15-494/694: Cognitive Robotics

Dave Touretzky

Lecture 14:
ImageNet and Transfer Learning

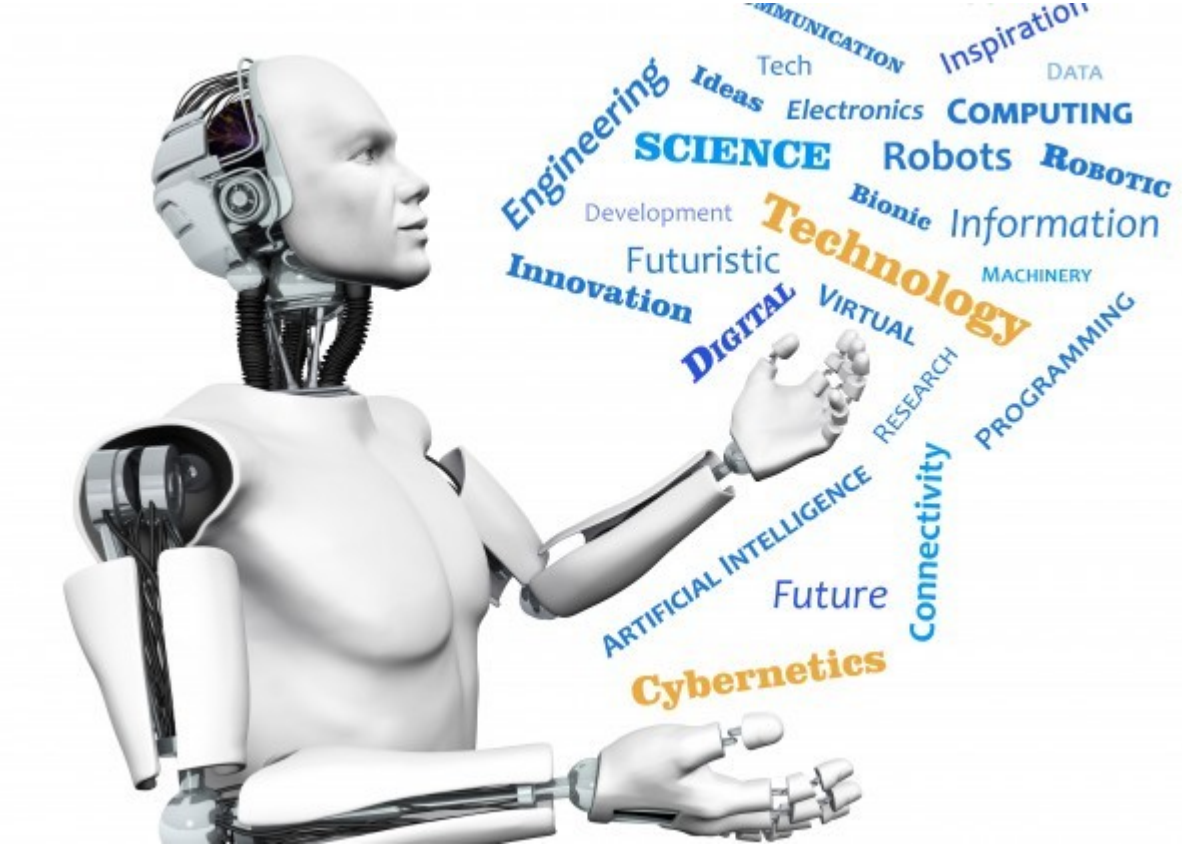


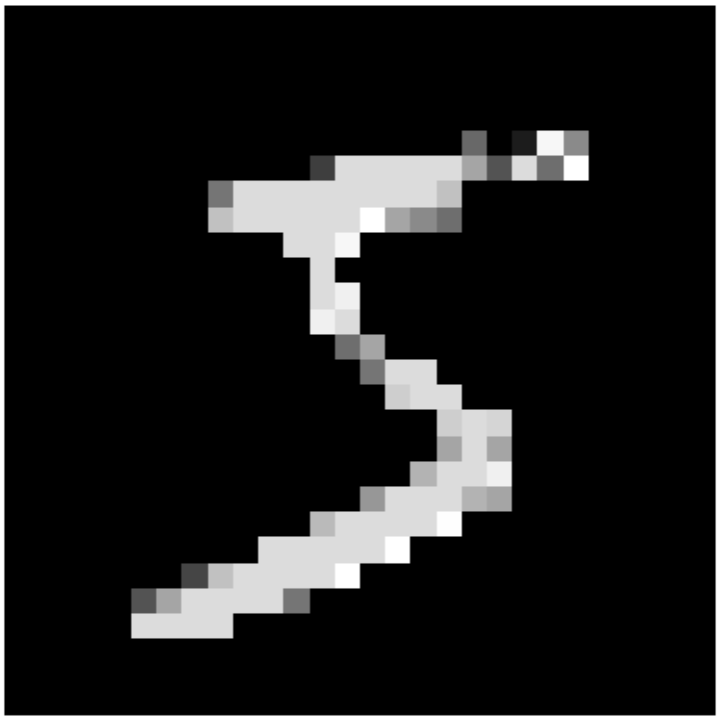
Image from http://www.futuristgerd.com/2015/09/10

Training With Pytorch

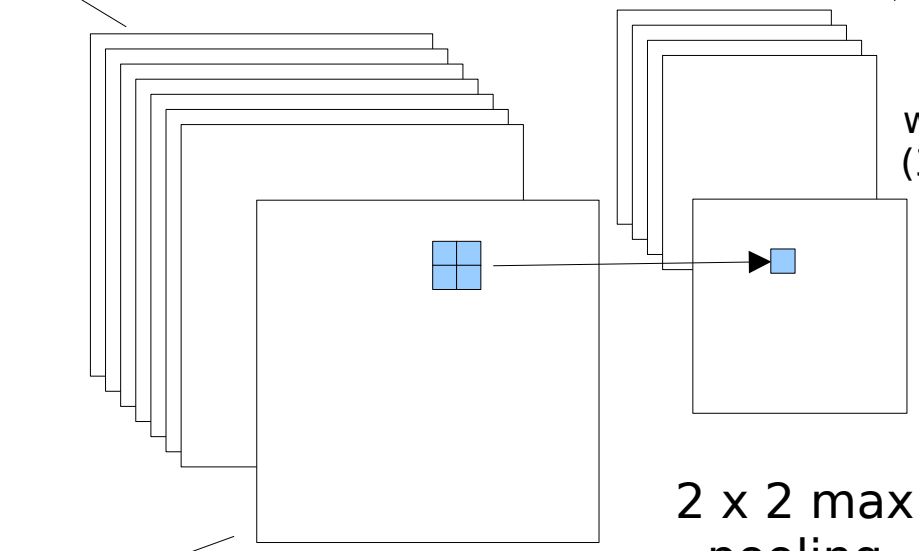
Components needed to train a classifier:

- Model:
  - Specify the input and output size
  - Define the layers and connections
  - Perform forward propagation
- Dataset loader: provides the training data
- Loss criterion: how we measure error
- Optimizer: updates the model parameters

MNIST With A CNN



28 x 28 image



32 kernels 5x5 pixels
stride 1
padding 2
ReLU nonlinearity
image maps 28 x 28
batch normalization

2 x 2 max
pooling

32 image maps
14 x 14

fully
connected
weight matrix
(32*14*14+1)
x 10

Cross-
entropy
loss
function

parameters = 63,626
How many connections?

