3: 'tiger shark, Galeocerdo cuvieri',
 4: 'hammerhead, hammerhead shark',
 5: 'electric ray, crampfish, numbfish, torpedo',
 6: 'stingray',
 7: 'cock',
 8: 'hen',
 9: 'ostrich, Struthio camelus',
 10: 'brambling, Fringilla montifringilla',
 11: 'goldfinch, Carduelis carduelis',
 12: 'house finch, linnet, Carpodacus mexicanus',
 13: 'junco, snowbird',
 14: 'indigo bunting, indigo finch, indigo bird, Passerina cyanea',
 15: 'robin, American robin, Turdus migratorius',
 16: 'bulbul',
 17: 'jay',
 18: 'magpie',
 19: 'chickadee',
 20: 'water ouzel, dipper',
```

IMAGENET



Project led by Fei-Fei Li at Stanford
CVPR 2009

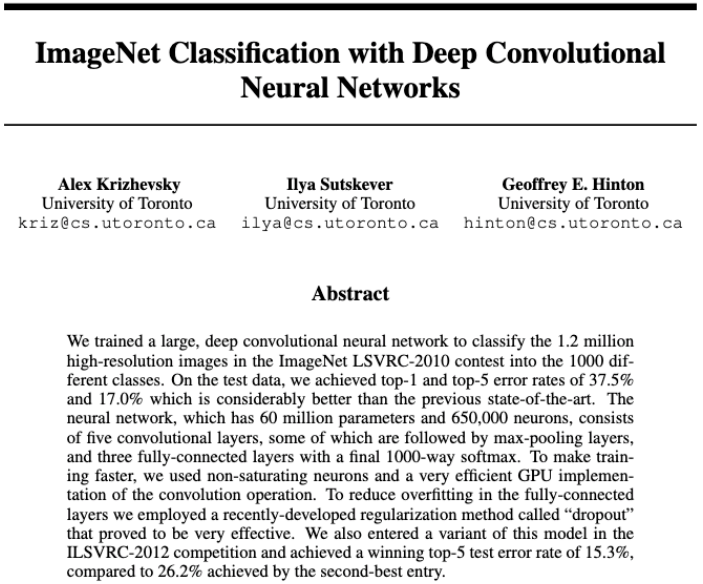
Recap: ImageNet Challenge 2012



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

Popular CNN: AlexNet

