Trustworthy AI Systems

-- Image Classification

Instructor: Guangjing Wang

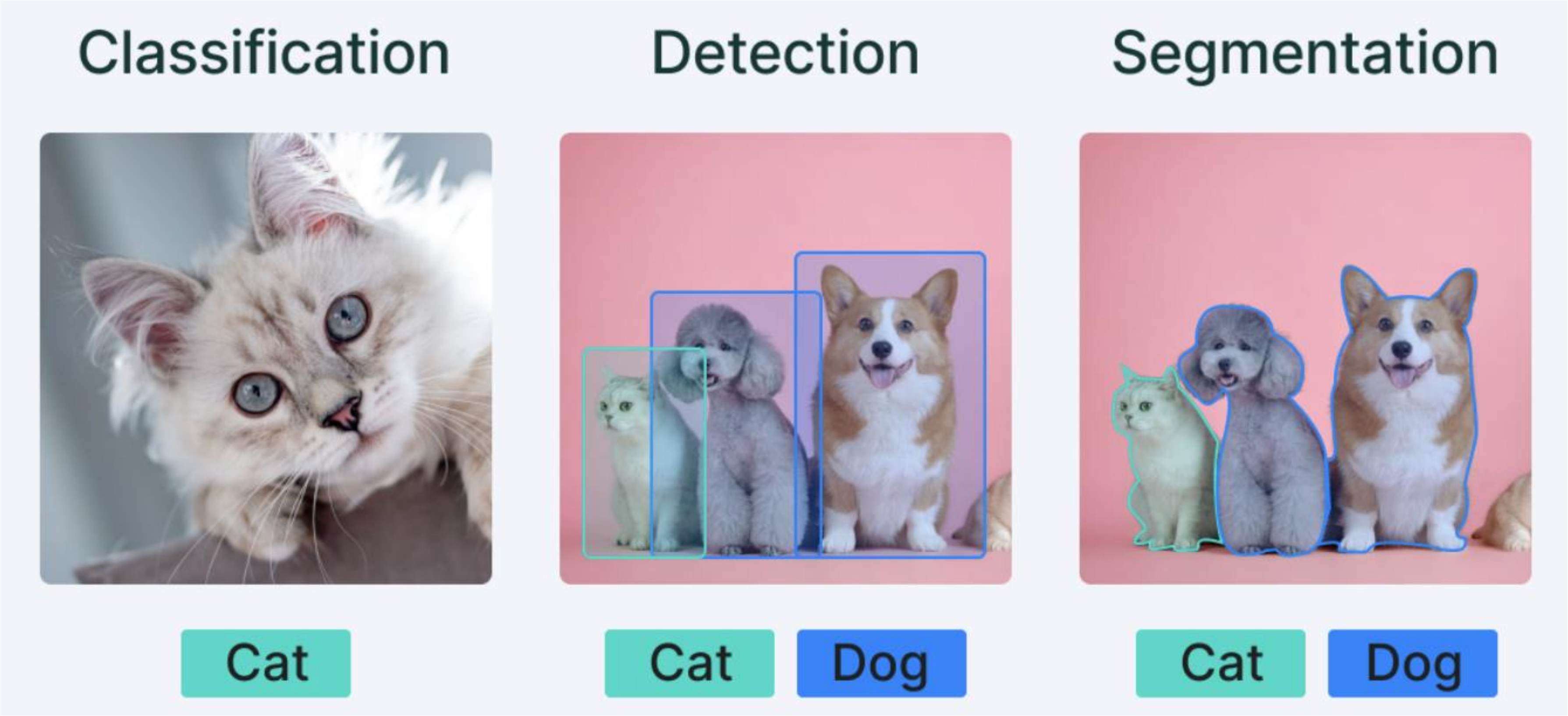
guangjingwang@usf.edu

Trustworthy AI Systems

- This course is NOT a traditional…, but comprehensive
  - Computer Vision
  - Natural Language Processing
  - Speech Recognition
  - Deep Learning
  - Machine Learning
  - Artificial Intelligence

- Last Lecture: An overview of Trustworthy AI Systems

Classical Computer Vision Tasks



Localizing and labeling objects

Dividing images into regions

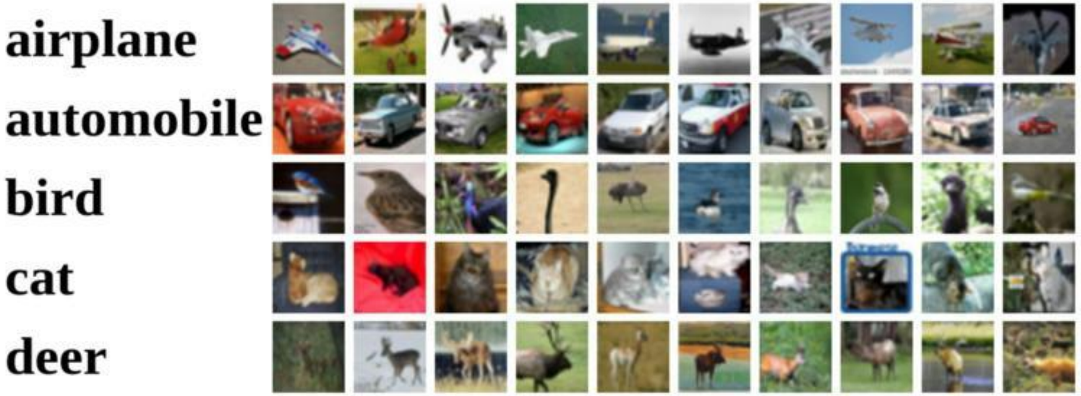
https://www.v7labs.com/blog/what-is-computer-vision

Data-driven Computer Vision

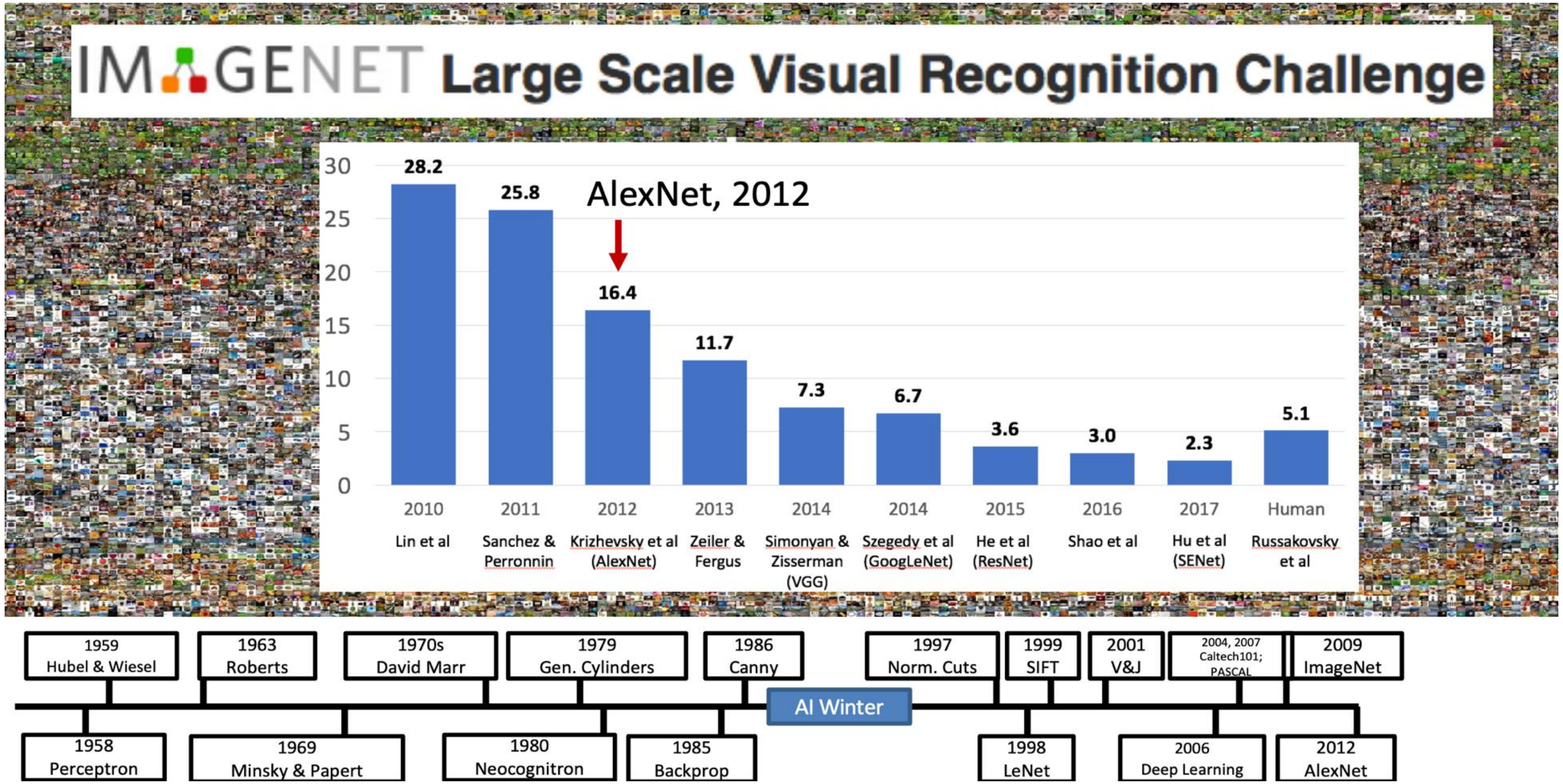
1. Collect a set of images and labels
2. Use deep learning algorithms to train a classifier or regression model
3. Evaluate the model on new images

```
def train(images, labels):
  # Machine learning!
  return model
```

```
def predict(model, test_images):
  # Use model to predict labels
  return test_labels
```