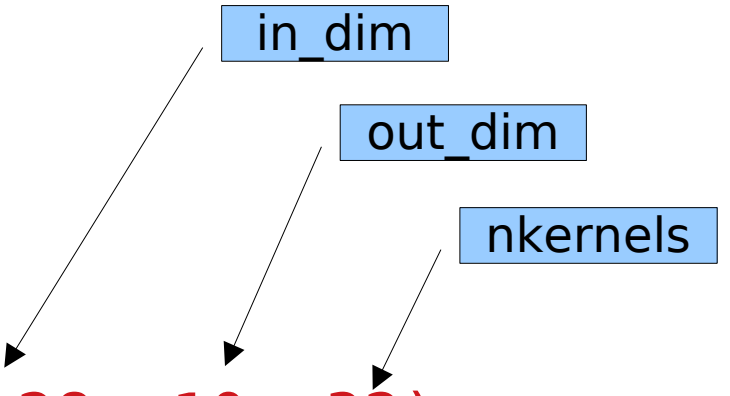
Accuracy on training set: 98.7%

Defining the Model mnist3

```
class OneConvLayer(nn.Module):

  def __init__(self, in_dim, out_dim, nkernels):
    super(OneConvLayer, self).__init__()
    self.network1 = nn.Sequential(
      nn.Conv2d(in_channels=1,
                out_channels=nkernels,
                kernel_size=5,
                stride=1,
                padding=2),
      nn.BatchNorm2d(nkernels),
      nn.ReLU(),
      nn.MaxPool2d(kernel_size=2)
    )
    self.network2 = nn.Linear(nkernels*14*14,
                              out_dim)
```

Defining mnist3 (cont.)

```
  def forward(self, x):
    out = self.network1(x)
    out = out.view(out.size(0), -1)
    out = self.network2(out)
    return out
```




```
model = OneConvLayer(28*28, 10, 32)
```

Automatic Differentiation

- Each layer of the model (Conv2D, ReLU, MaxPool, Linear) knows how to calculate its own derivative.

- When each layer produces its output (a tensor), that tensor is given attributes that allow backpropagation of the gradient.
  - This is another way that tensors differ from ordinary numpy arrays.

Dataset Loader

- Reads in training data from a file
- Supplies data in chunks according to the batch size we specify
- Shuffles the data if asked to do so

```
trainset = torchvision.datasets.MNIST(
                root='./mnist_data',
                download = True,
                transform = transforms.ToTensor())

trainloader = torch.utils.data.DataLoader(
                    dataset = trainset,
                    batch_size = batchSize,
                    shuffle = True)
```

Loss Functions

How do we measure error?

- Mean Square Error: nn.MSELoss

$$E = \frac{1}{2P}\sum_{p}\left(d^{p}-y^{p}\right)^{2}$$

- Cross-Entropy: nn.CrossEntropyLoss

$$E = \sum_{p} -d^{p}\log\left(y^{p}\right)-\left(1-d^{p}\right)\log\left(1-y^{p}\right)$$

- Lots of other choices.

```
criterion = nn.CrossEntropyLoss()
```

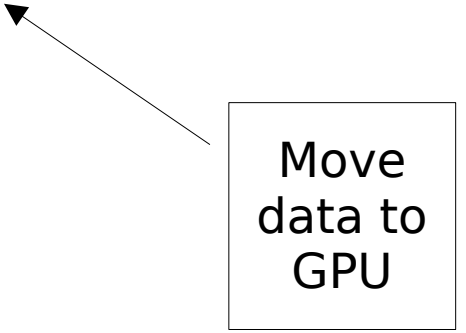
Optimizers

- Once we've measured the error gradient, what do we do about it?
- An optimizer adjusts the weights based on the error gradient and various parameters: learning rate, momentum, etc.
- Lots of choices: SGD, ADAM, etc.

```
optimizer = torch.optim.SGD(model.parameters(), lr=0.005)
```

Training the Model

```
device = torch.device(0) # or torch.device('cpu')

for epoch in range(nepochs):

  for (images,labels) in trainloader:

    images = images.view(-1, 28*28).to(device)
    labels = labels.to(device)
    outputs = model(images)

    optimizer.zero_grad()
    loss = criterion(outputs, labels)
    loss.backward()

    optimizer.step()
```



Object Recognition

Object Recognition Challenge

- Computer vision researchers use challenge events to measure progress in the state of the art.
- PASCAL VOC (Visual Object Classes) Challenge:
  - Ran from 2005 to 2012
  - 2005 version had 4 categories (bicycles, motorcycles, people, cars) and 1,578 training images
  - 2012 version had 20 categories and 5,717 training images

ImageNet

- Created by Fei-Fei Li at Stanford.
- See www.image-net.org
- 15 million labeled images, 22,000 categories
- ILSVRC: ImageNet Large Scale Visual Recognition Challenge: 2009-2017
  - 1000 categories, including 118 dog breeds
  - 1.2 million training images

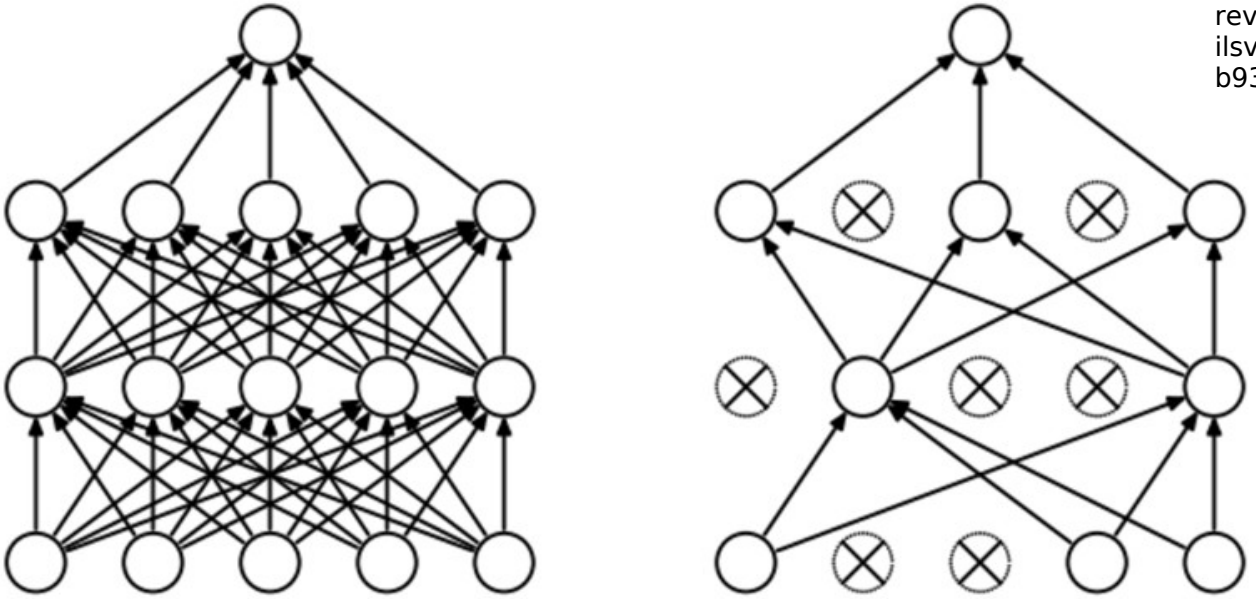
AlexNet

- The winners of the 2012 ILSVRC:
  - Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton
  - Deep convolutional neural net (DCNN) called AlexNet
  - Trained using two GPU boards
  - Introduced ReLU in place of tanh
  - Used “dropout” to reduce overfitting
  - Error rate of 15.3% was 10% better than the runner-up
  - Put deep neural nets on the map

Dropout in AlexNet

- For each training step, disable 50% of the neurons for both the forward and backward pass.
- Reduces overfitting.