- AlexNet's important features. Similar framework to LeNet with important distinction such as:
  - Bigger model compared to LeNet
    - 5 convolution layers + 3 linear layers
    - 60,000,000 params
  - Trained on more data
    - 10^6 images (ImageNet) vs. 10^3 images (digit imag
  - GPU implementation (50x speedup over CPU)
    - trained on two GPUs for a week
  - Better regularization for training
    - introduced a new technique called **Dropout**
    - **Dropout:** randomly turning off neurons from the network



ImageNet Classification with Deep Convolutional Neural Networks

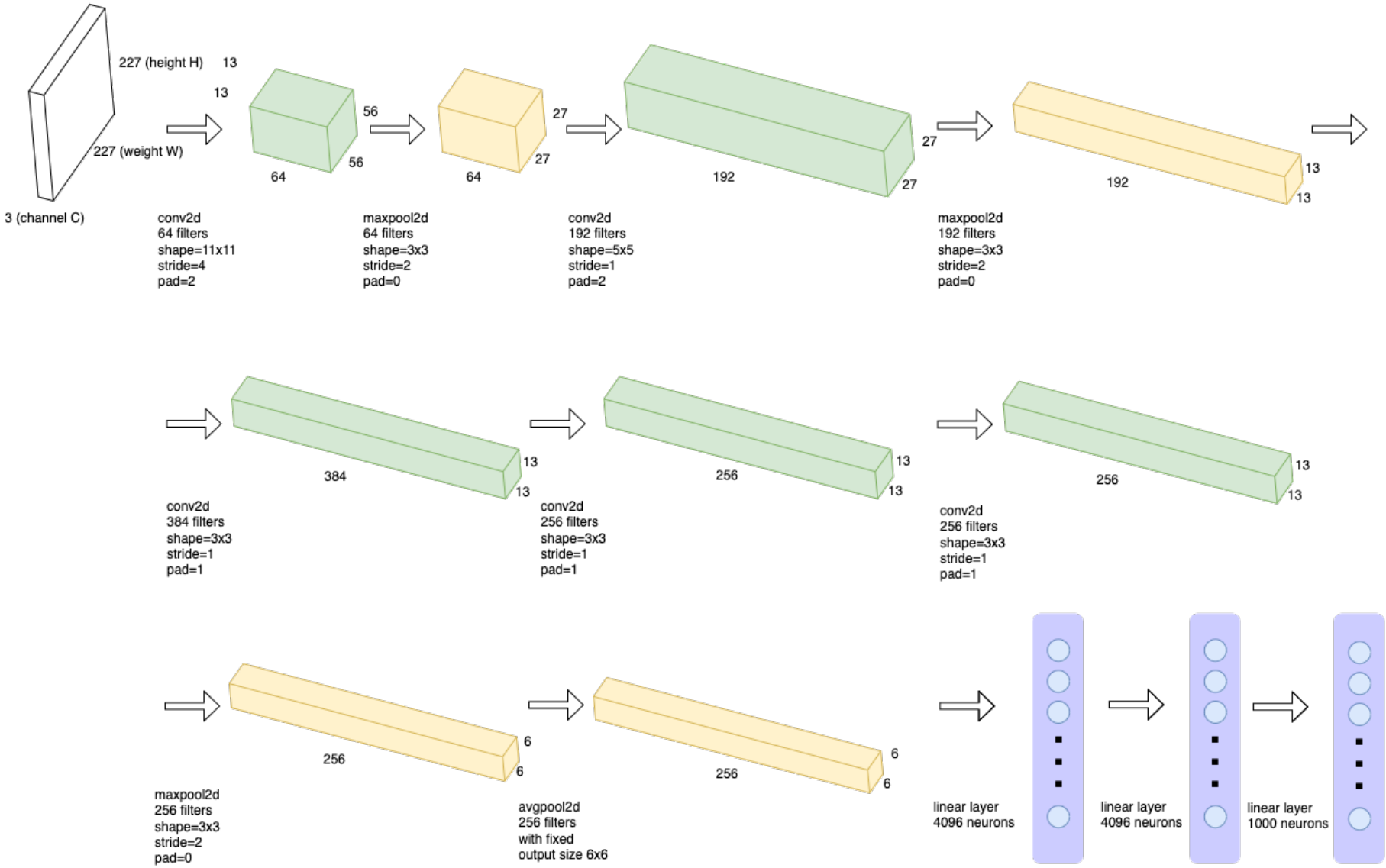
Alex Krizhevsky University of Toronto kriz@cs.utoronto.ca
Ilya Sutskever University of Toronto ilya@cs.utoronto.ca