Data-driven Computer Vision

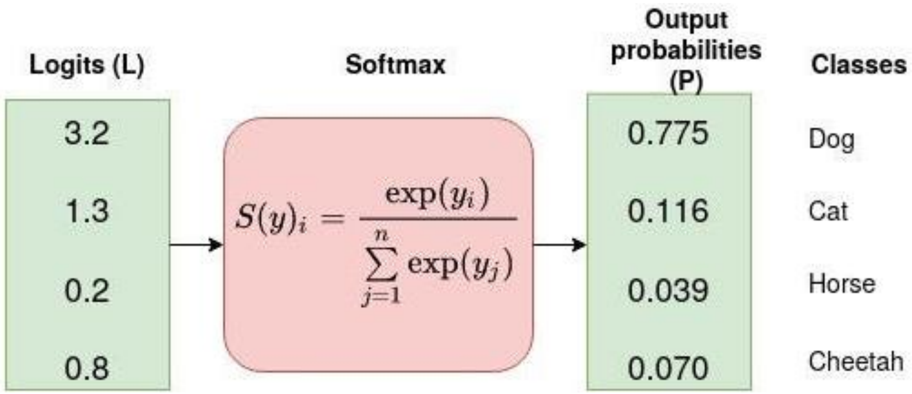


https://cs231n.stanford.edu/slides/2024/lecture_1_part_1.pdf

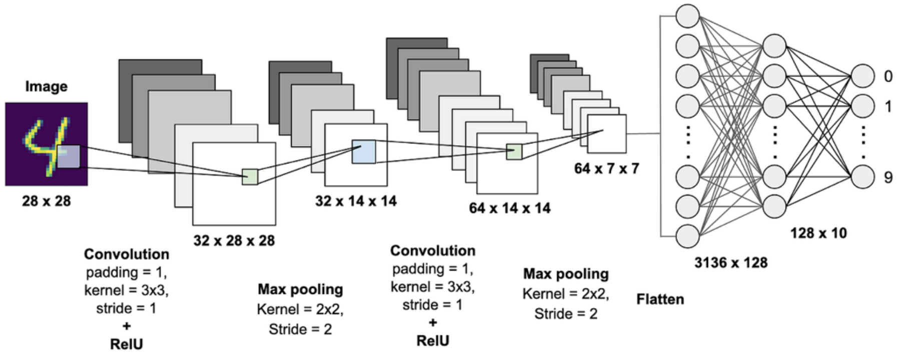
Deep Learning for Image Classification

https://stock.adobe.com/search?k=panda

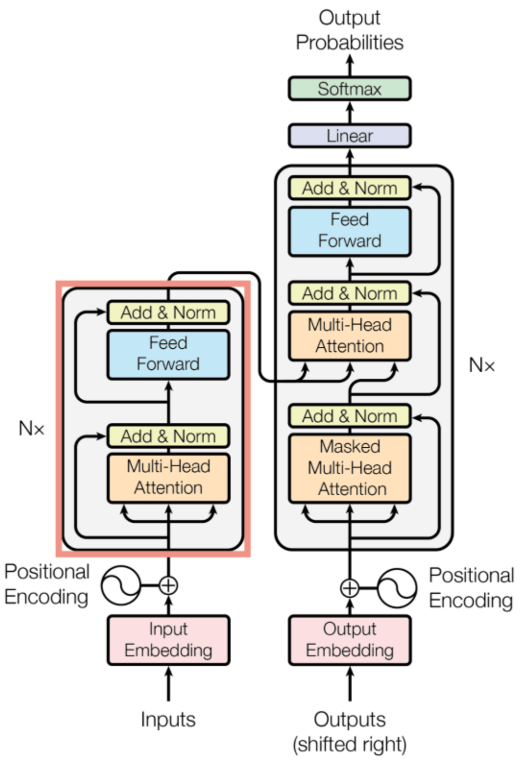
```python
import torch
import torch.nn as nn
class Model4_1(nn.Module):
    def __init__(self):
        super(Model4_1, self).__init__()
        self.lin1 = nn.Linear(784, 100)
        self.relu = nn.ReLU()
        self.lin2 = nn.Linear(100, 10)

    def forward(self, x):
        out = self.lin1(x)
        out = self.relu(out)
        out = self.lin2(out)
        return out

model4_1 = Model4_1()
```

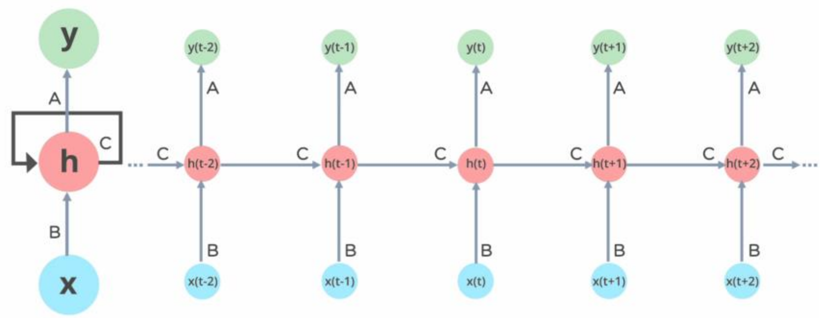


Deep Learning: A general term for various DNNs



https://becominghuman.ai/building-a-convolutional-neural-network-**cnn-model**-for-image-classification-116f77a7a236

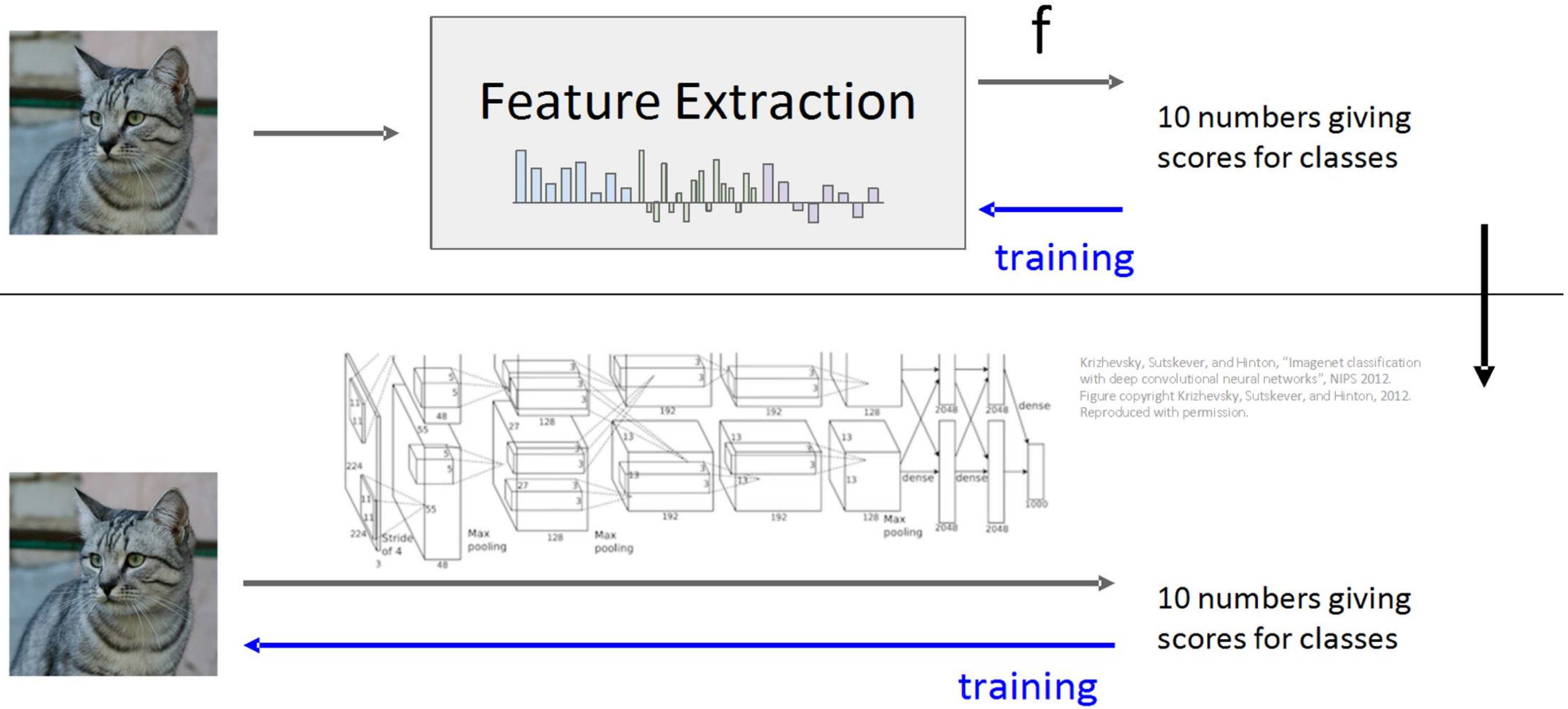




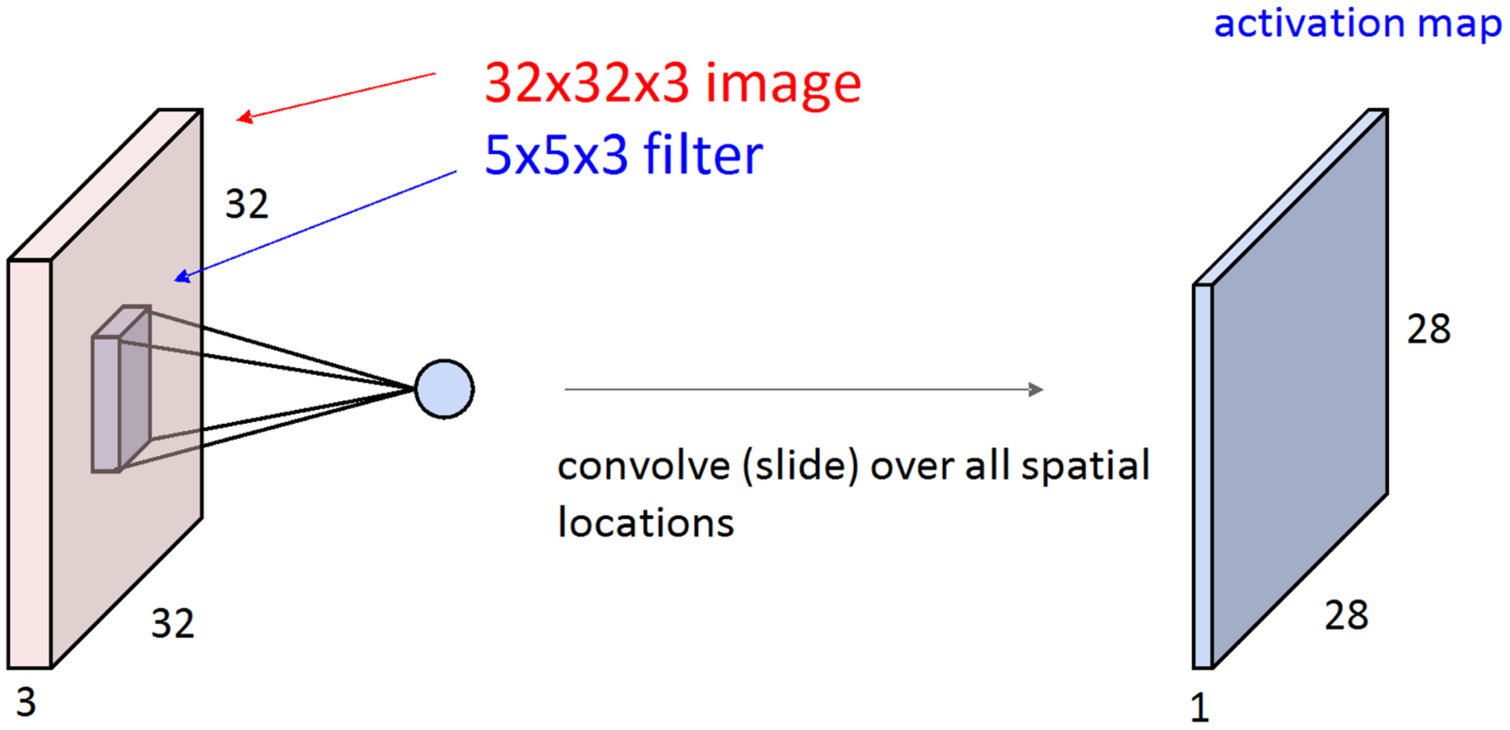
https://www.analyticsvidhya.com/blog/2022/03/a-brief-overview-of-**recurrent-neural-networks**-rnn/