Figure from https://medium.com/coinmonks/paper-review-of-alexnet-caffenet-winner-in-ilsvrc-2012-image-classification-b93598314160



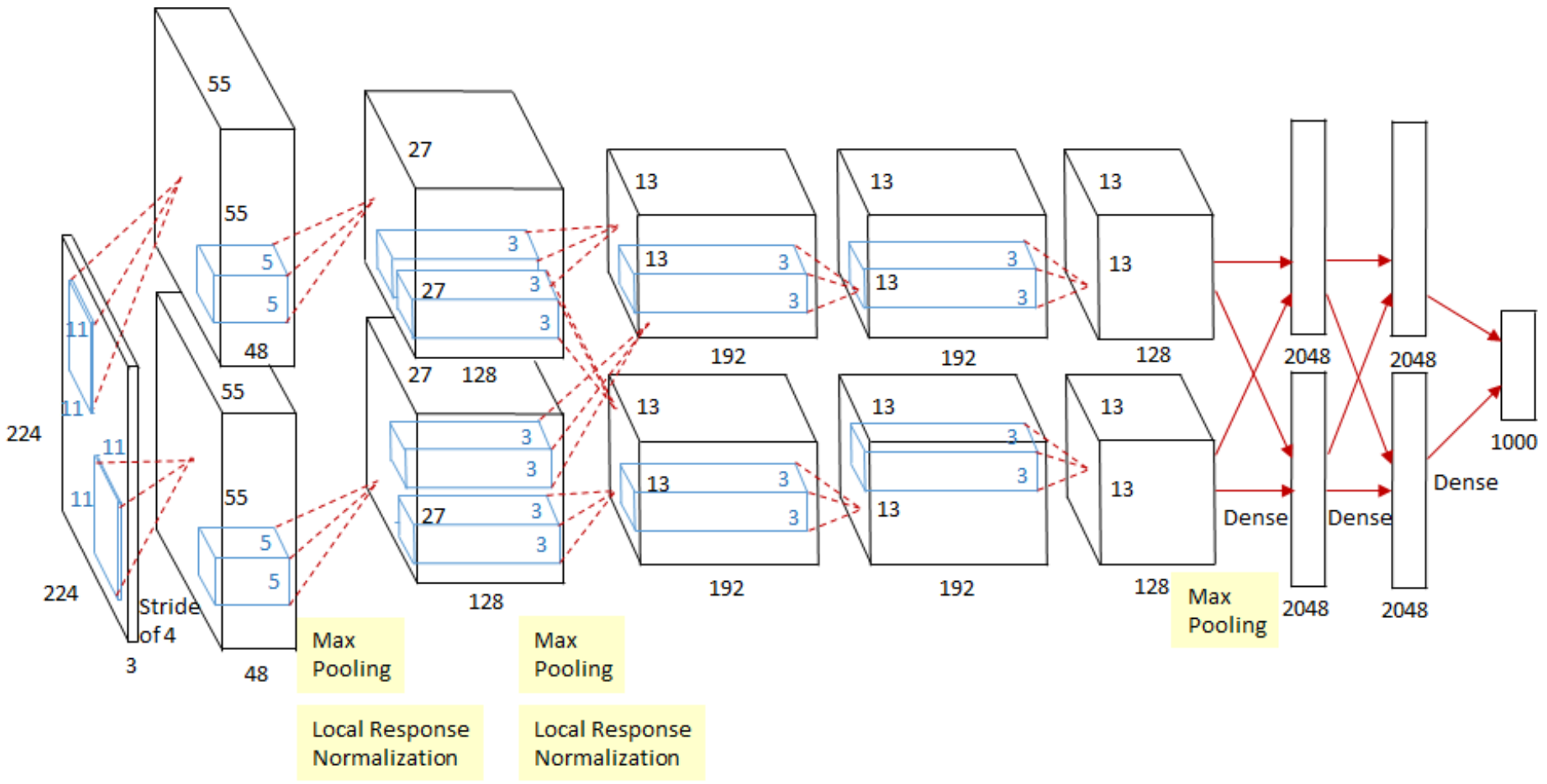
(a) Standard Neural Net

(b) After applying dropout.

Data Augmentation in AlexNet

- Take random 224x224 crops of a 256x256 image, plus their horizontal reflections. Increases training set size by a factor of $32^2 \times 2 = 2048$.
- Add random factors to RGB values to simulate variations in lighting.
- These steps help the network generalize better.

AlexNet Architecture



All hidden layers were split in two and trained on different GPU boards due to GPU memory limitations.

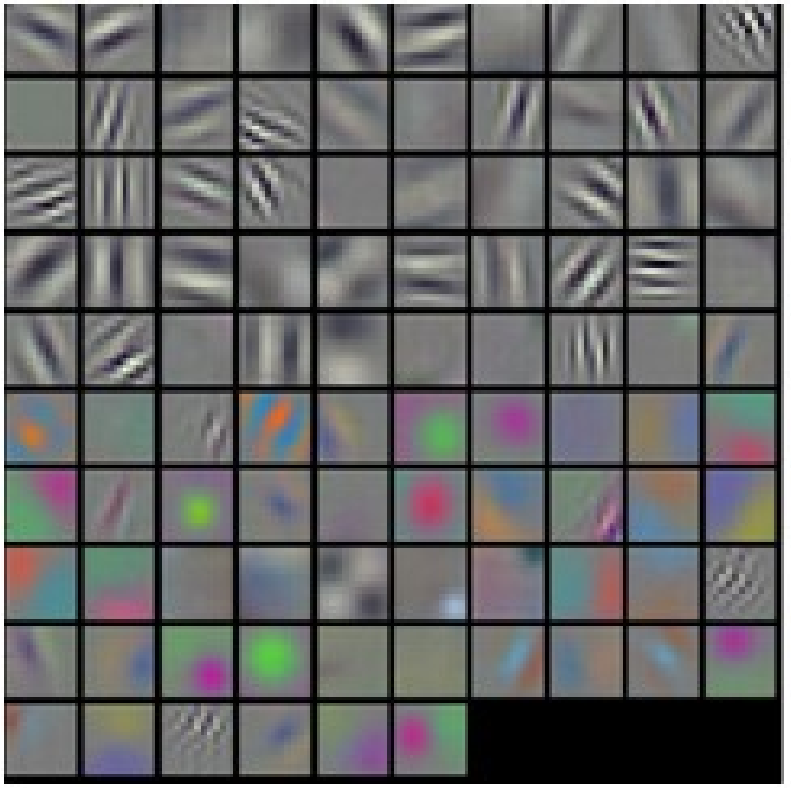
AlexNet Layer 1 Kernels

AlexNet's 96 11x11 layer 1 kernels.