Geoffrey E. Hinton University of Toronto hinton@cs.utoronto.ca

Abstract

We trained a large, deep convolutional neural network to classify the 1.2 million high-resolution images in the ImageNet LSVRC-2010 contest into the 1000 different classes. On the test data, we achieved top-1 and top-5 error rates of 37.5% and 17.0% which is considerably better than the previous state-of-the-art. The neural network, which has 60 million parameters and 650,000 neurons, consists of five convolutional layers, some of which are followed by max-pooling layers, and three fully-connected layers with a final 1000-way softmax. To make training faster, we used non-saturating neurons and a very efficient GPU implementation of the convolution operation. To reduce overfitting in the fully-connected layers we employed a recently-developed regularization method called "dropout" that proved to be very effective. We also entered a variant of this model in the ILSVRC-2012 competition and achieved a winning top-5 test error rate of 15.3%, compared to 26.2% achieved by the second-best entry.

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

AlexNet Network Architecture (PyTorch Library's implementation)



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

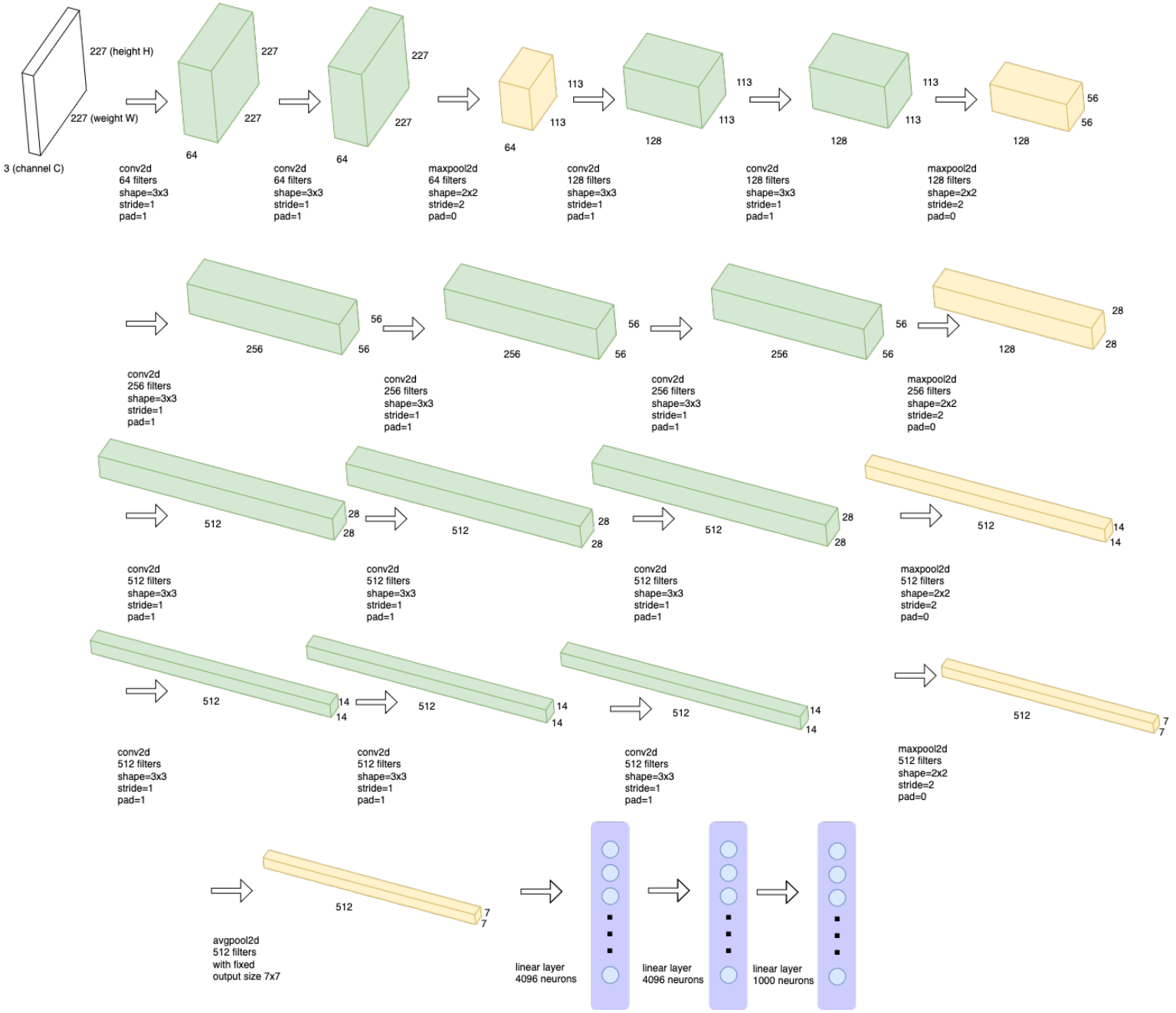
- Popular CNNs
  - LeNet
  - AlexNet
  - VGG
  - ResNet
- Training vs. Fine-tuning
- Fine-tuning a popular CNN (eg, AlexNet) using an arbitrary dataset

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

VGG-16 Network Architecture (PyTorch Library's implementation)