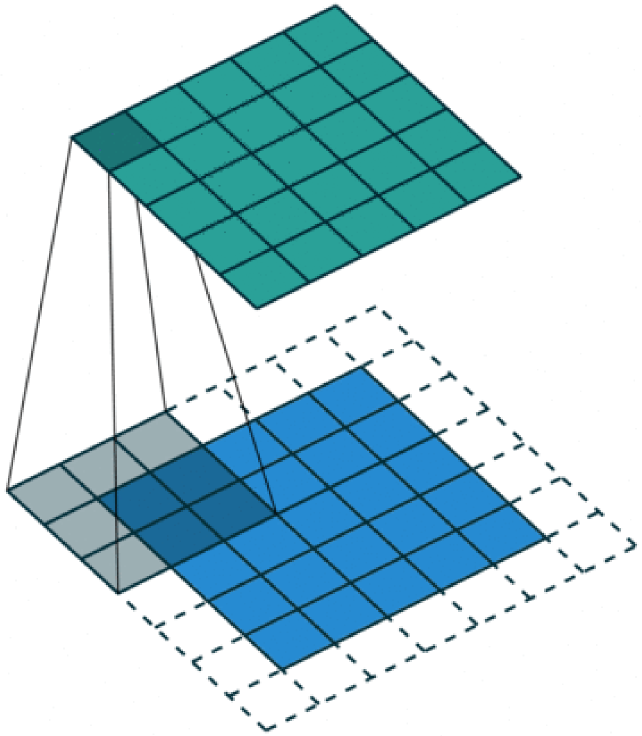
https://machinelearningmastery.com/the-**transformer**-model/

Why Deep Learning?



Convolutional Layer

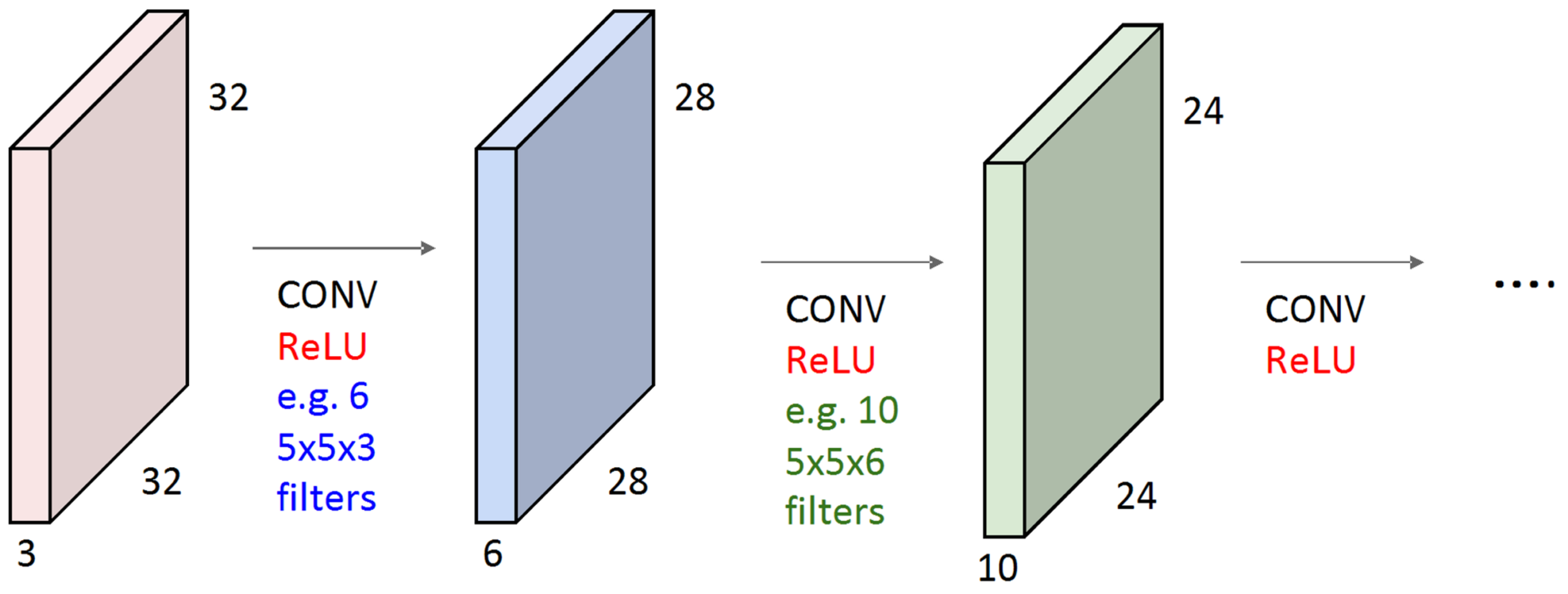




Filter: a small matrix of weights

https://hannibunny.github.io/mlbook/neuralnetworks/convolutionDemos.html

Convolutional Neural Network



Conv Layer in PyTorch

Conv2d

CLASS torch.nn.Conv2d(*in_channels*, *out_channels*, *kernel_size*, *stride=1*, *padding=0*, *dilation=1*, *groups=1*, *bias=True*, *padding_mode='zeros'*, *device=None*, *dtype=None*) [SOURCE]

Applies a 2D convolution over an input signal composed of several input planes.

In the simplest case, the output value of the layer with input size (N, C_{in}, H, W) and output $(N, C_{\text{out}}, H_{\text{out}}, W_{\text{out}})$ can be precisely described as:

$$\text{out}(N_i, C_{\text{out}_j}) = \text{bias}(C_{\text{out}_j}) + \sum_{k=0}^{C_{\text{in}}-1} \text{weight}(C_{\text{out}_j}, k) \star \text{input}(N_i, k)$$

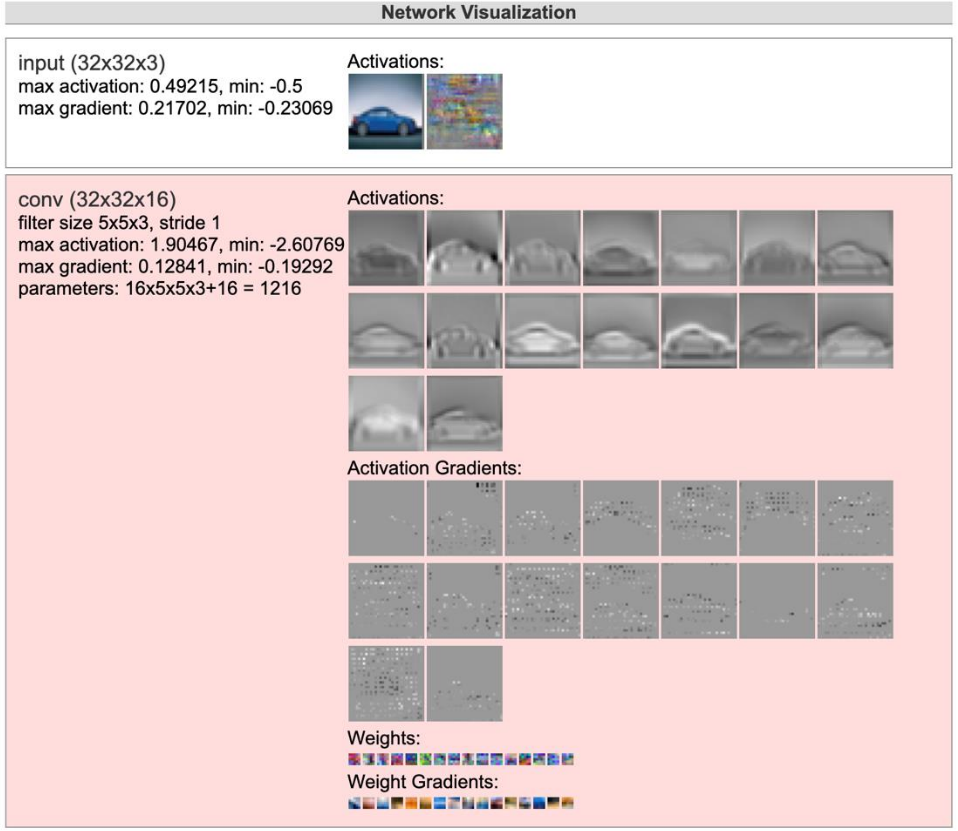
where $\star$ is the valid 2D cross-correlation operator, N is a batch size, C denotes a number of channels, H is a height of input planes in pixels, and W is width in pixels.

This module supports TensorFloat32.

On certain ROCm devices, when using float16 inputs this module will use different precision for backward.

- `stride` controls the stride for the cross-correlation, a single number or a tuple.
- `padding` controls the amount of padding applied to the input. It can be either a string {'valid', 'same'} or an int / a tuple of ints giving the amount of implicit padding applied on both sides.
- `dilation` controls the spacing between the kernel points; also known as the u00e0 trous algorithm. It is harder to describe, but this link has a nice visualization of what `dilation` does.