First 48 trained on GPU 1 look for edges.

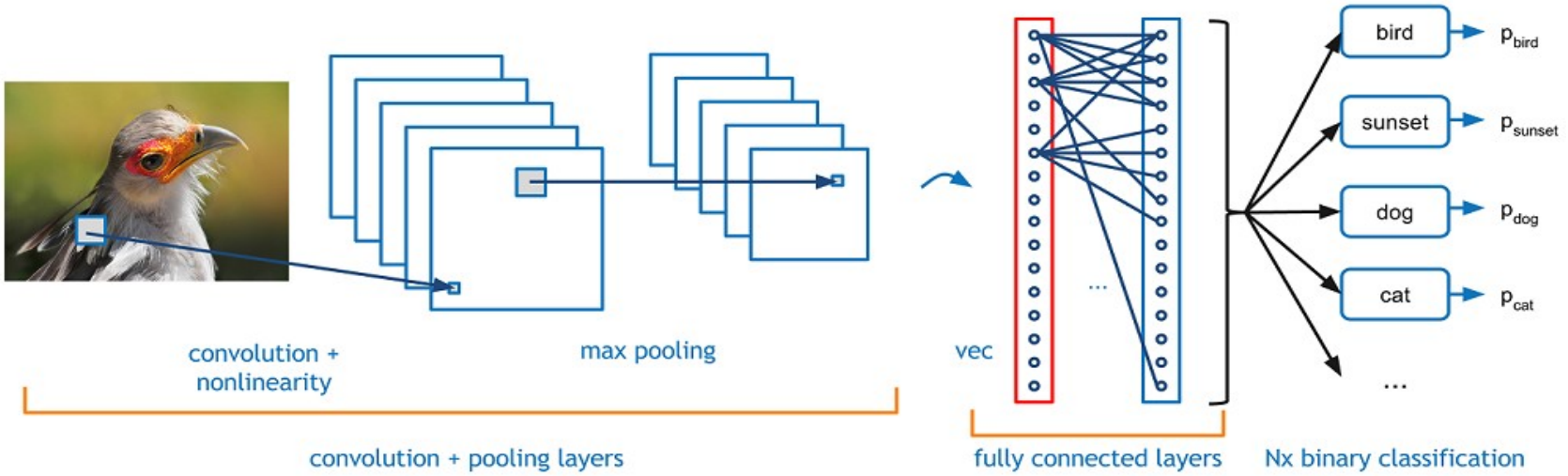
Second 48 trained on GPU 2 look for color.

This separation is a natural consequence of the normalization terms in the early layers.



Visualizations of filters

Generic Object Recognition CNN



https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/

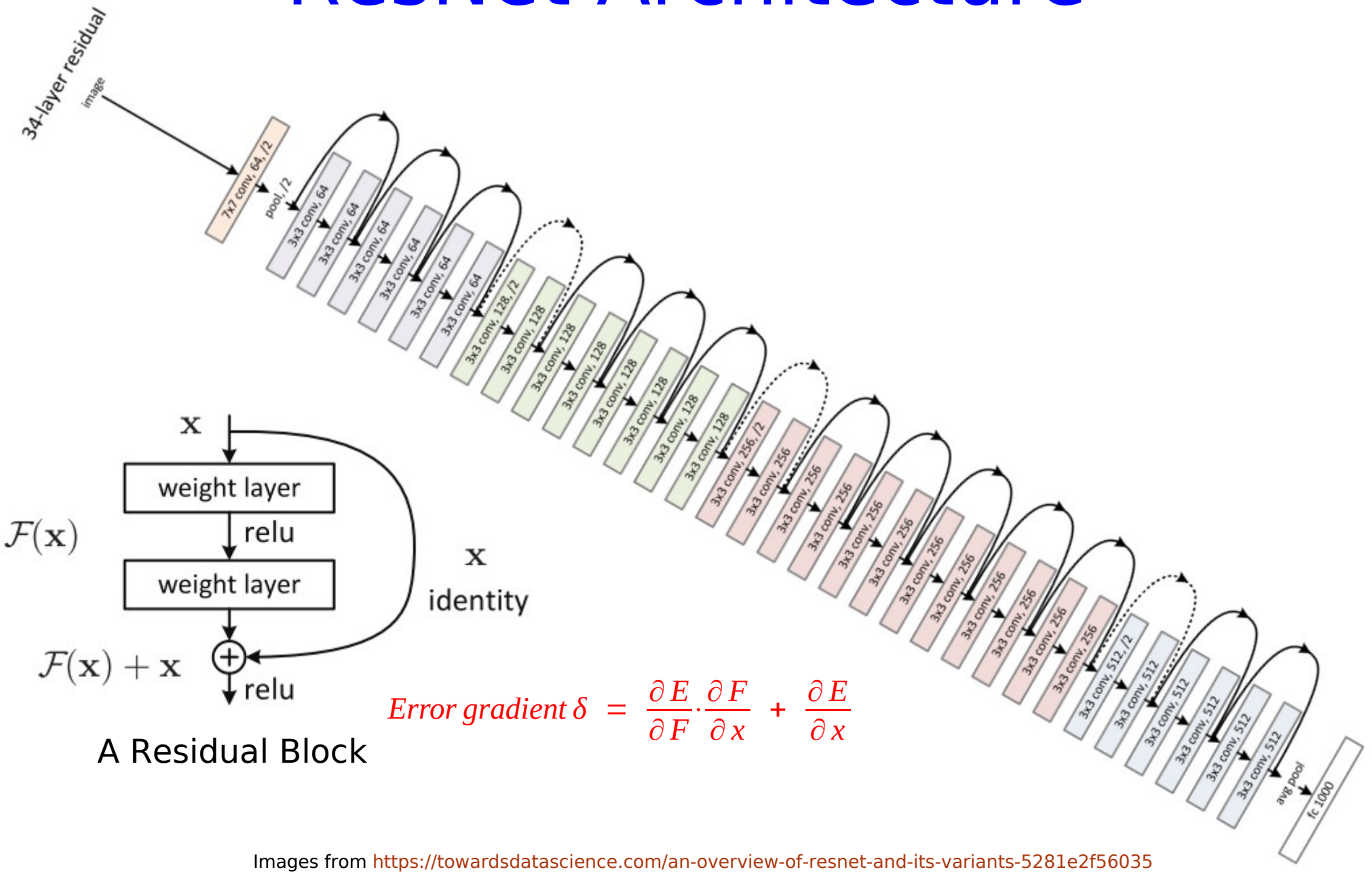
After AlexNet

- AlexNet had 8 layers: 5 convolutional and 3 fully connected.
- In 2015 Microsoft won the ILSVRC using a deep neural network with 100 layers.
- By the end of the ILSVRC in 2017, the best entrants were seeing accuracies of over 95% (error rate < 5%).

Residual Blocks

- Residual blocks were introduced in ResNet:
  - For really deep networks, it's hard for the error signal to propagate backwards through many layers.
  - Solution: add shortcut connections, e.g., from layer i to layer i+2, so that error can back-propagate more quickly.
  - A residual block contains hidden layers with a shortcut connection.

ResNet Architecture



A Residual Block

$$Error\ gradient\ \delta = \frac{\partial E}{\partial F}\cdot\frac{\partial F}{\partial x} + \frac{\partial E}{\partial x}$$

Images from https://towardsdatascience.com/an-overview-of-resnet-and-its-variants-5281e2f56035

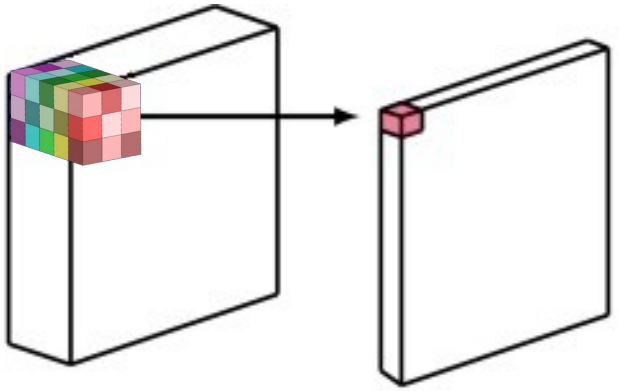
Mobile Implementations

- People want to implement computer vision on mobile phones. Networks must be reduced in size.

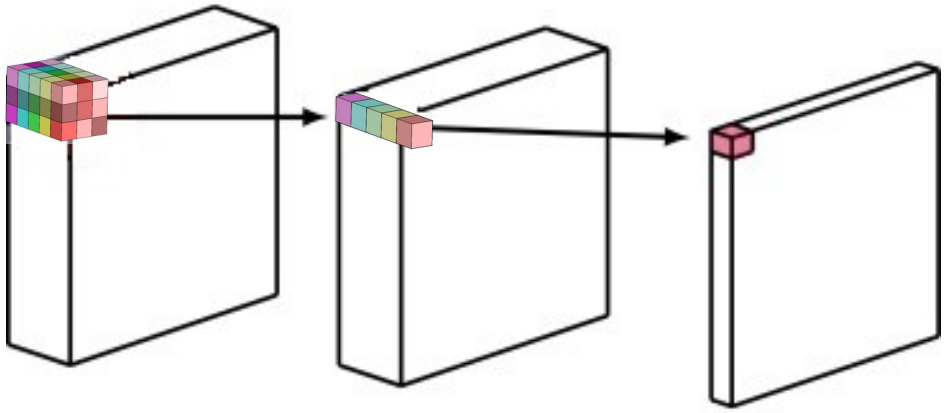
- Various architectures explore ways to reduce the size of the network and the number of multiply-add operations.
  - Separable convolutions
  - Bottlenecks

- Examples: MobileNet, SqueezeNet

Separable Convolutions



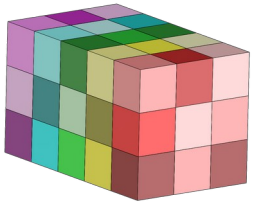
(a) Conventional Convolutional Neural Network



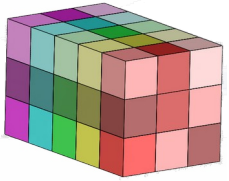
(b) Depthwise Separable Convolutional Neural Network

3x3x6 kernel covering 6 channels

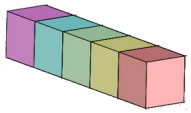
3x3x6 = 54 weights



Depthwise convolution: one 3x3 kernel applied to all 6 channels



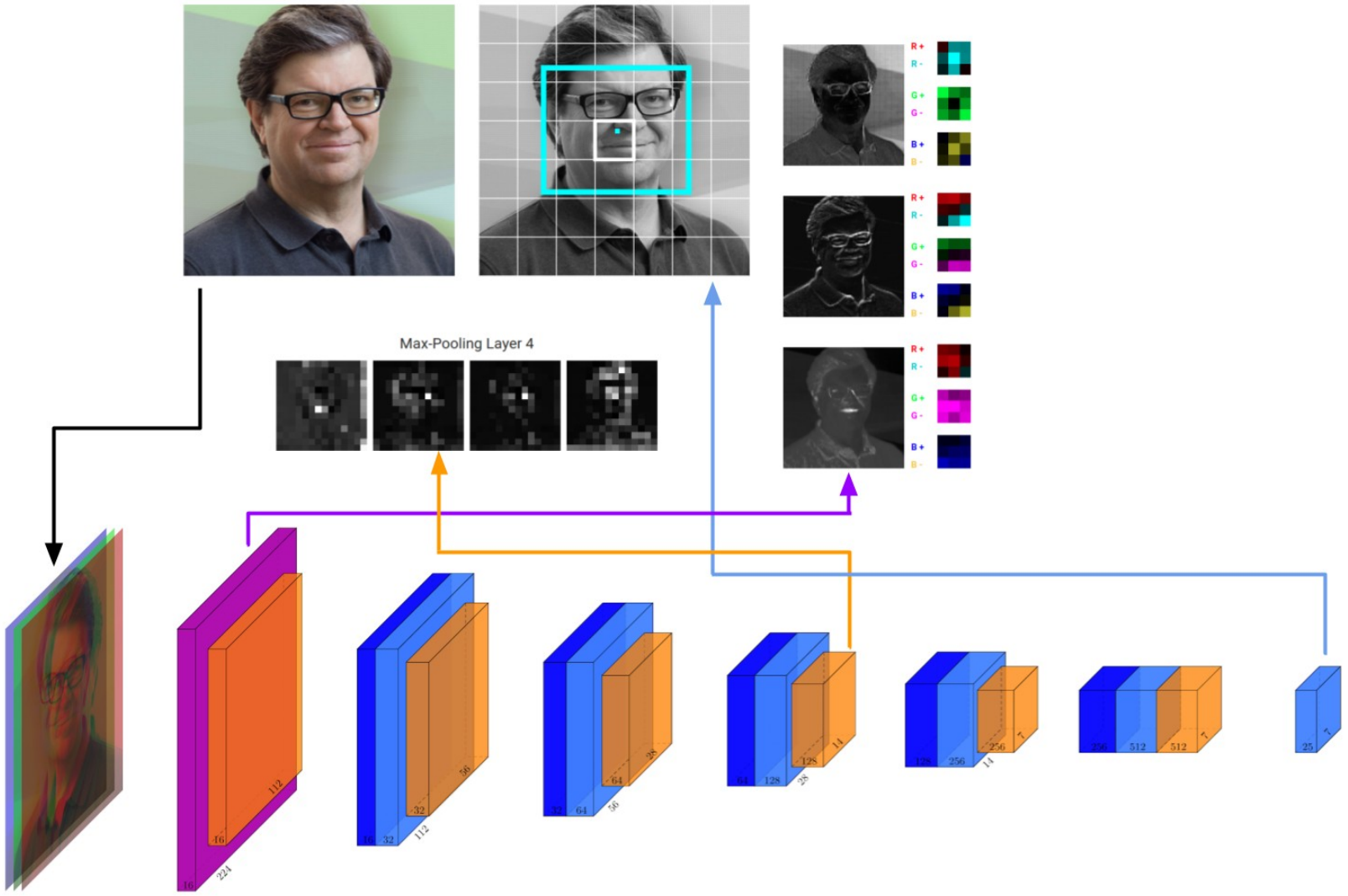
Pointwise convolution: linear weighted combination of one pixel's values across all 6 channels



Combined network:
3x3 + 6 = 15 weights

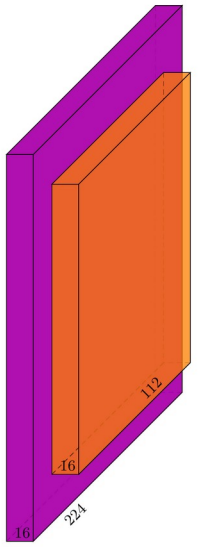
Image modified from
https://www.sciencedirect.com/science/article/pii/S0168169918318696

TinyYOLOV2 Face Recognition



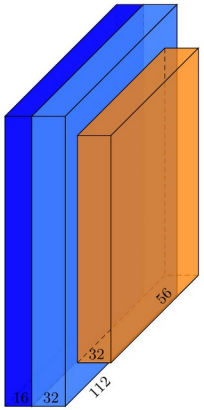
TinyYOLOV2 Architecture

Input Image
R, G, B Channels
224×224×3



conv1: [3,3,3]/1 × 16
224×224×16

max1: [2,2,1]/2 × 16
112×112×16



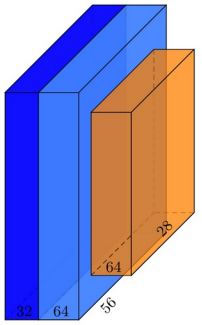
conv2: [3,3,1]/1 × 16
112×112×16

conv3: [1,1,16]/1 × 32
112×112×32

max2: [2,2,1]/2 × 32
56×56×32

Conv & Pool Key: kernelType [kw,hw,inchan]/stride × outchan
Layer Dimension Key: width × height × channels

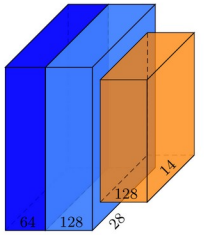
Purple: Full Convolution
Dark Blue: Depthwise Convolution
Light Blue: Pointwise Convolution
Orange: Max Pooling



conv4: [3,3,1]/1 × 32
56×56×32

conv5: [1,1,32]/1 × 64
56×56×64

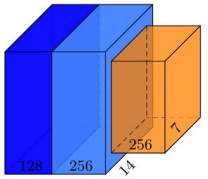
max3: [2,2,1]/2 × 64
28×28×64



conv6: [3,3,1]/1 × 64
28×28×64

conv7: [1,1,64]/1 × 128
28×28×128

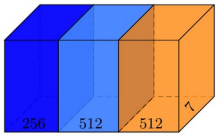
max4: [2,2,1]/2 × 128
14×14×128



conv8: [3,3,1]/1 × 128
14×14×128

conv9: [1,1,128]/1 × 256
14×14×256

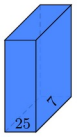
max5: [2,2,1]/2 × 256
7×7×256



conv10: [3,3,1]/1 × 256
7×7×256

conv11: [1,1,256]/1 × 512
7×7×512

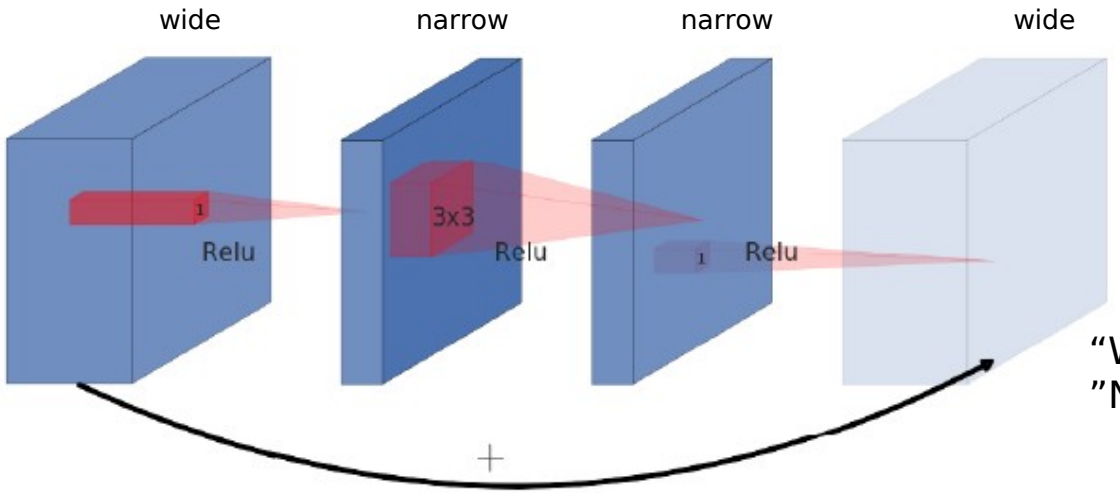
max6: [2,2,1]/1 × 512
7×7×512



conv12: [1,1,512]/1 × 25
7×7×25

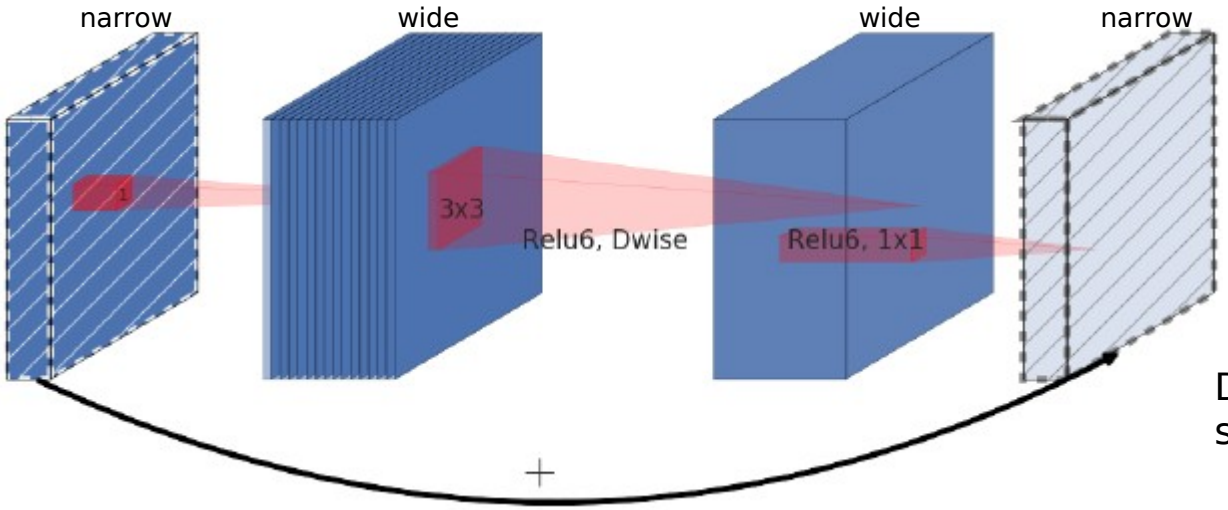
Layer depths are drawn with logarithmic scaling. Max pooling layer widths and heights are shrunk by $\frac{1}{4}$ instead of $\frac{1}{2}$.

Bottlenecks with Residuals



MobileNet: residual bottleneck

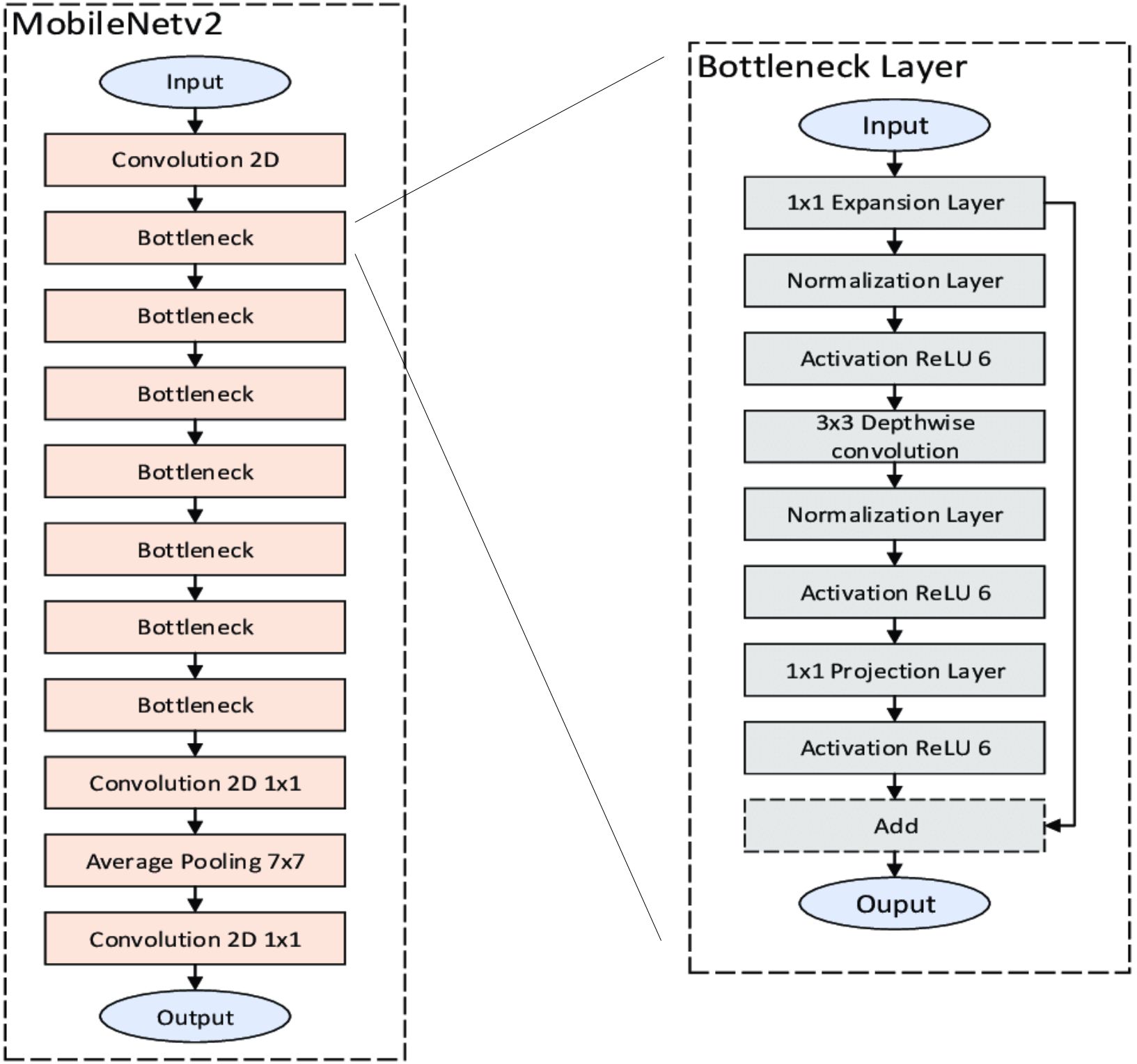
"Wide" layers have many channels.
"Narrow" layers have few channels.



MobileNetV2: inverted residual bottleneck

Depthwise convolution applies the same 3x3 kernel to all channels.

Images from https://towardsdatascience.com/mobilenetv2-inverted-residuals-and-linear-bottlenecks-8a4362f4ffd5


MobileNetv2
Input
Convolution 2D
Bottleneck
Bottleneck
Bottleneck
Bottleneck
Bottleneck
Bottleneck
Bottleneck
Bottleneck
Convolution 2D 1x1
Average Pooling 7x7
Convolution 2D 1x1
Output
Bottleneck Layer
Input
1x1 Expansion Layer
Normalization Layer
Activation ReLU 6
3x3 Depthwise convolution
Normalization Layer
Activation ReLU 6
1x1 Projection Layer
Activation ReLU 6
Add
Ouput

PyTorch Vision Models

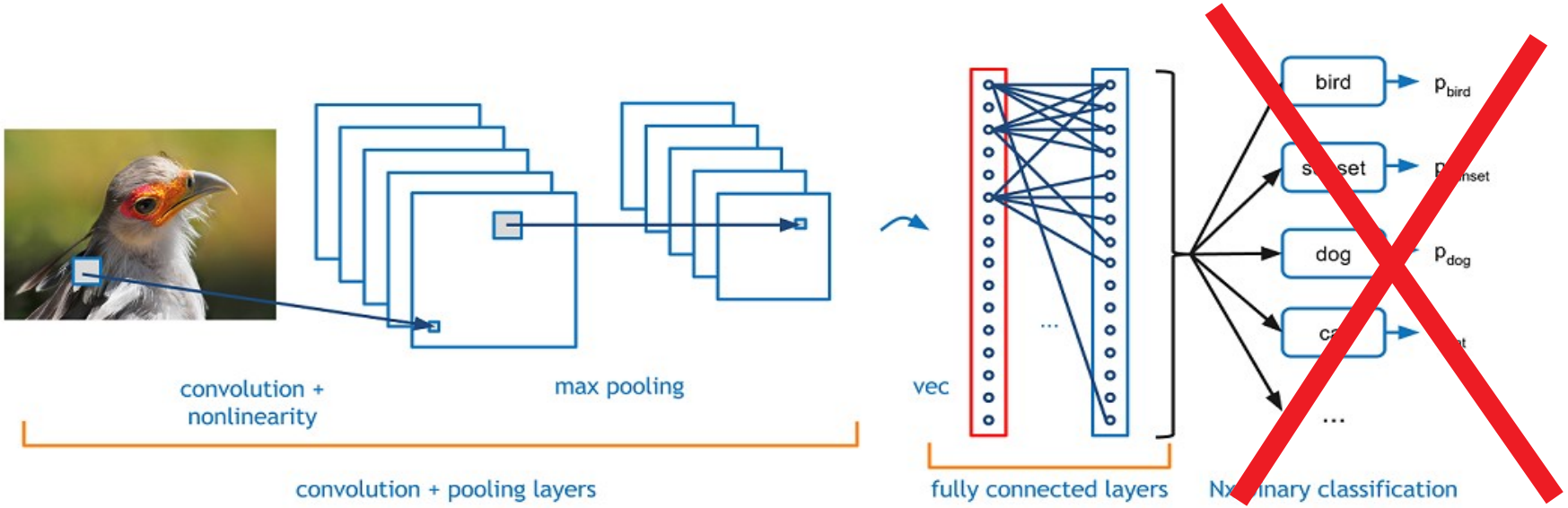
- PyTorch contains several pre-trained object recognition models, including AlexNet, ResNet, Inception, VGG, and MobileNetV2.
- Look in torchvision.models for a list.
- Models are trained on the ImageNet dataset.

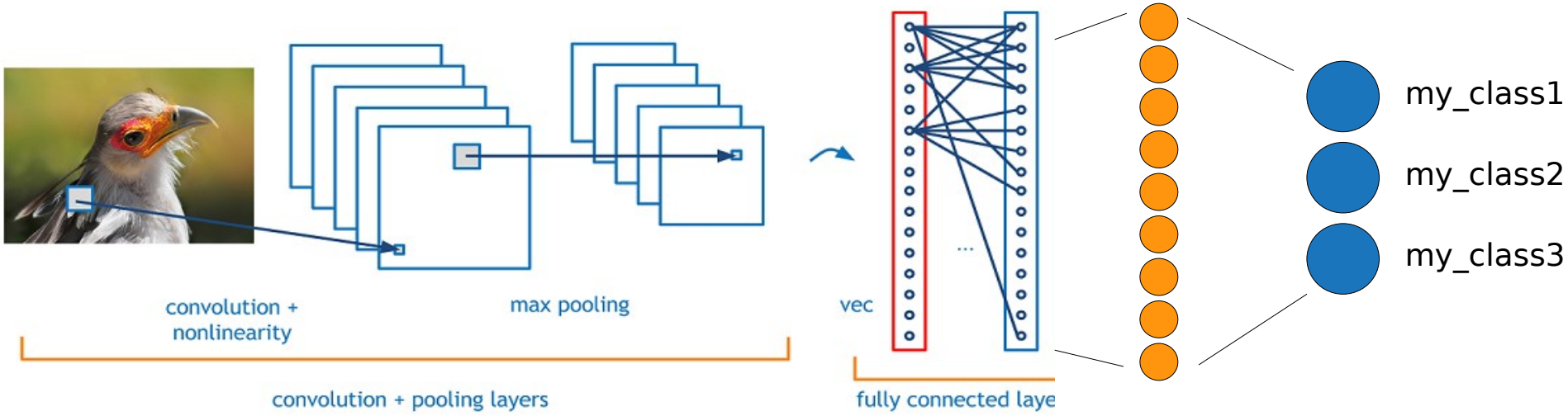
MobileNetV2 on VEX AIM

- See the course’s demos folder.
- Uses pre-trained MobileNetV2 model from torchvision.models.
- Feeds a 224x224 camera image into the network and reports the top 5 labels.

Transfer Learning

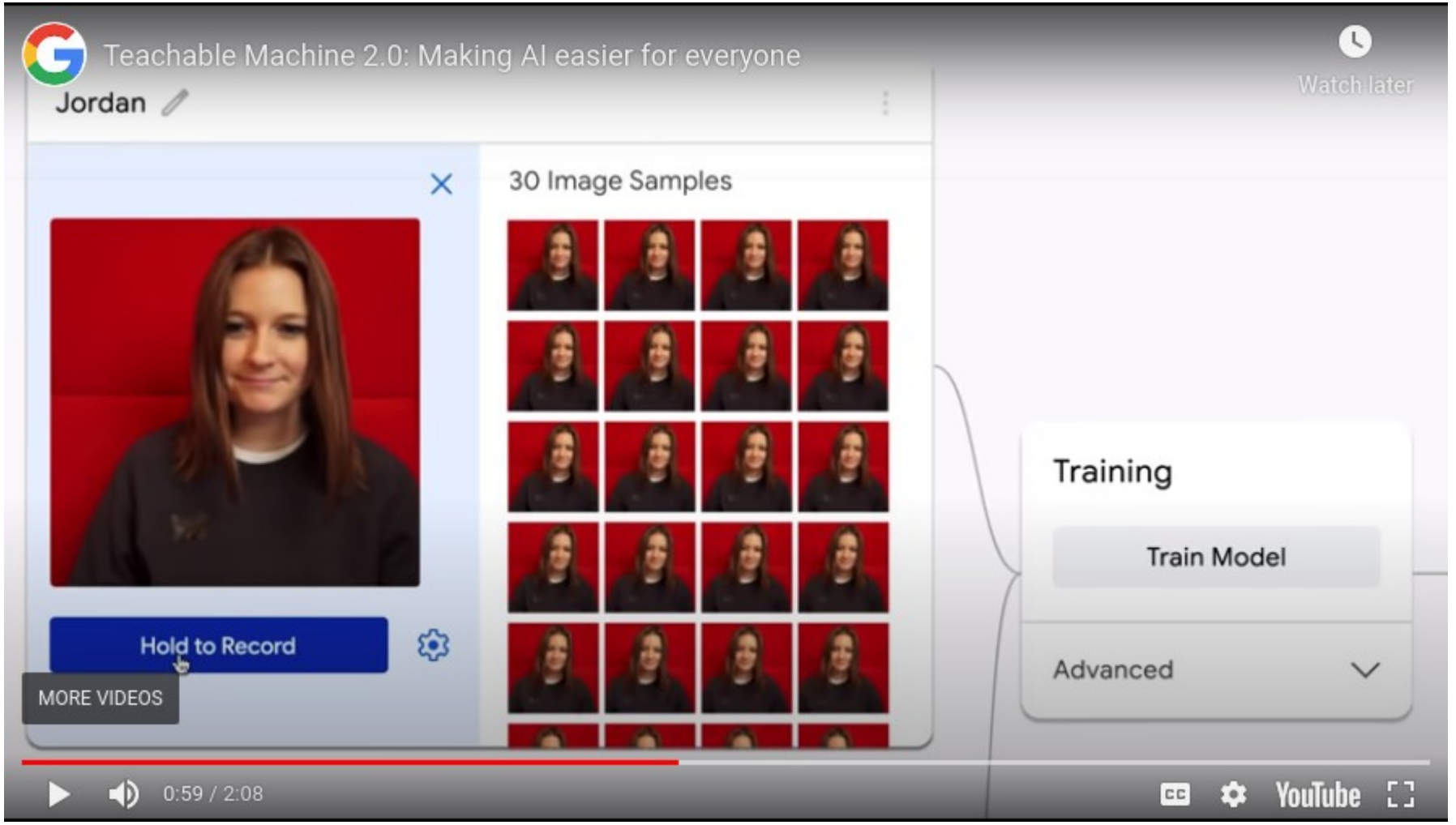
- How can we quickly train a visual classifier for a new object class?
- Use the last hidden layer of a pre-trained ImageNet classifier as a feature vector.
- Train a classifier on the new categories using just 1-2 layers of trainable weights, or just use k-nearest neighbor.
- This is how Teachable Machine works.


convolution +
nonlinearity
max pooling
convolution + pooling layers
vec
fully connected layers
bird
dog
...


convolution +
nonlinearity
max pooling
convolution + pooling layers
vec
fully connected laye
my_class1
my_class2
my_class3

Teachable Machine

https://teachablemachine.withgoogle.com



Object Recognition in GPT-4o

- You can send camera images to GPT-4o and ask what it sees in the image.

- This is advanced computer vision, beyond simple object recognition:
  - Describe multiple objects
  - Describe relationships between objects
  - Describe an overall scene, e.g., “a classroom scene”