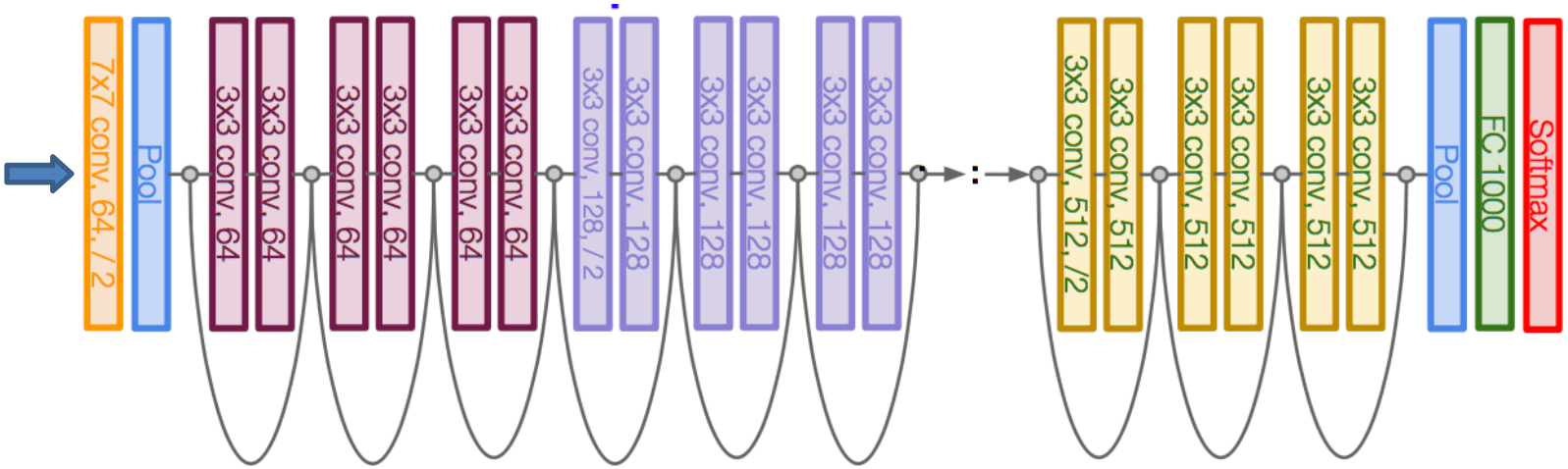
Today's Agenda

- 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

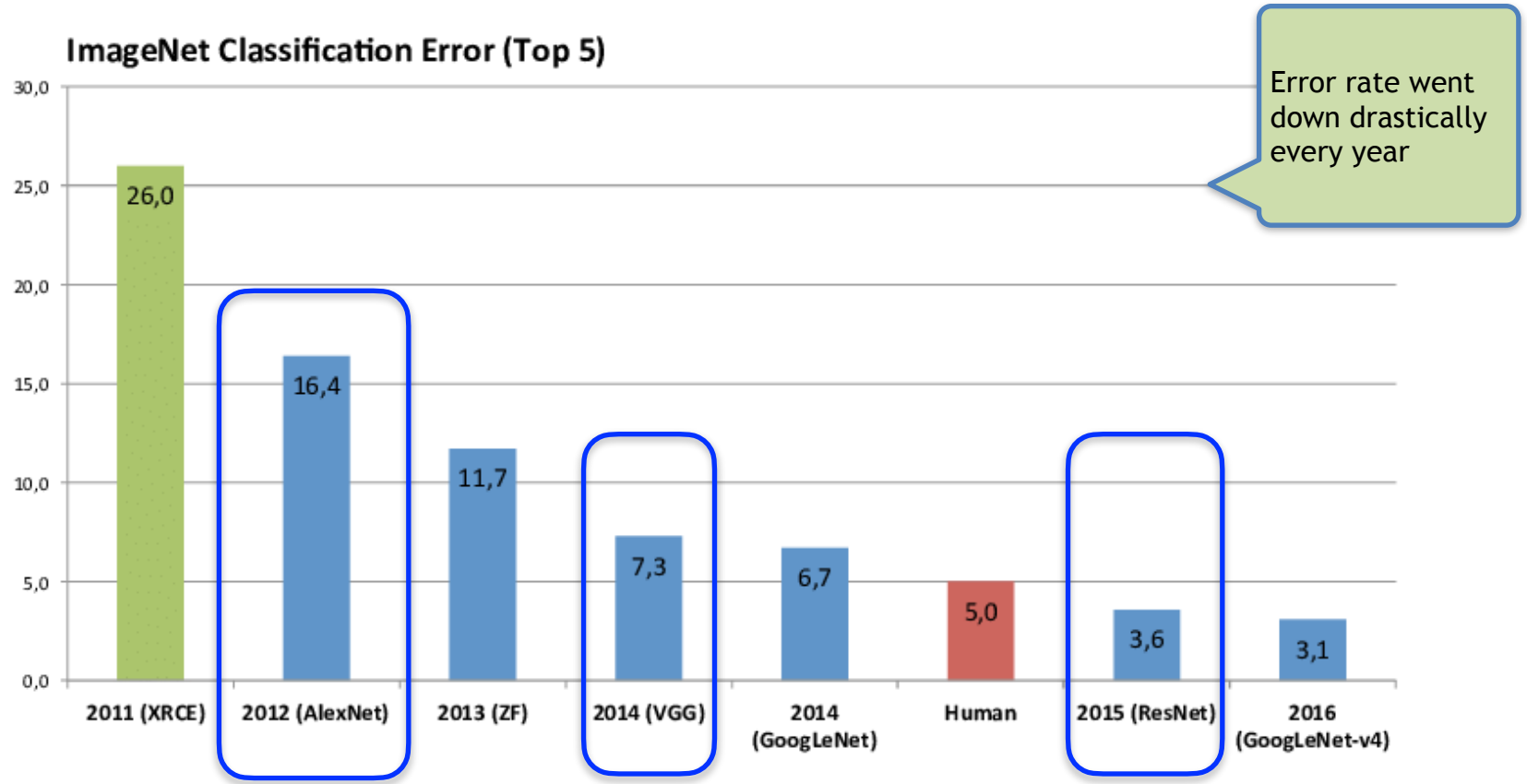
Input image



[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)

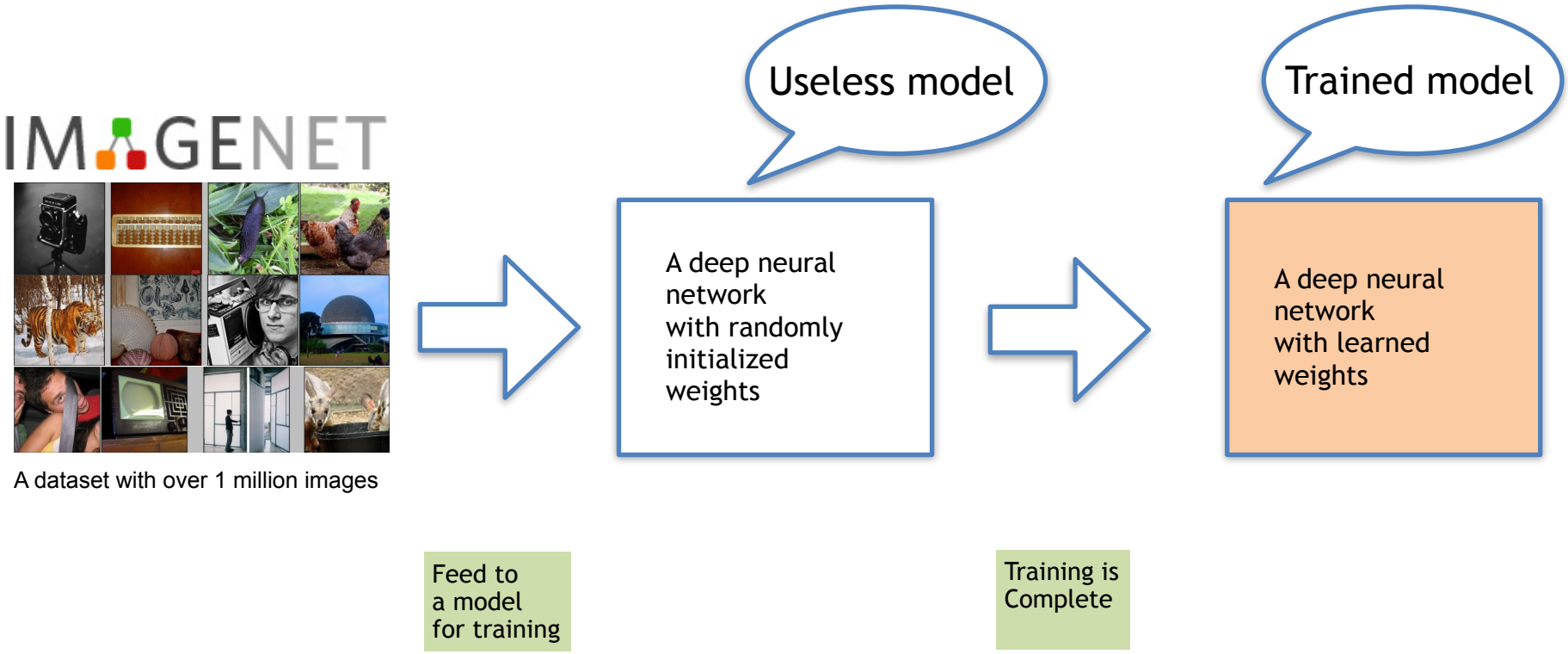


Today's Agenda

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

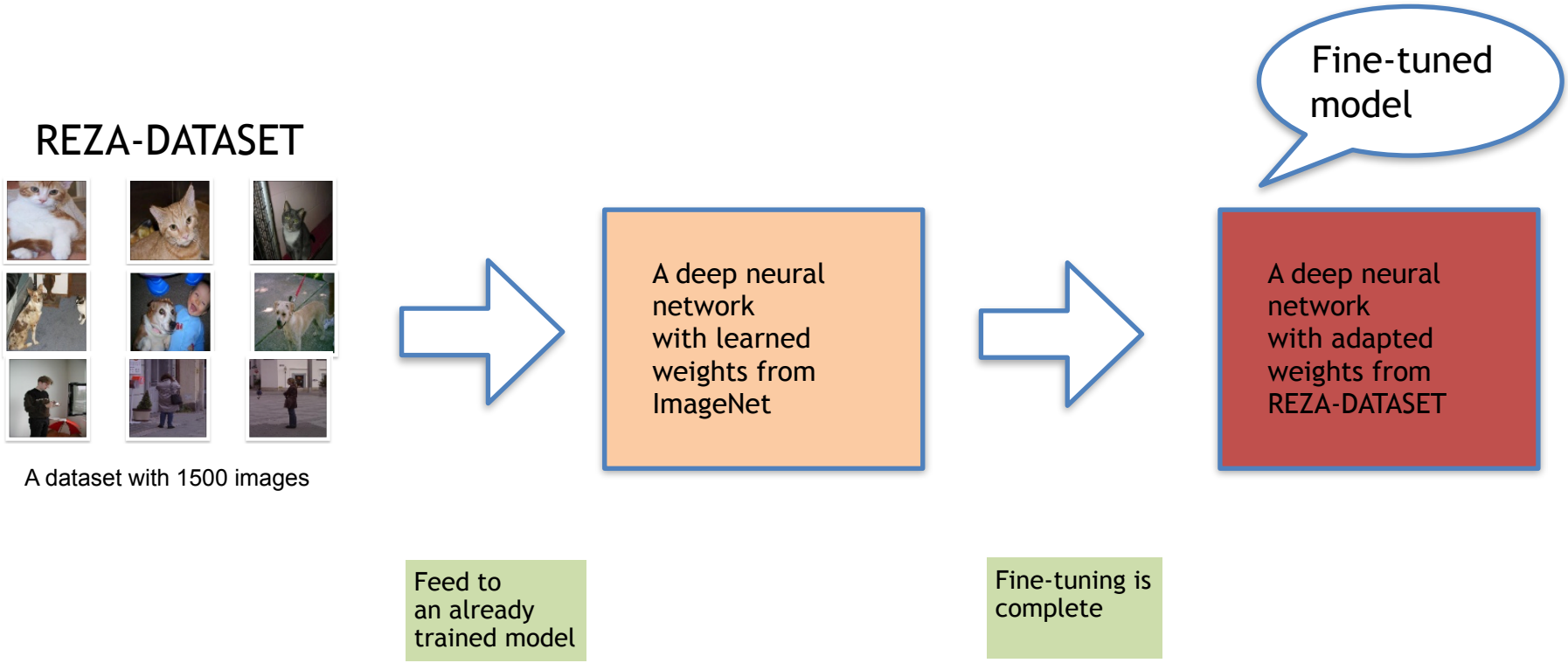
Training a Model

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



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.

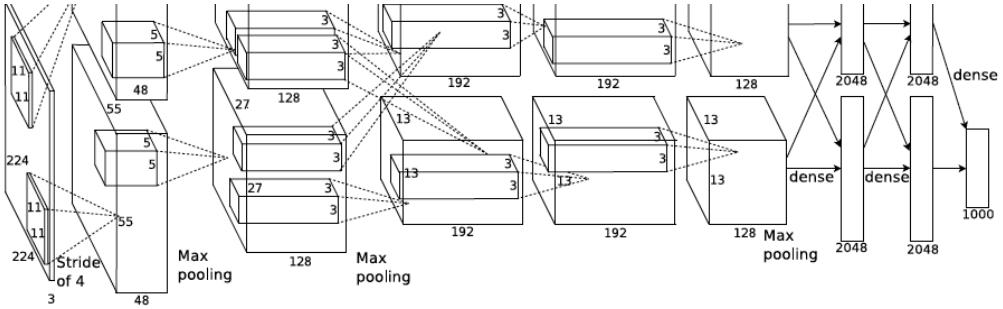


Today's Agenda

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

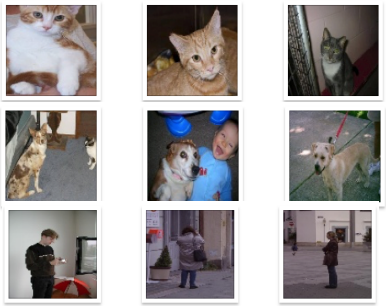
Fine-tuning AlexNet on an Arbitrary Dataset

- 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.