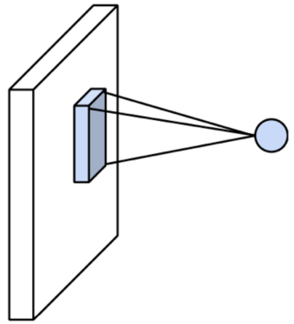
ConvNet JS Demo



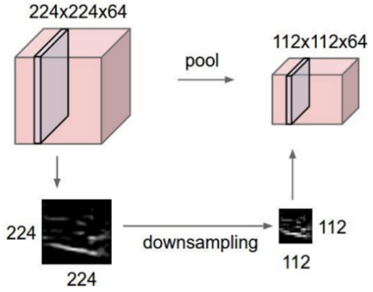
https://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html

Components of CNNs

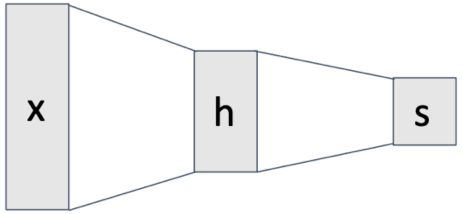
Convolution Layers



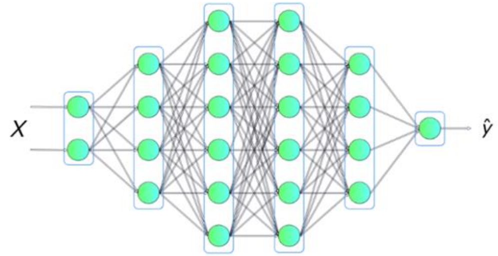
Pooling Layers



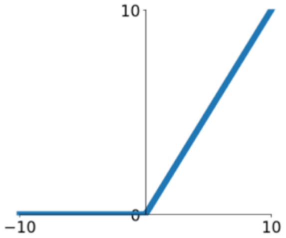
Fully-Connected Layers



DNN Example



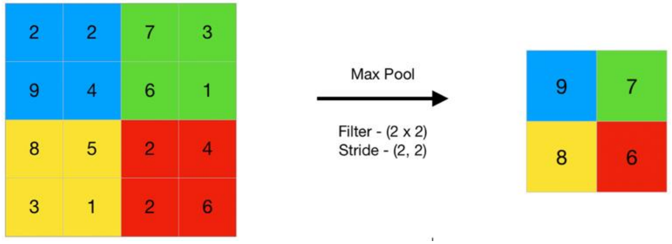
Activation Function



Normalization

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

Max Pooling Example



Batch Normalization (1)

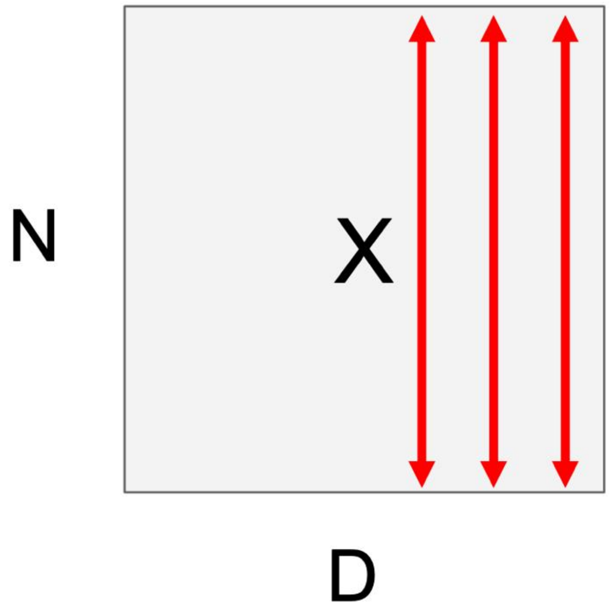
Consider a single layer $y = Wx$

The following could lead to tough optimization:
- Inputs x are not *centered around zero* (need large bias)
- Inputs x have different scaling per-element
  (entries in W will need to vary a lot)

Idea: force inputs to be “nicely scaled” at each layer!

Batch Normalization (2)

Input: $x : N \times D$



$$\mu_j = \frac{1}{N}\sum_{i=1}^{N} x_{i,j}$$ Per-channel mean, shape is D

$$\sigma_j^2 = \frac{1}{N}\sum_{i=1}^{N} (x_{i,j} - \mu_j)^2$$ Per-channel var, shape is D

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$ Normalized x, Shape is N x D

Problem: What if zero-mean, unit variance is too hard of a constraint?

Batch Normalization (3)

Input: $x : N \times D$

Learnable scale and shift parameters:

$\gamma, \beta : D$

Learning $\gamma = \sigma$, $\beta = \mu$ will recover the identity function!

$$\mu_j = \frac{1}{N}\sum_{i=1}^{N} x_{i,j}$$

Per-channel mean, shape is D

$$\sigma_j^2 = \frac{1}{N}\sum_{i=1}^{N} (x_{i,j} - \mu_j)^2$$

Per-channel var, shape is D

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

Normalized x, Shape is N x D

$$y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j$$

Output, Shape is N x D

Batch Normalization: Test Time

Input: $x : N \times D$

Learnable scale and shift parameters:

$\gamma, \beta : D$

During testing batchnorm becomes a linear operator! Can be fused with the previous fully-connected or conv layer

$\mu_j =$ (Running) average of values seen during training — Per-channel mean, shape is D

$\sigma_j^2 =$ (Running) average of values seen during training — Per-channel var, shape is D

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

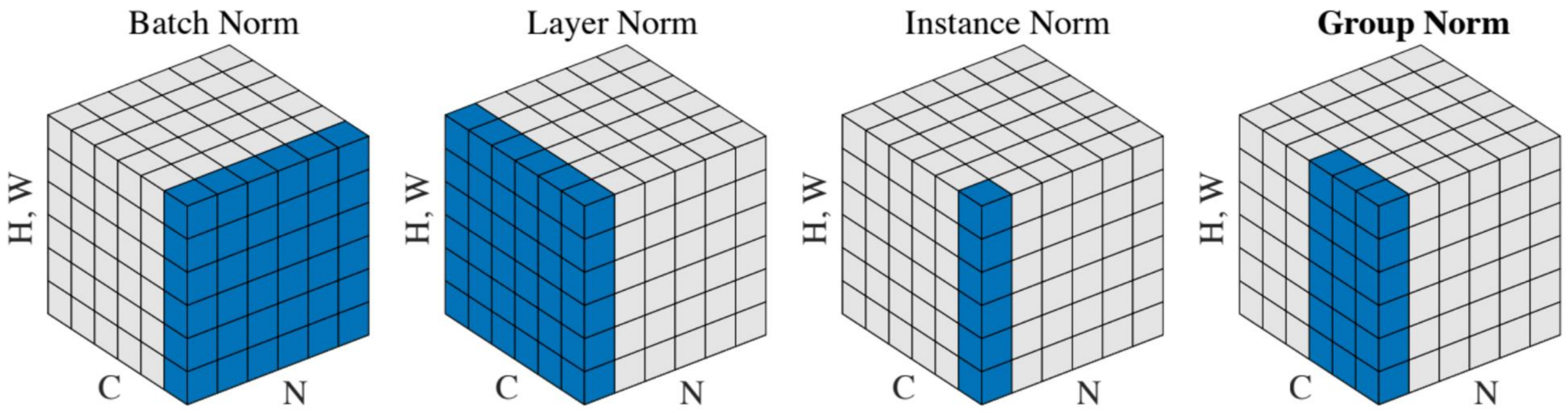
Normalized x, Shape is N x D

$$y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j$$

Output, Shape is N x D

Not homework… but read papers to learn

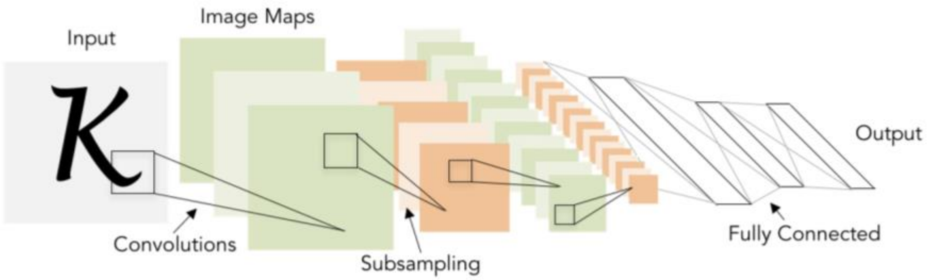
- Why using normalization?
- Other normalization techniques?
- N: batch size, C: channel size, H,W: height and width