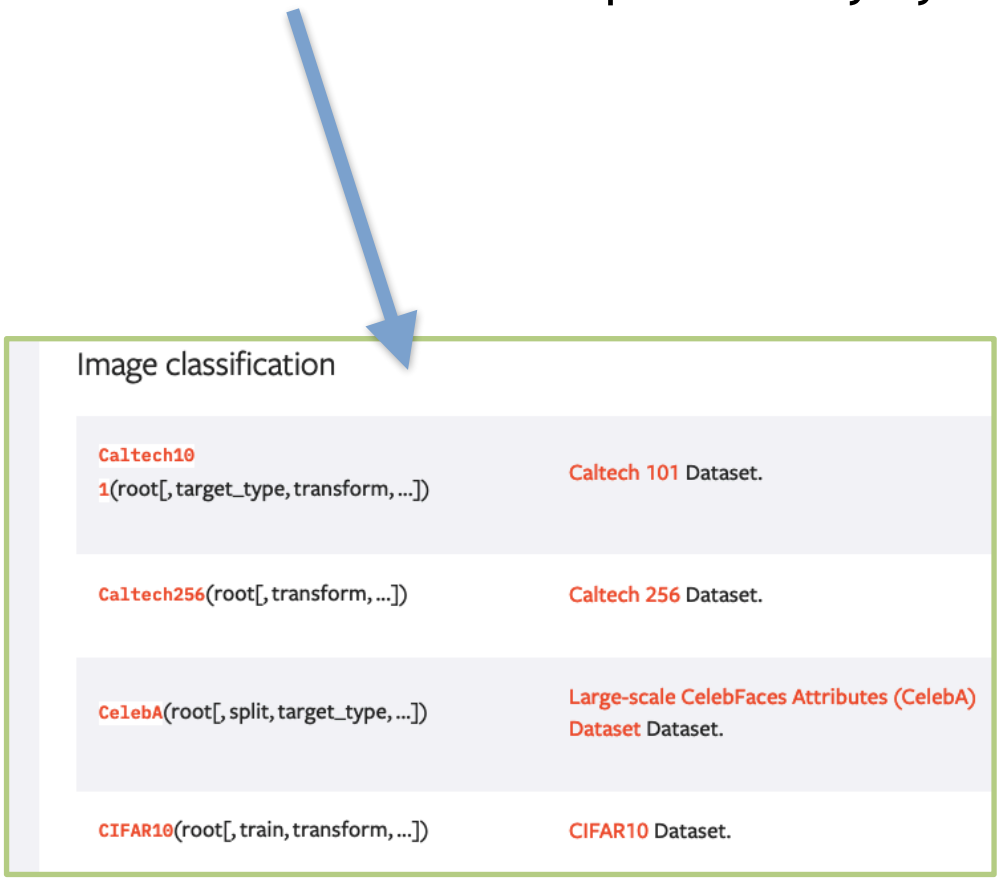


Image classification

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_fall24/dataset/bcdp_v1.zip

Fine-tuning AlexNet on an Arbitrary Dataset

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

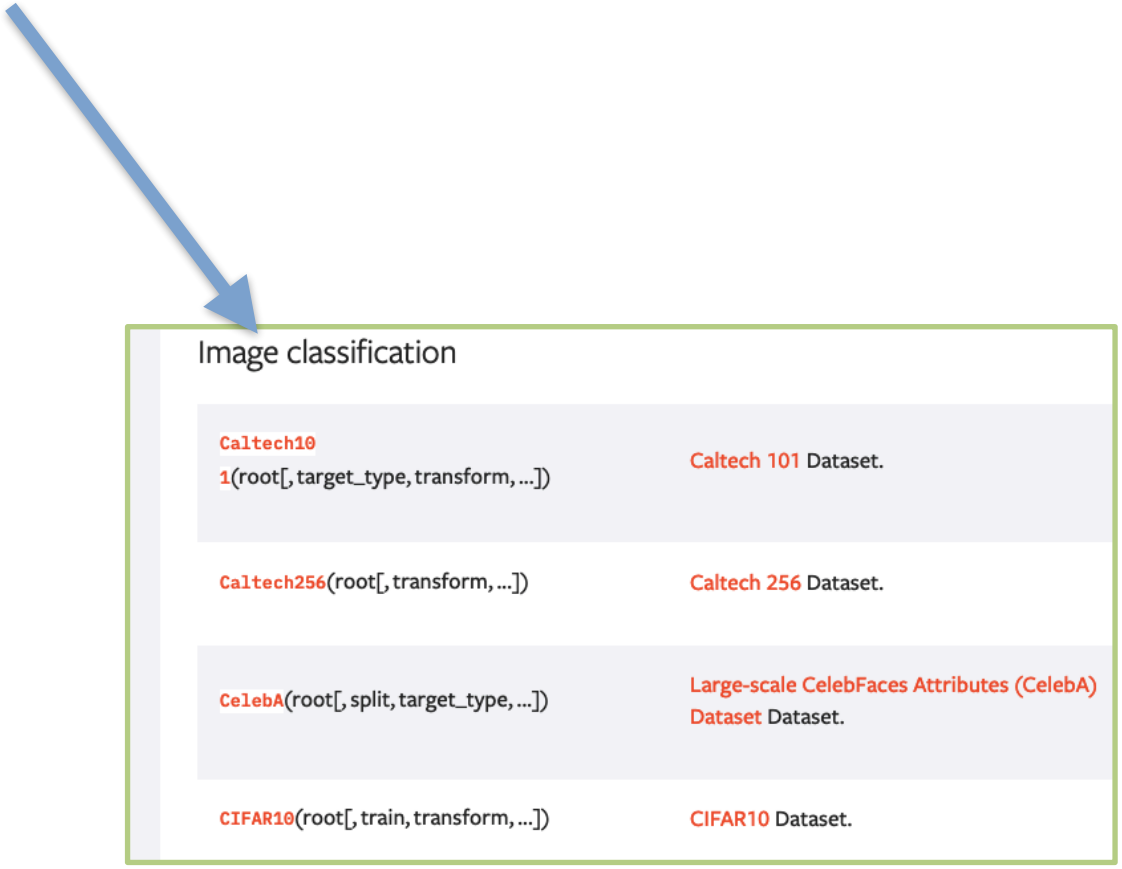
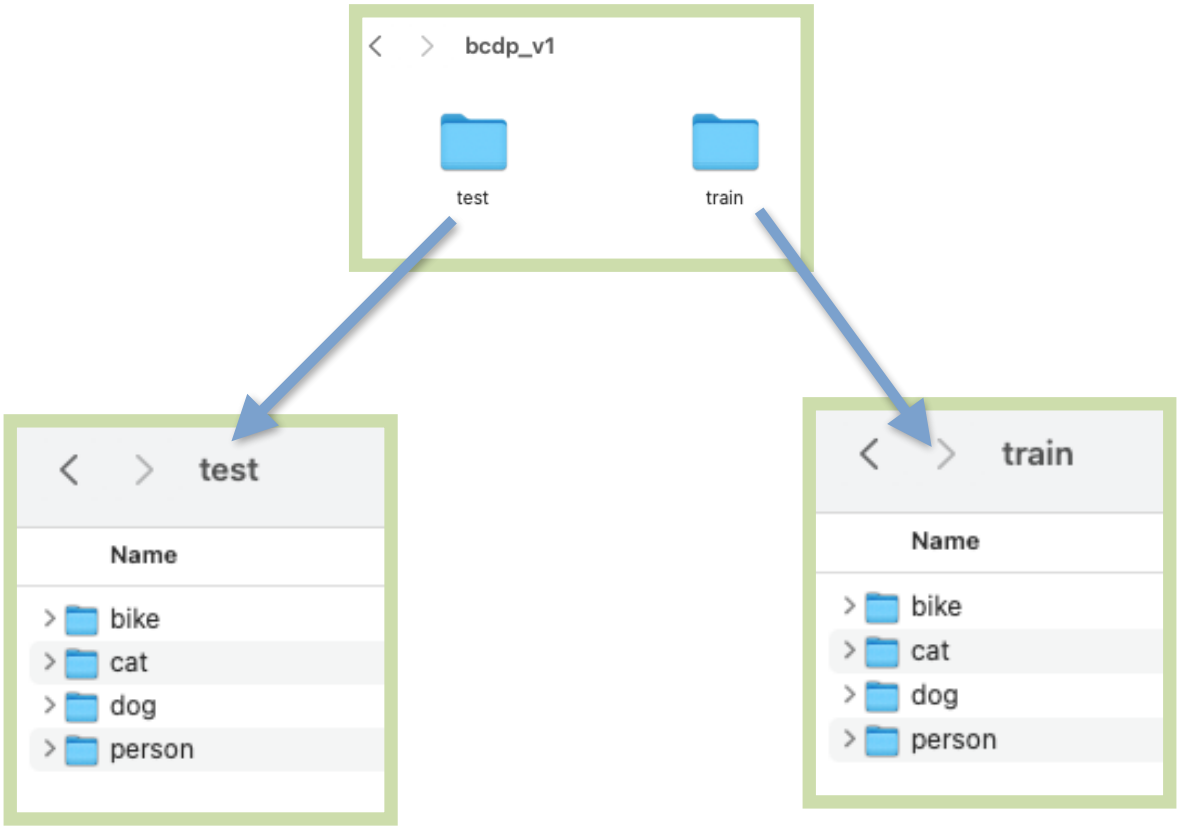


Image classification

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.