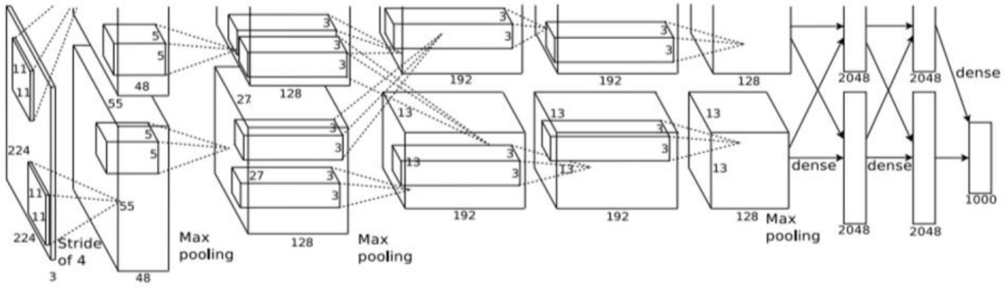


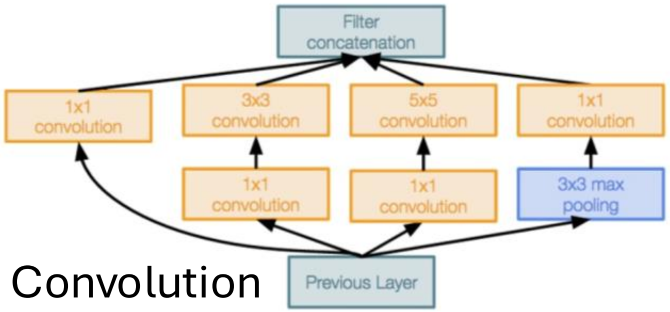
μ, σ: 1×C×1×1

μ, σ: N×C×1×1

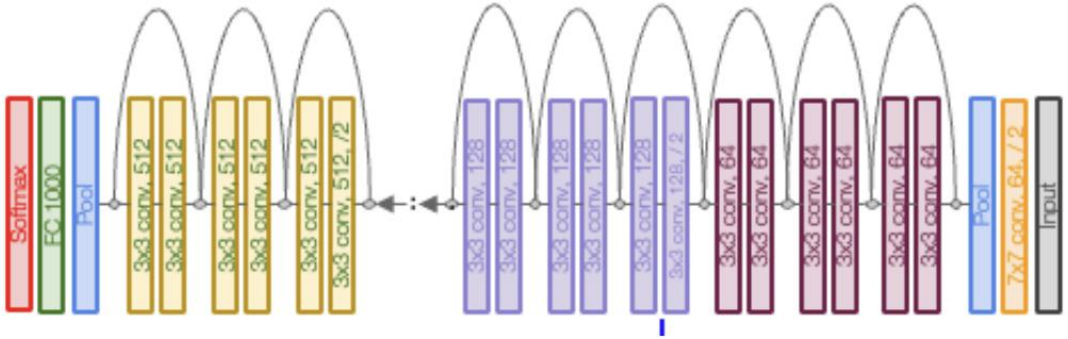
CNN Architectures







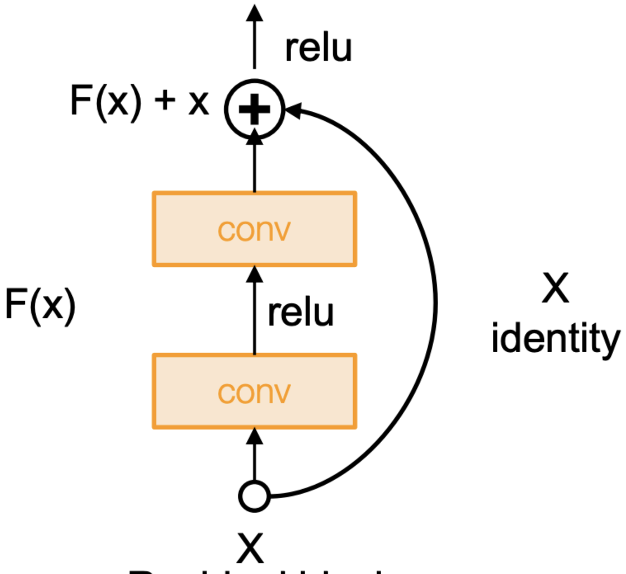
Pointwise Convolution



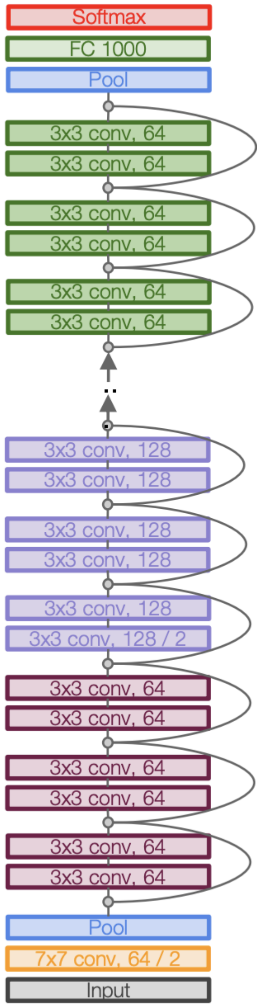
ResNet (1)

Very deep networks using residual connections:

- 152-layer model for ImageNet
- ILSVRC'15 classification winner (3.57% top 5 error) - Swept all classification and detection competitions in ILSVRC'15 and COCO'15!

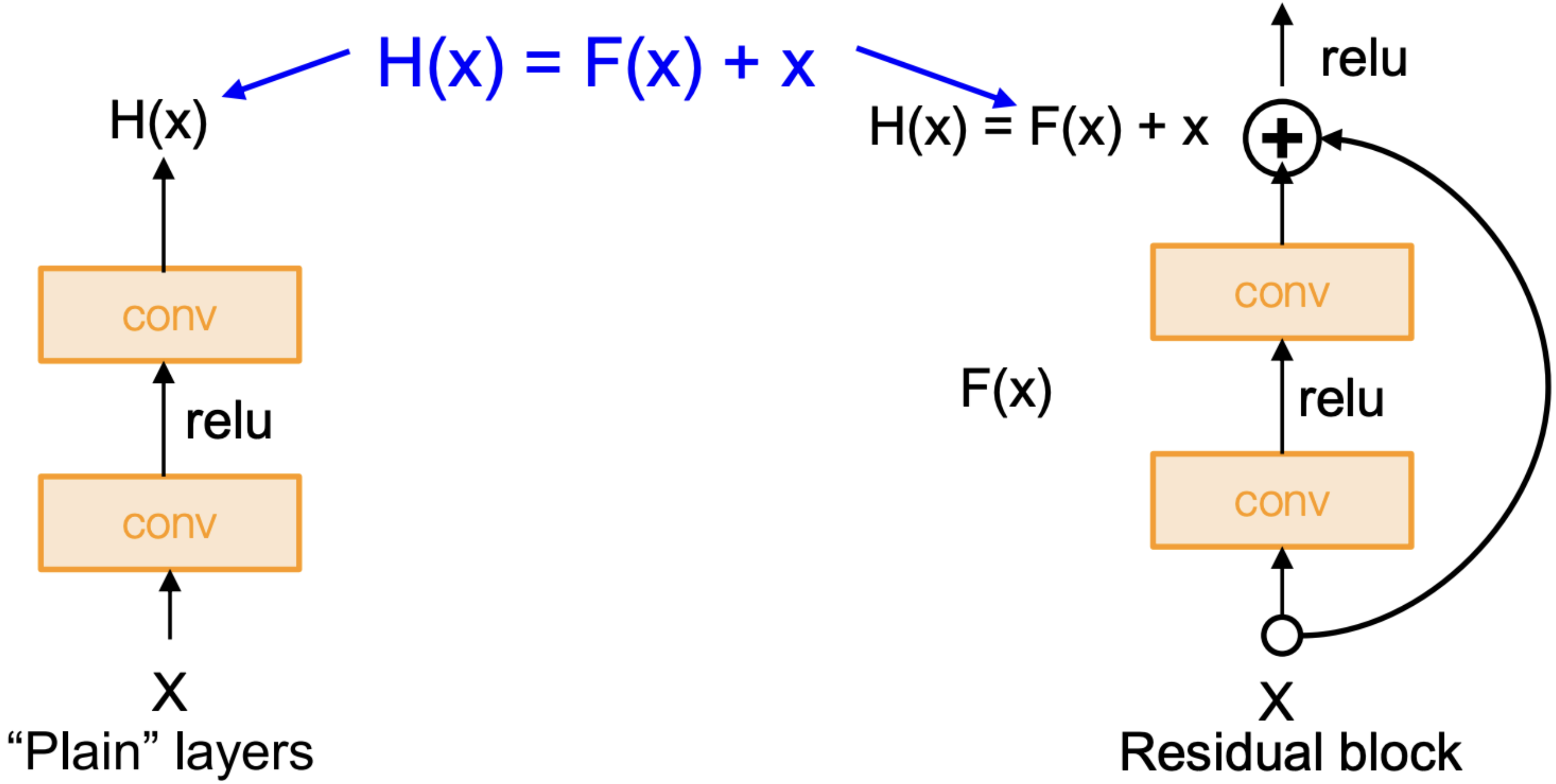


Residual block



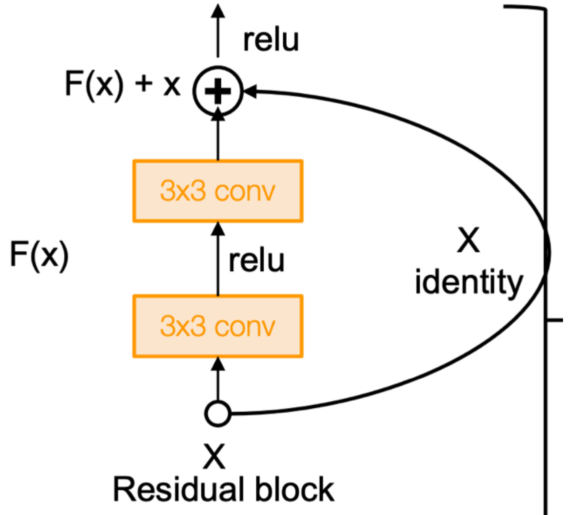
ResNet (2)

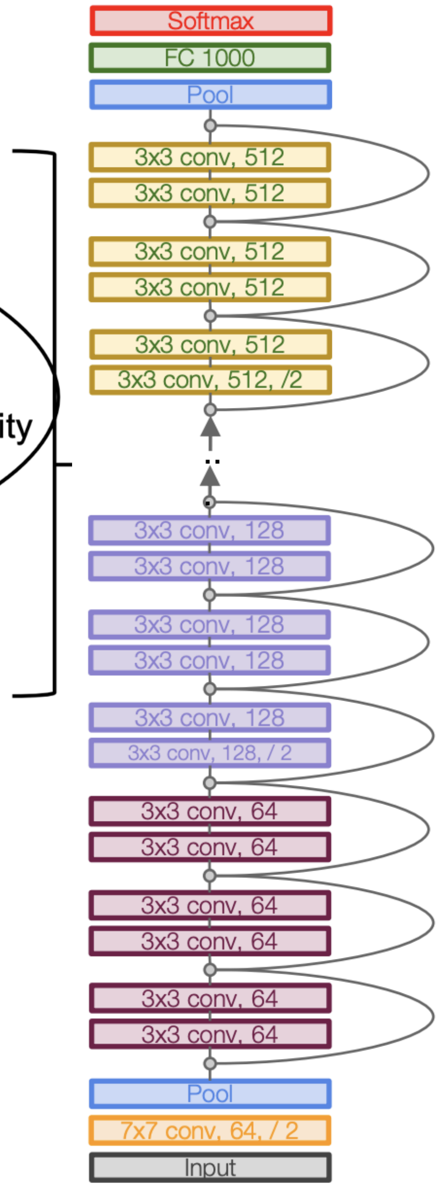
$H(x) = F(x) + x$



ResNet (3)

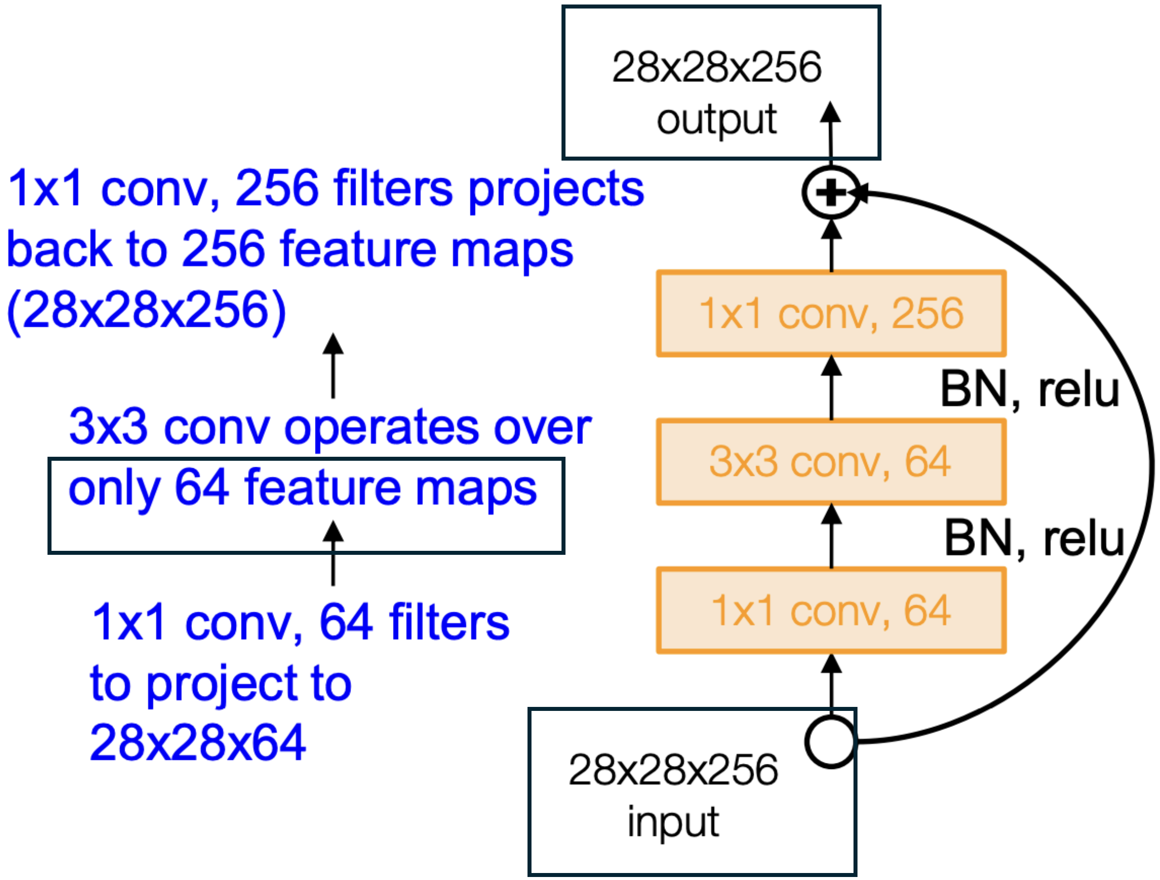
- Total depths of 18, 34, 50, 101, or 152 layers for ImageNet
- Stack residual blocks
- Every residual block has two 3x3 conv layers
- Periodically, double number of filters and down sample spatially using stride 2 (/2 in each dimension)
- Additional conv layer at the beginning
- No FC layers at the end (only FC 1000 to output classes)



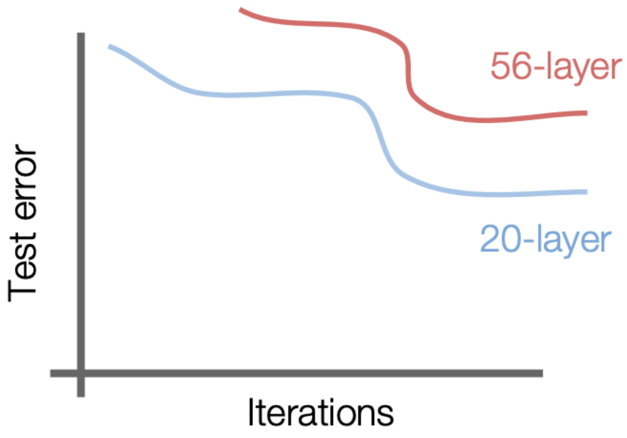


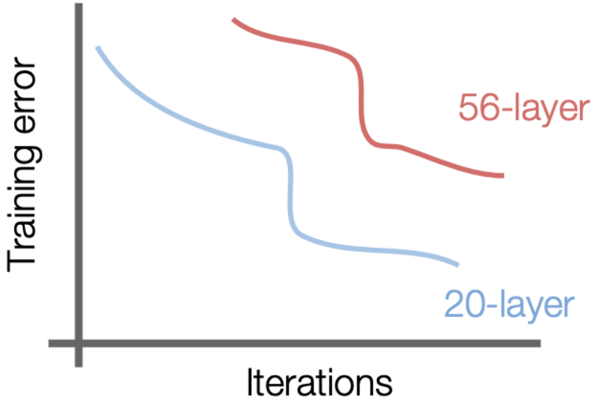
ResNet (4)

For deeper networks (ResNet-50+), use "bottleneck" layer to improve efficiency (similar to GoogLeNet)



Why ResNet





Problem: Deeper models are harder to optimize

Solution: Copying the learned layers from the shallower model and setting additional layers to identity mapping

ResNet Practice

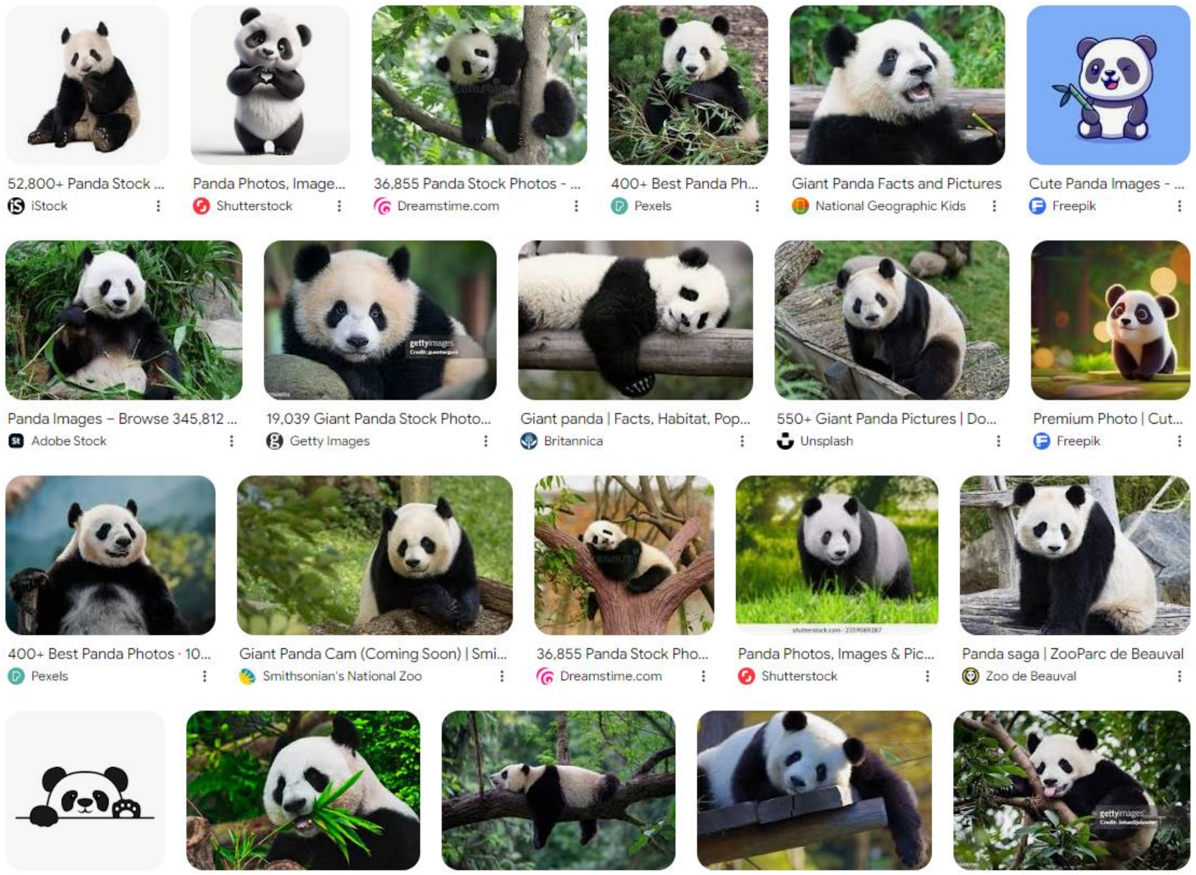
Training ResNet in practice:

- Batch Normalization after every CONV layer
- Xavier initialization from He et al.
- SGD + Momentum (0.9)
- Learning rate: 0.1, divided by 10 when validation error plateaus
- Mini-batch size 256
- Weight decay of 1e-5
- No dropout used

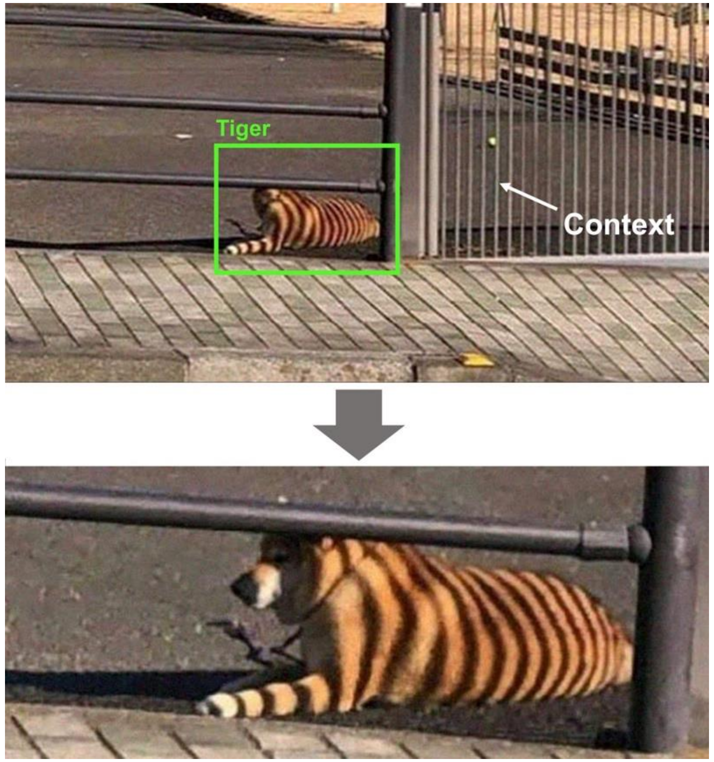
Back to Classification: Challenges

Variations in the physical world

- Illumination
- Background Clutter
- Occlusion
- Deformation
- Intraclass variation

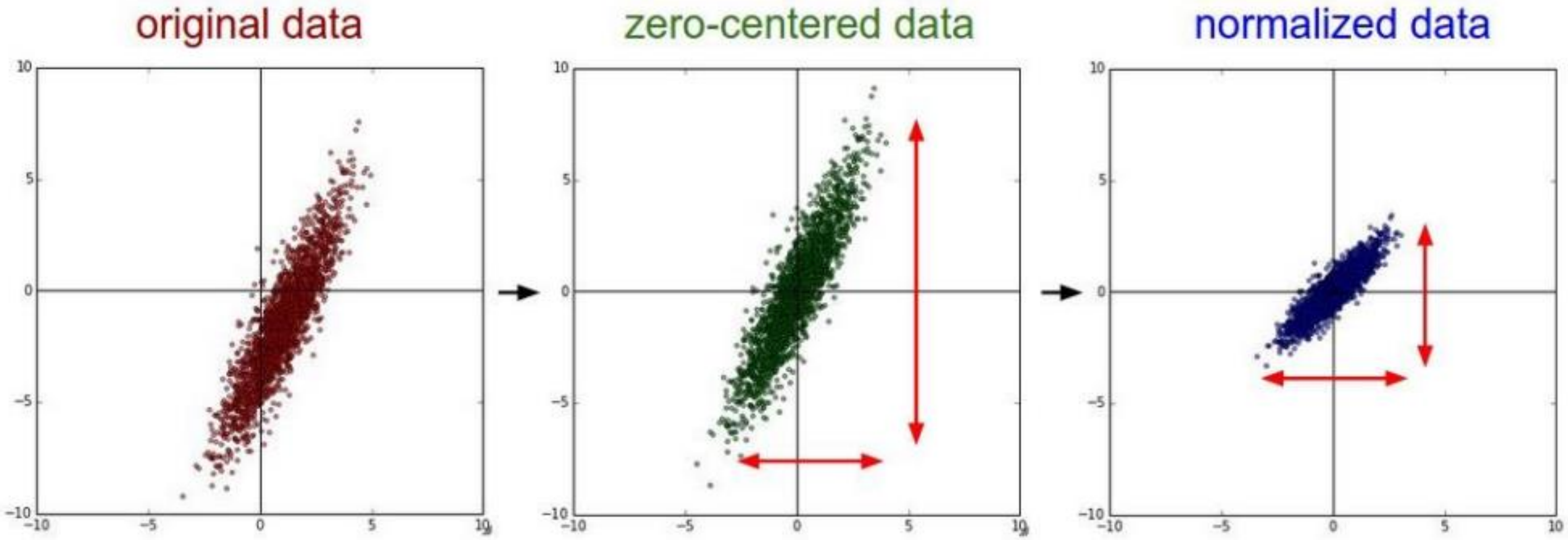


Challenges in Classification: Context



https://www.linkedin.com/posts/ralph-aboujaoude-diaz-40838313_technology-artificialintelligence-computervision-activity-6912446088364875776-h-lq/?utm_source=linkedin_share&utm_medium=member_desktop_web

For your project: Data Preprocessing




```
X -= np.mean(X, axis = 0)
```

```
X /= np.std(X, axis = 0)
```

For your project: Transfer Learning

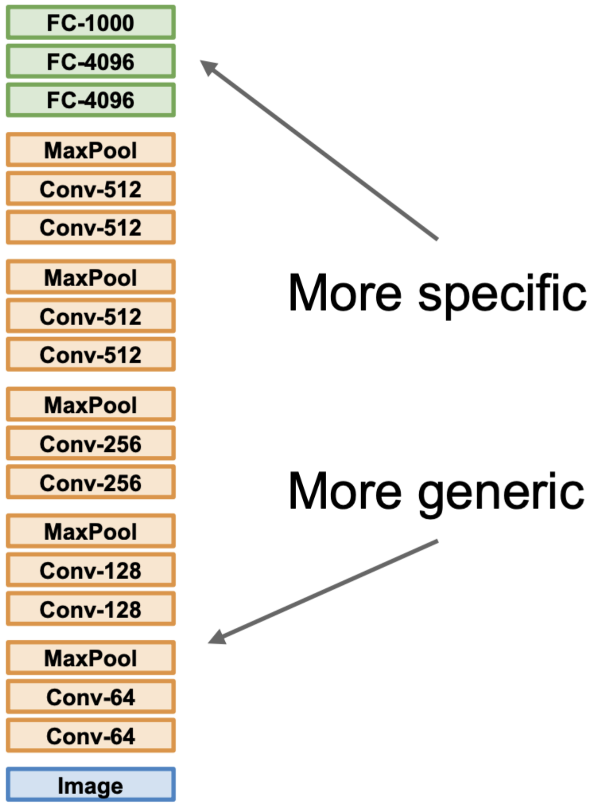
Have some dataset of interest but it has < ~1M images?

- Find a very large dataset that has similar data, train a big model there
- Transfer learn to your dataset

Deep learning frameworks provide a “Model Zoo” of pretrained models so you don’t need to train your own

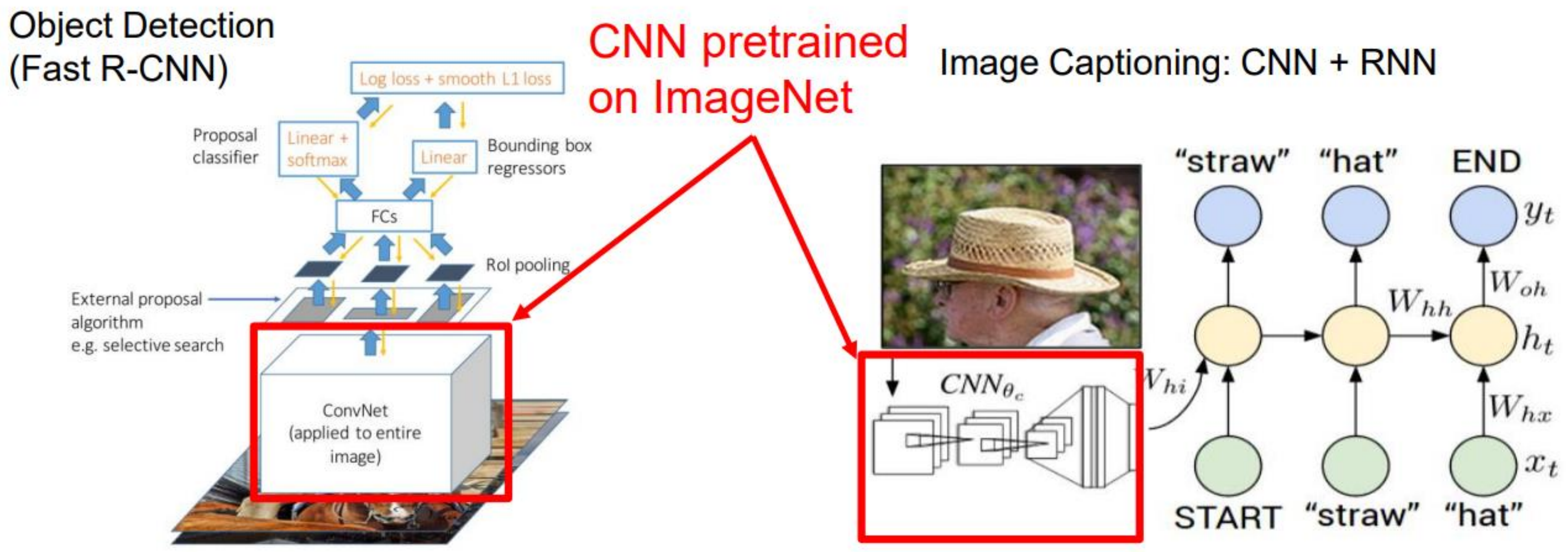
- https://github.com/tensorflow/models
- https://github.com/pytorch/vision

For your project: Transfer Learning



	very similar dataset	**very different dataset**
very little data	Use Linear Classifier on top layer	You're in trouble… Try linear classifier from different stages
quite a lot of data	Finetune a few layers	Finetune a larger number of layers or start from scratch!

For your project: Transfer Learning

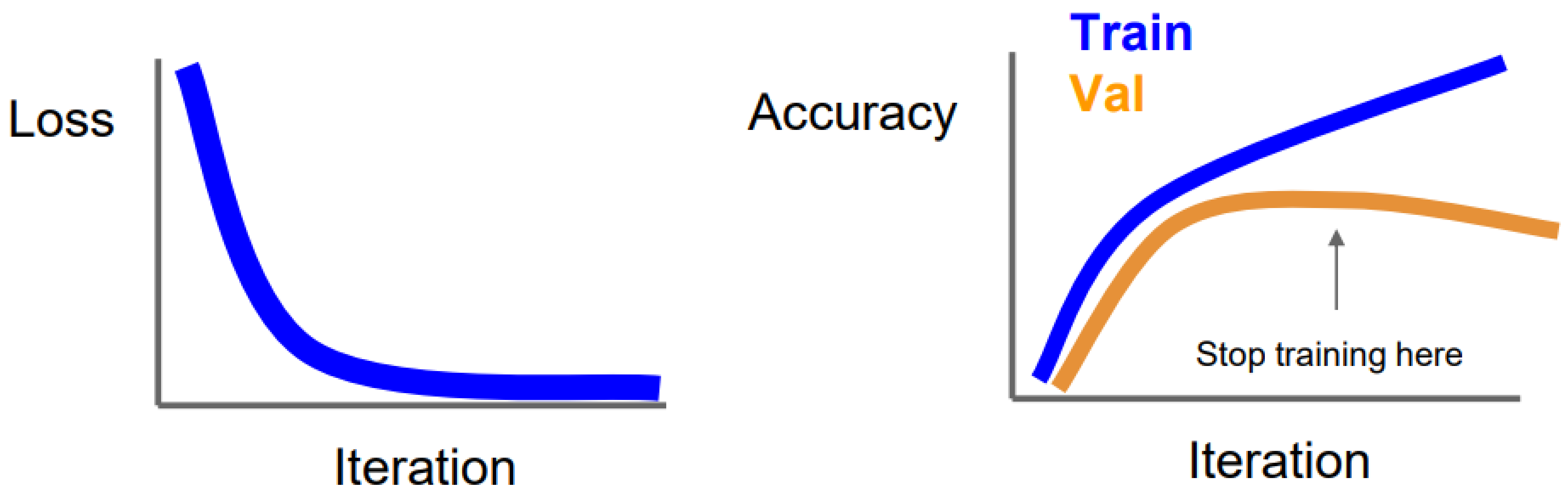


For your project: Some Practices

Consider CIFAR-10 example with [32,32,3] images:

- Data Preprocessing:
  - Subtract the mean image (e.g. AlexNet) (mean image = [32,32,3] array)
  - Subtract per-channel mean (e.g. VGGNet) (mean along each channel = 3 numbers)
  - Subtract per-channel mean and Divide by per-channel std (e.g. ResNet and beyond) (mean along each channel = 3 numbers)
- Weight Initialization: Kaiming / MSRA Initialization
- Use ReLU. Be careful with your learning rates
- Try out Leaky ReLU / PReLU / GELU (Check them out by yourself)

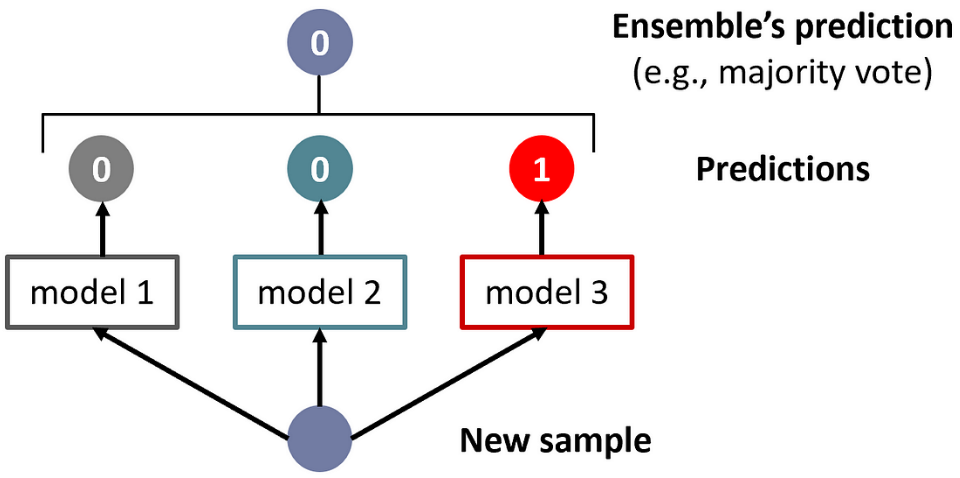
For your project: Early Stopping



Stop training the model when accuracy on the validation set decreases. Train for a long time, but always keep track of the model snapshot that worked best on val.

For your project: Model Ensembles

- Train multiple independent models
- At test time average their results



https://pub.towardsai.net/introduction-to-ensemble-methods-226a5a421687

For your project: Regularization (1)

- Add a term to a loss:

$$L = \frac{1}{N} \sum_{i=1}^{N} \sum_{j \neq y_i} \max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + 1) + \boxed{\lambda R(W)}$$

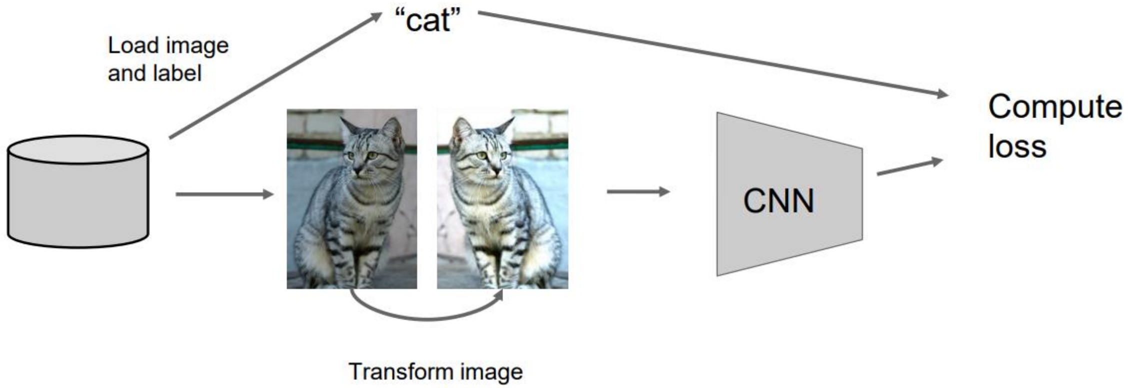
In common use:

L2 regularization	$R(W) = \sum_k \sum_l W_{k,l}^2$	(Weight decay)
L1 regularization	$R(W) = \sum_k \sum_l \lvert W_{k,l} \rvert$	
Elastic net (L1 + L2)	$R(W) = \sum_k \sum_l \beta W_{k,l}^2 + \lvert W_{k,l} \rvert$	

- Random Dropout, 0.5 is common

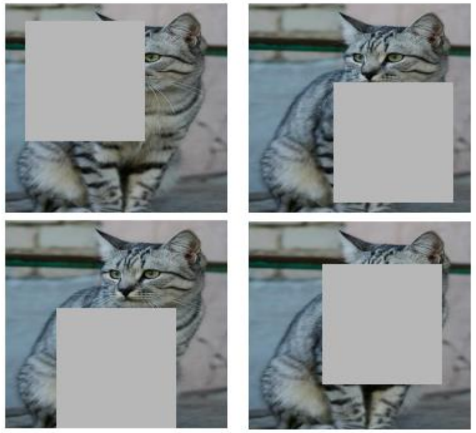
For your project: Regularization (2)

- Data Augmentation
  - Horizontal Flips
  - Random crops and scales
  - Color Jitter
  - Rotation
  - Shearing
  - ....



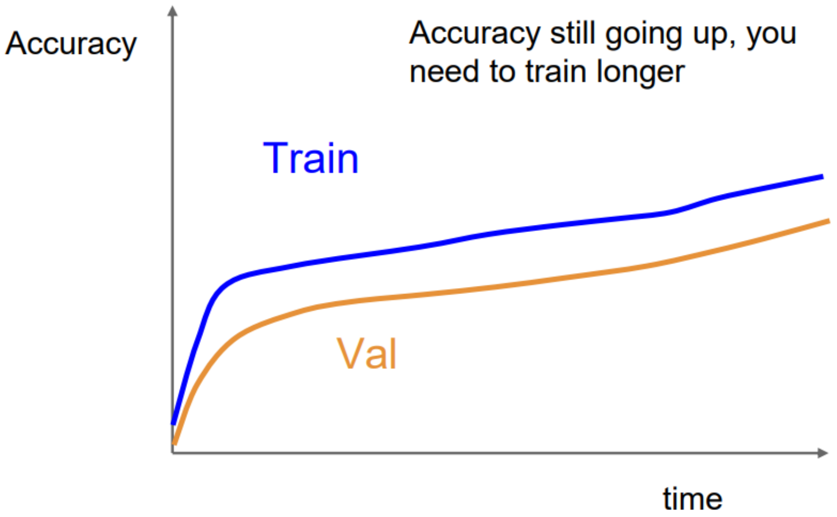
For your project: Regularization (3)

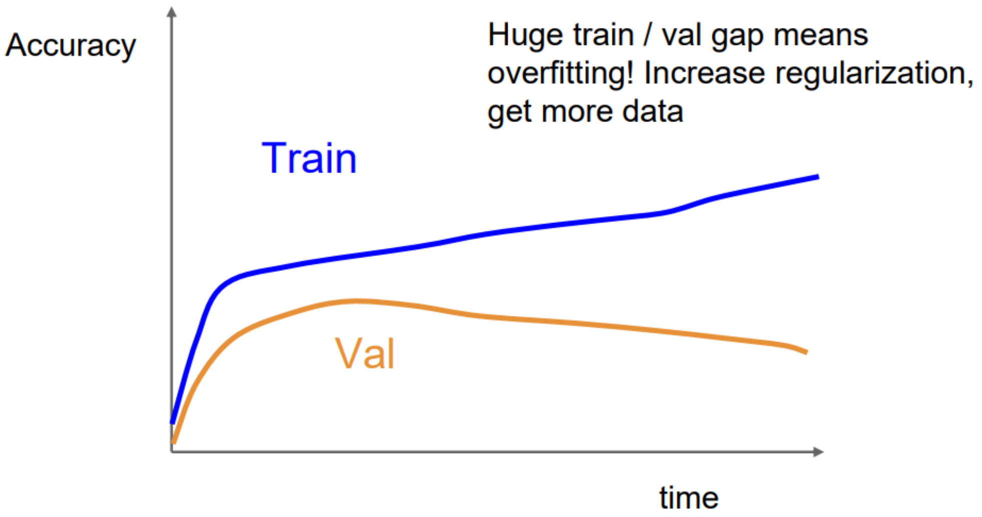
- Training: the core is to add random noise
  - Dropout: Consider dropout for large fully connected layers
  - Batch Normalization
  - Data Augmentation
  - Cutout / Random Crop :Try cutout especially for small classification datasets

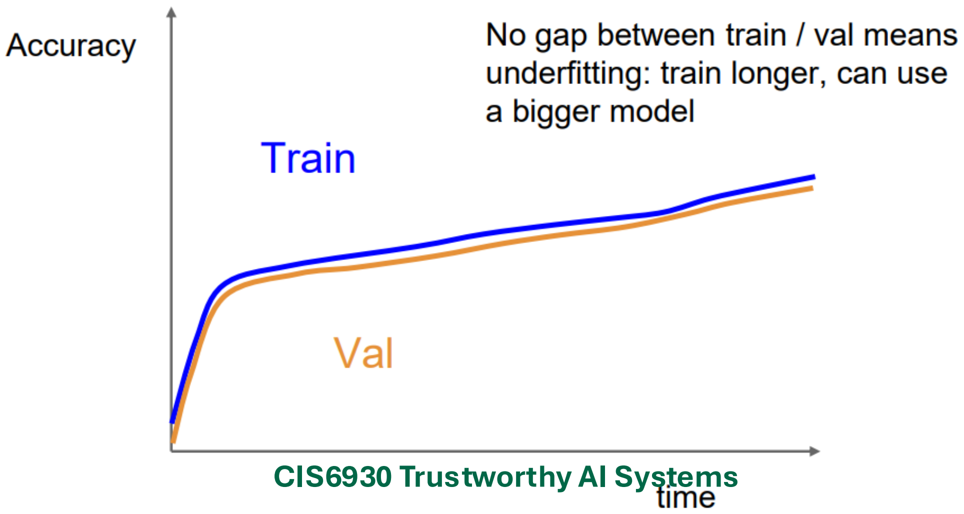


- Testing: Marginalize over the noise

For your project: Look at the Learning Curve







Reference: Stanford Spring 2024 cs231n

- https://cs231n.stanford.edu/schedule.html
- https://cs231n.stanford.edu/slides/2024/lecture_5.pdf
- https://cs231n.stanford.edu/slides/2024/lecture_6_part_1.pdf
- https://cs231n.stanford.edu/slides/2024/lecture_6_part_2.pdf