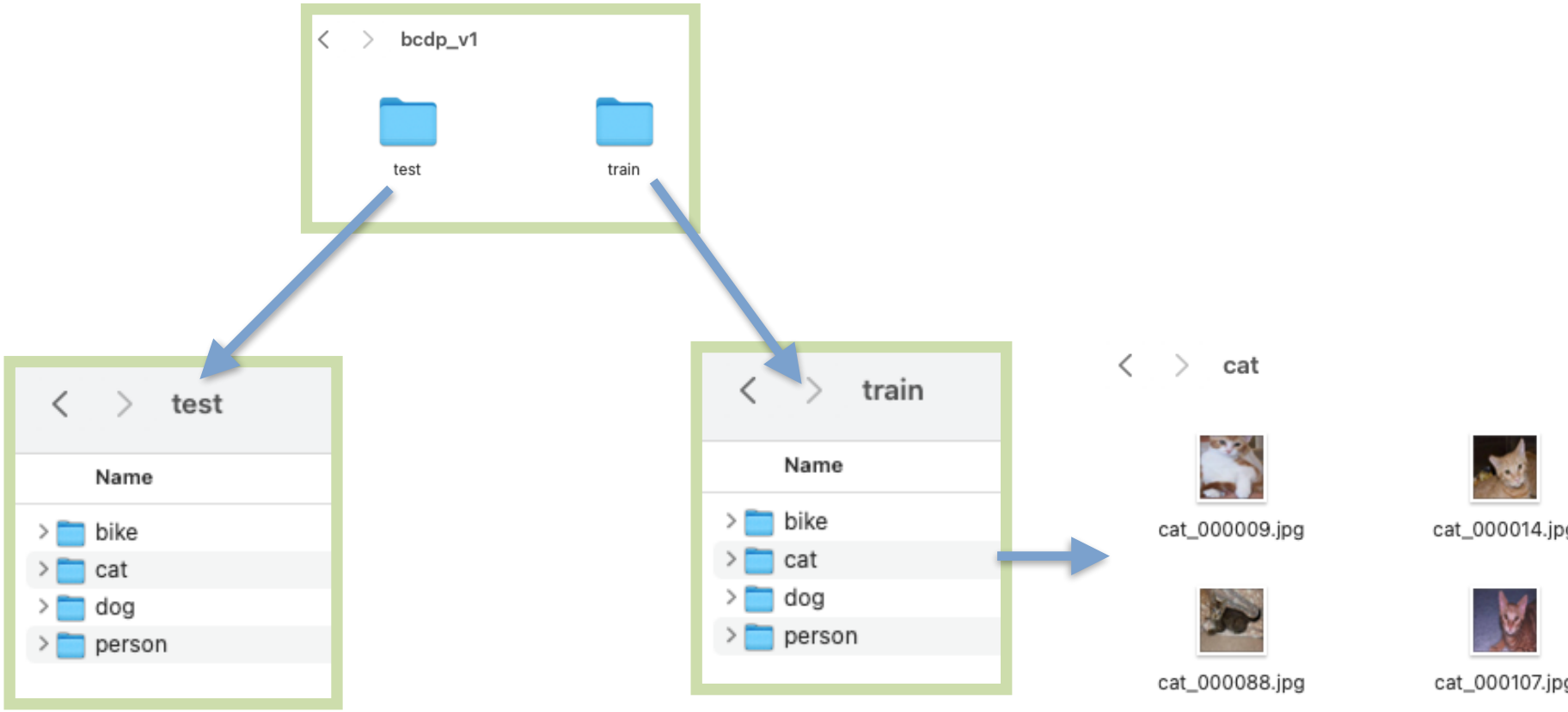
Fine-tuning AlexNet on an Arbitrary Dataset

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



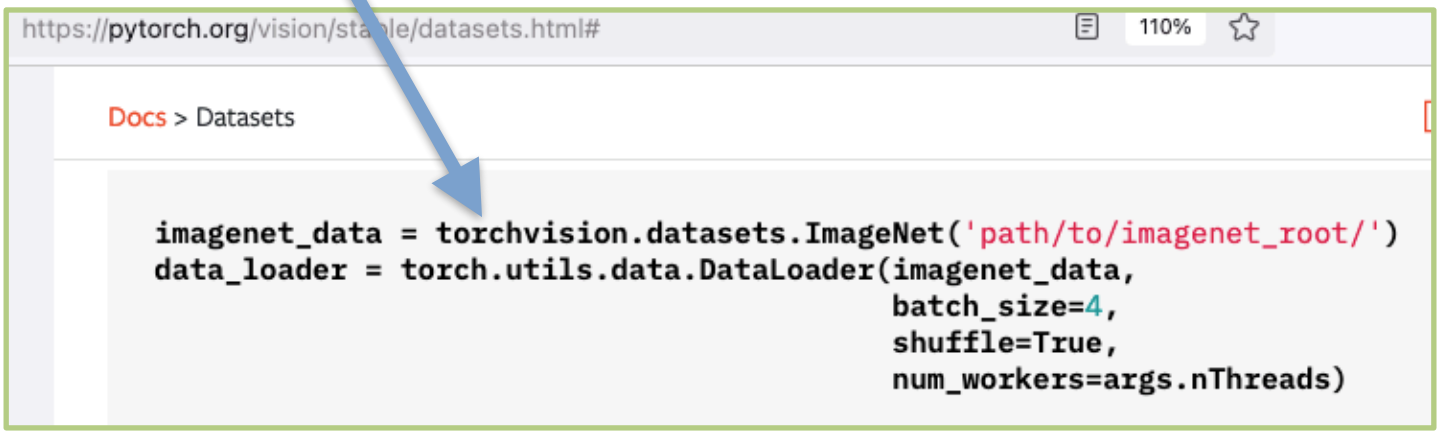
Fine-tuning AlexNet on an Arbitrary Dataset

- 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.



- 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: mean (R, G, B) and standard deviation (R, G, B)
    ])


    train_dir       = '/content/drive/MyDrive/cs167_sp25/datasets/bcdp_v1/train'
    test_dir        = '/content/drive/MyDrive/cs167_sp25/datasets/bcdp_v1/test'

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

    N_train         = len(train_dataset)
    N_test          = len(test_dataset)

    number_of_classes = 4

    print("Size of train set:", N_train)
    print("Size of test set:",  N_test)
```

```
Size of train set: 1500
Size of test set: 300
```

```
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: