Linear Algebra Preparation for Emergent Neural Network Architectures (LAPENNA)

Unit 9

Network Model, Computing, Optimization, Applications, More…

Kwai Wong, Stan Tomov,
Julian Halloy, Stephen Qiu, Eric Zhao

March 31, 2022

University of Tennessee, Knoxville

Acknowledgements:

- Support from NSF, UTK, JICS, ICL, NICS
- LAPENNA, www.jics.utk.edu/lapenna, NSF award #202409
- www.icl.utk.edu, cfdlab.utk.edu, www.xsede.org, www.jics.utk.edu/recsem-reu,
- MagmaDNN is a project grown out from the RECSEM REU Summer program supported under NSF award #1659502
- Source code: www.bitbucket.org/icl/magmadnn
- www.bitbucket.org/cfdl/opendnnwheel

INNOVATIVE COMPUTING LABORATORY

Overview of LAPENNA

The major goal of The Linear Algebra Preparation for Emergent Neural Network Architectures (LAPENNA) program is to provide essential knowledge to advance literacy in AI to sustain the growth and development of the workforce in the cyberinfrastructure (CI) ecosystem. This program aims to prepare college faculty, researchers, and industrial practitioners to design, enable and direct their own course curricula, collaborative projects, and training programs for in-house data-driven sciences programs. The LAPENNA program focuses on delivering computational techniques, numerical algorithms and libraries, and implementation of AI software on emergent CPU and GPU platforms.

Ecosystem Linear Algebra Preparation for Emergent Neural Network Architectures (LAPENNA)

Modeling, Numerical Linear Algebra, Data Analytics, Machine Learning, DNN, GPU, HPC

Session 1	Session 2	Session 3	Session 4
7/2020 - 12/2020	1/2021- 6/2021	7/2021 - 1/2022	1/2022 - 6/2022
16 participants	16 participants	16 participants	16 participants
Faculty/Students	Faculty/Students	Faculty/Students	Faculty/Students
10 webinars	10 webinars	10 webinars	10 webinars
4 Q & A	4 Q & A	4 Q & A	4 Q & A

Colleges courses, continuous integration, online courses, projects, software support

Web-based resources, tutorials, webinars, training, outreach

Teams and Participants of LAPENNA

- ✓ **PIs : Kwai Wong (JICS), Stan Tomov (ICL), University of Tennessee, Knoxville**
  - **Stephen Qiu, Julian Halloy, Eric Zhao (Students)**

- ✓ **Team : Clemson University**
- ✓ **Teams : University of Arkansas**
- ✓ **Team : University of Houston, Clear Lake**
- ✓ **Team : Miami University, Ohio**
- ✓ **Team : Boston University**
- ✓ **Team : West Virginia University**
- ✓ **Team : Louisiana State University, Alexandria**
- ✓ **Teams : Jackson Laboratory**
- ✓ **Team : Georgia State University**
- ✓ **Teams : University of Tennessee, Knoxville**
- ✓ **Teams : Morehouse College, Atlanta**
- ✓ **Team : North Carolina A & T University**
- ✓ **Team : Clark Atlanta University, Atlanta**
- ✓ **Team : Alabama A & M University**
- ✓ **Team : Slippery Rock University**
- ✓ **Team : University of Maryland, Baltimore County**

Meeting Time

- ✓ **Webinar Meeting time. Thursday 8:00 – 10:00 pm ET,**
- ✓ **Tentative schedule, www.jics.utk.edu/lapenna --> Spring 2022**

Topic: LAPENNA Spring 2022 Webinar
Time: Feb 3, 2022 08:00 PM Eastern Time (US and Canada)
Every week on Thu, 12 occurrence(s)
Feb 3, 2022 07:30 PM
Feb 10, 2022 07:30 PM
Feb 17, 2022 07:30 PM
Feb 24, 2022 07:30 PM
Mar 3, 2022 07:30 PM
Mar 10, 2022 07:30 PM
Mar 17, 2022 07:30 PM
Mar 24, 2022 07:30 PM
Mar 31, 2022 07:30 PM
Apr 7, 2022 07:30 PM
Apr 14, 2022 07:30 PM
Apr 21, 2022 07:30 PM

Join from PC, Mac, Linux, iOS or Android: https://tennessee.zoom.us/j/94140469394
Password: 708069

Schedule of LAPENNA Spring 2022
Thursday 8:00pm -10:00pm Eastern Time

The goal of The Linear Algebra Preparation for Emergent Neural Network Architectures (LAPENNA) program is to provide essential knowledge to advance literacy in AI to sustain the growth and development of the workforce in the cyberinfrastructure (CI) ecosystem for data-driven applications.

Month	Week	Date	Topics
February	Week 01	3	Logistics, High Performance Computing
	Week 02	10	Computational Ecosystem, Linear Algebra
	Week 03	17	Introduction to DNN, Forward Path (MLP)
	Week 04	24	Backward Path (MLP), Math, Example
March	Week 05	3	Backpro (MLP), CNN Computation
	Week 06	10	CNN Backpropagation, Example
	Week 07	17	CNN Network, Linear Algebra
	Week 08	24	Segmentation, Unet,
	Week 09	31	Object Detection, RC vehicle
April	Week 10	7	RNN, LSTEM, Transformers
	Week 11	14	DNN Computing on GPU
	Week 12	21	Overview, Closing
June or July	Workshop	To be arranged	4 Days In Person at UTK

Topics of Unit 11

- ✓ **UNIT 11 : DNN Computing**
  - **DNN Principles: Mathematics**
  - **CNN Model: ResNet**
  - **DNN Computing: Essential Practices**
  - **DNN Usage: I/O, Transfer Learning**
  - **CNN Applications: Unet, Object Detection, Autonomous Vehicle**
  - **Advanced Topics: GAN, Reinforcement Learning**

DNN Model

Applications

+

Data Ensemble + Input

+

It's all about

linear algebra calculations

DNN in particular

+

Algorithms, Software

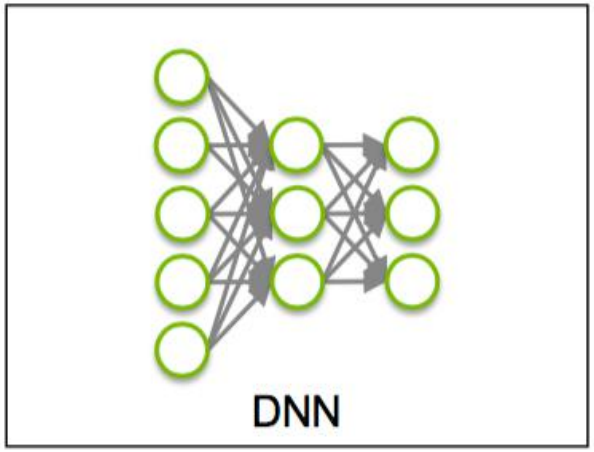
+

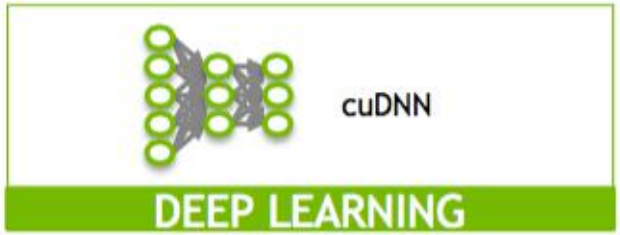
Output + Data Analysis

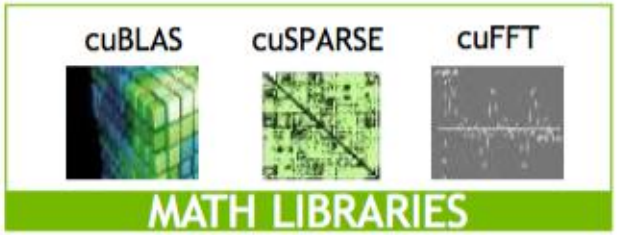
+

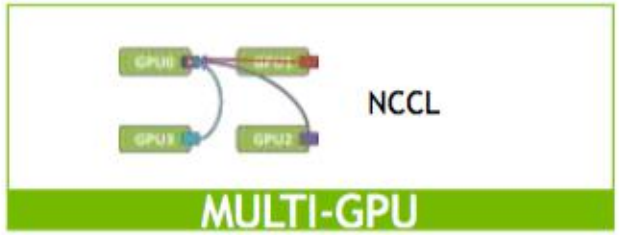
Hardware

Machine Learning – GPU acceleration









Source captured frin - Julie Bernauer – HPC Advisory Council Stanford Tutorial – 2017/02/07

Supervised learning is a type of machine learning which automate decision-making processes by generalizing from known examples. That is, the users provides the algorithm with pairs of inputs and desired outputs. The algorithm finds a way to produce the desired output given an input. In other words, we construct a model for the problem. The model is governed by some unknown parameters which we will learn from the data.

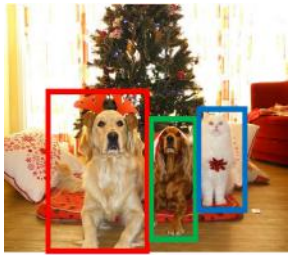
Supervised Learning Data:
(x, y) x is data, y is label

Goal: Learn a function to map x -> y

Examples:
Classification (discrete values),
regression (numerical values),

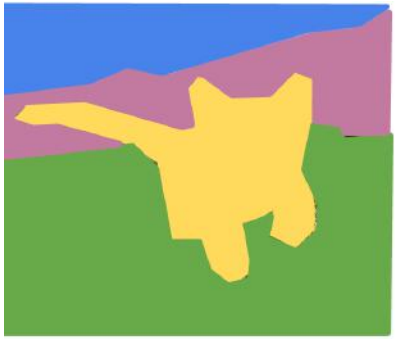
CAT

Classification



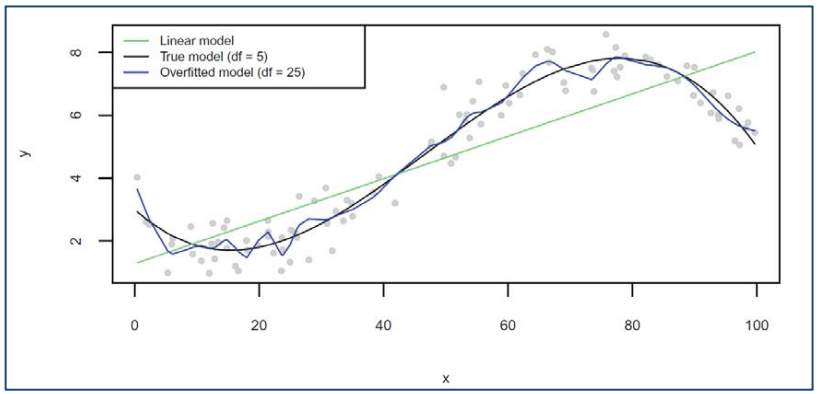
DOG, DOG, CAT

Object Detection



GRASS, CAT, TREE, SKY

Semantic Segmentation



Basic Ideas

Typical Neural Network – MLP

Convolutional Neural Network

STEP 1 : Model Definition

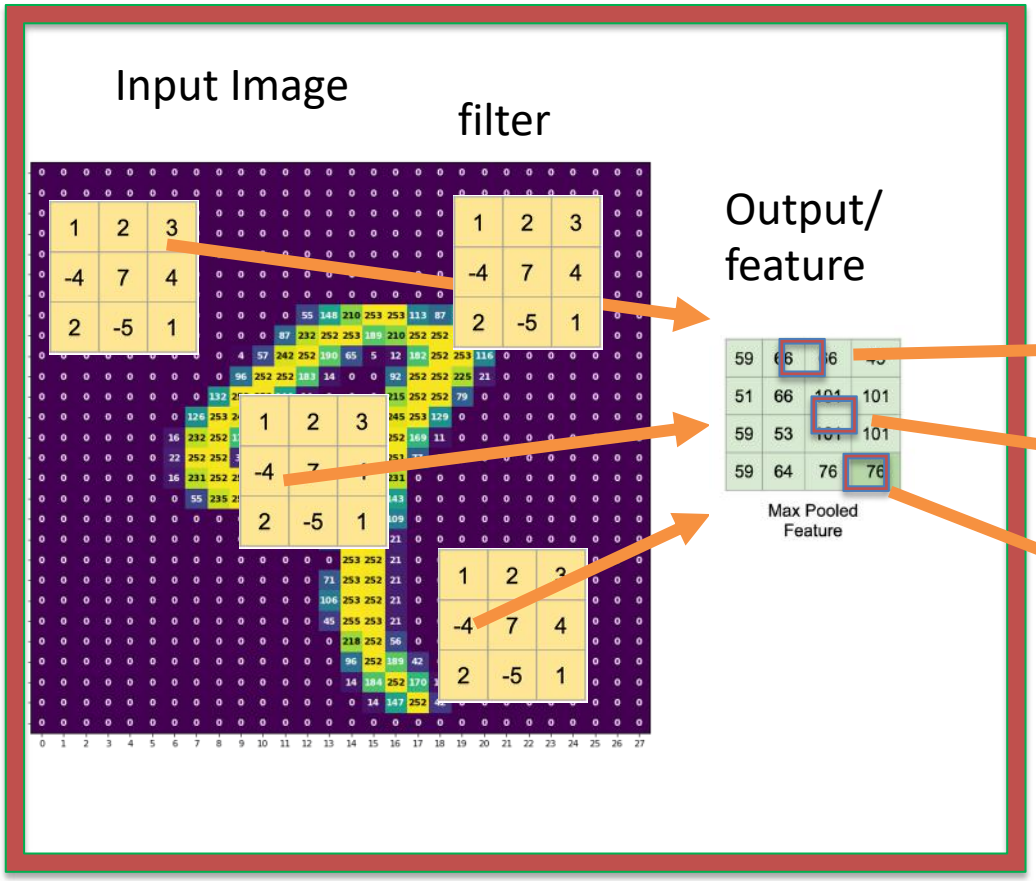
STEP 2 : Cost Function

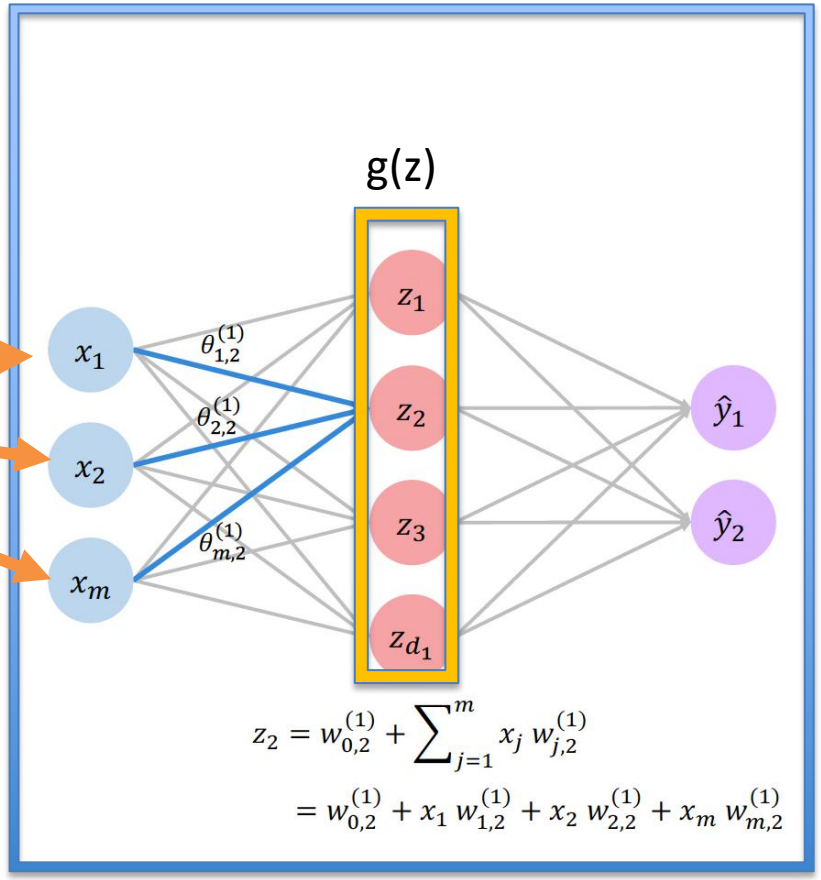
Step 3 : Optimization Scheme

Step 4 : Numerical Implementation (fitting)

Step 5 : Evaluation

Step 1: Parametric Model : Convolutional Neural Network (CNN)
Convolutional filter + Connected Neural Network



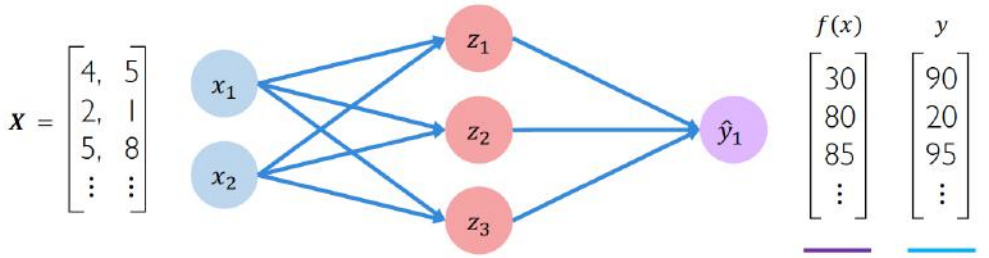


Convolutional filtering

Multilayer Perceptron

NN Cost Function (regression) : Mean Squared Error Loss

Mean squared error loss can be used with regression models that output continuous real numbers



$$J(W) = \frac{1}{n}\sum_{i=1}^{n}\left(y^{(i)} - f(x^{(i)}; W)\right)^2$$

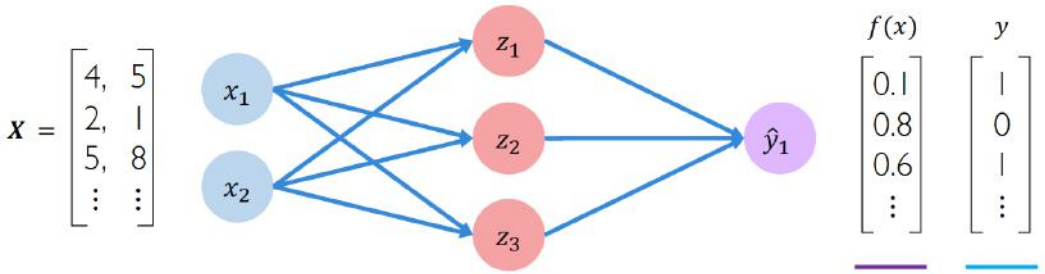
Actual Predicted

```
loss = tf.reduce_mean( tf.square(tf.subtract(model.y, model.pred) )
```

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \tilde{y}_i)^2$$

NN Cost Function (Classification) : Cross Entropy Loss

Cross entropy loss can be used with models that output a probability between 0 and 1



$$J(W) = \frac{1}{n}\sum_{i=1}^{n} y^{(i)} \log\left(f(x^{(i)}; W)\right) + (1 - y^{(i)}) \log\left(1 - f(x^{(i)}; W)\right)$$

Actual Predicted Actual Predicted

```
loss = tf.reduce_mean( tf.nn.softmax_cross_entropy_with_logits(model.y, model.pred) )
```

$$L_i = -\log\left(\frac{e^{sy_i}}{\sum_j e^{s_j}}\right)$$

DNN Forward Backward Calculation : Weights and bias

Algorithm

1. Initialize weights randomly $\sim \mathcal{N}(0, \sigma^2)$

```
weights = tf.random_normal(shape, stddev=sigma)
```

2. Loop until convergence:
3. Compute gradient, $\frac{\partial J(W)}{\partial W}$

```
grads = tf.gradients(ys=loss, xs=weights)
```

4. Update weights, $W \leftarrow W - \eta \frac{\partial J(W)}{\partial W}$

```
weights_new = weights.assign(weights - lr * grads)
```

5. Return weights

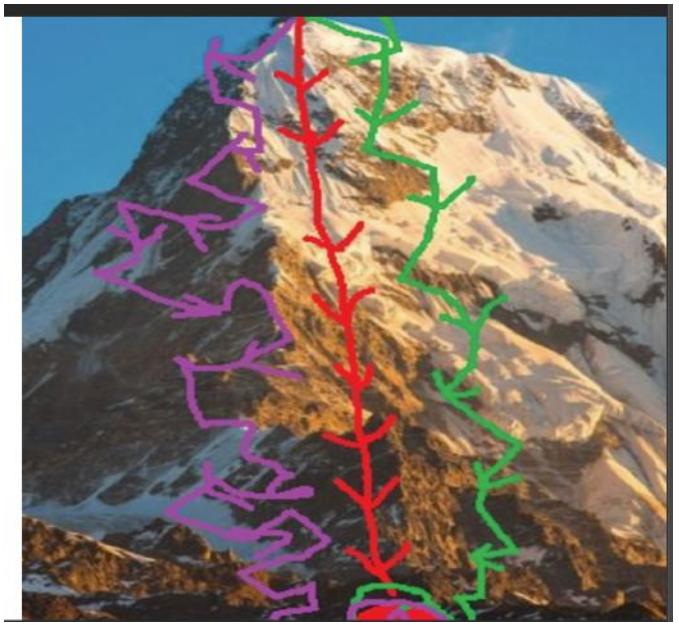
W+ = W – (learning rate) * dJ/dw

In one epoch

1. Pick a mini-batch
2. Feed it to Neural Network
3. Forward path calculation
4. Calculate the mean gradient of the mini-batch
5. Use the mean gradient calculated in step 4 to update the weights
6. Repeat steps 1–5 for the mini-batches we created

Optimization Schemes:

Gradient Descent :

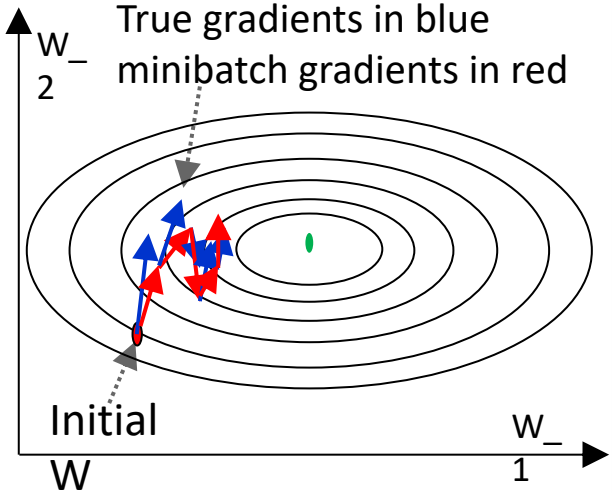


Optimization: Batch SGD

Instead of computing a gradient across the entire dataset with size N, divides the dataset into a fixed-size subset ("minibatch") with size m, and computes the gradient for each update on this smaller batch:

(N is the dataset size, m is the minibatch size)

$$\mathbf{g} = \frac{1}{m'}\sum_{i=1}^{m'} \nabla_{\boldsymbol{\theta}} E_i(\mathbf{x}^{(n_i)}, y^{(n_i)}, \boldsymbol{\theta}),$$
$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta\mathbf{g},$$



Epoch step

Given: training set $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$
Initialize all $\Theta^{(l)}$ randomly (NOT to 0!)
Loop // each iteration is called an epoch
  Loop // each iteration is a mini-batch
    Set $\Delta^{(l)}_{i,j} = 0 \quad \forall l, i, j$ (Used to accumulate gradient)
    Sample m training instances $\mathcal{X} = \{(\mathbf{x}'_1, y'_1), \ldots, (\mathbf{x}'_m, y'_m)\}$ without replacement
    For each instance in $\mathcal{X}$, $(\mathbf{x}_k, y_k)$:
      Set $\mathbf{a}^{(1)} = \mathbf{x}_k$
      Compute $\{\mathbf{a}^{(2)}, \ldots, \mathbf{a}^{(L)}\}$ via forward propagation
      Compute $\boldsymbol{\delta}^{(L)} = \mathbf{a}^{(L)} - y_k$
      Compute errors $\{\boldsymbol{\delta}^{(L-1)}, \ldots, \boldsymbol{\delta}^{(2)}\}$
      Compute gradients $\Delta^{(l)}_{ij} = \Delta^{(l)}_{ij} + a^{(l)}_j \delta^{(l+1)}_i$
    Compute mini-batch regularized gradient $D^{(l)}_{ij} = \begin{cases} \frac{1}{m}\Delta^{(l)}_{ij} + \lambda\Theta^{(l)}_{ij} & \text{if } j \neq 0 \\ \frac{1}{m}\Delta^{(l)}_{ij} & \text{otherwise} \end{cases}$
    Update weights via gradient step $\Theta^{(l)}_{ij} = \Theta^{(l)}_{ij} - \alpha D^{(l)}_{ij}$
  Until all training instances are seen
Until weights converge or max #epochs is reached

Batch iteration

Training

"Training a neural network"

What does it mean?

What does the network train?

minimize error of the computed values against the labeled values (loss function)

What are the training parameters

W, b ,**and values of the filters (Wf)**

How does it work?

optimization based on gradient descent algorithm

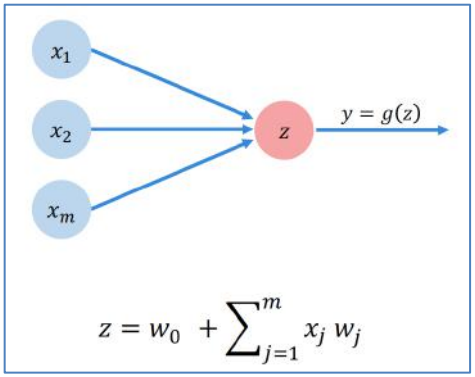
go back and forth in the NN model to adjust for w and b => need derivatives w.r.t. parameters

Go through forward calculation

training with backpropagation

Forward and backward computation operators

Z = (x1*w1 + x2*w2) + b = (-1.0*-2.0 + -2.0*3.0) + -3.0 = 4.0 – 3.0 = 1.0



Sigmoid function

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

Y = σ (z) = 1 / (1 + e $^{-Z}$)
= 1 / (1 + e $^{-1}$) = 0.731

Z = (x1*w1 + x2*w2) + b

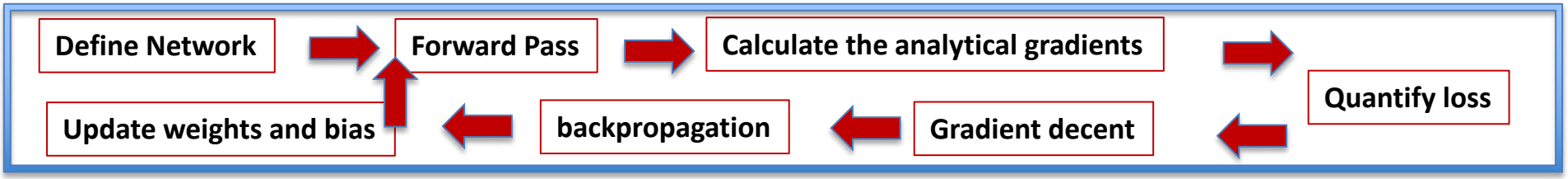
multiple operator :
downstream gradient =
upstream gradient * input value

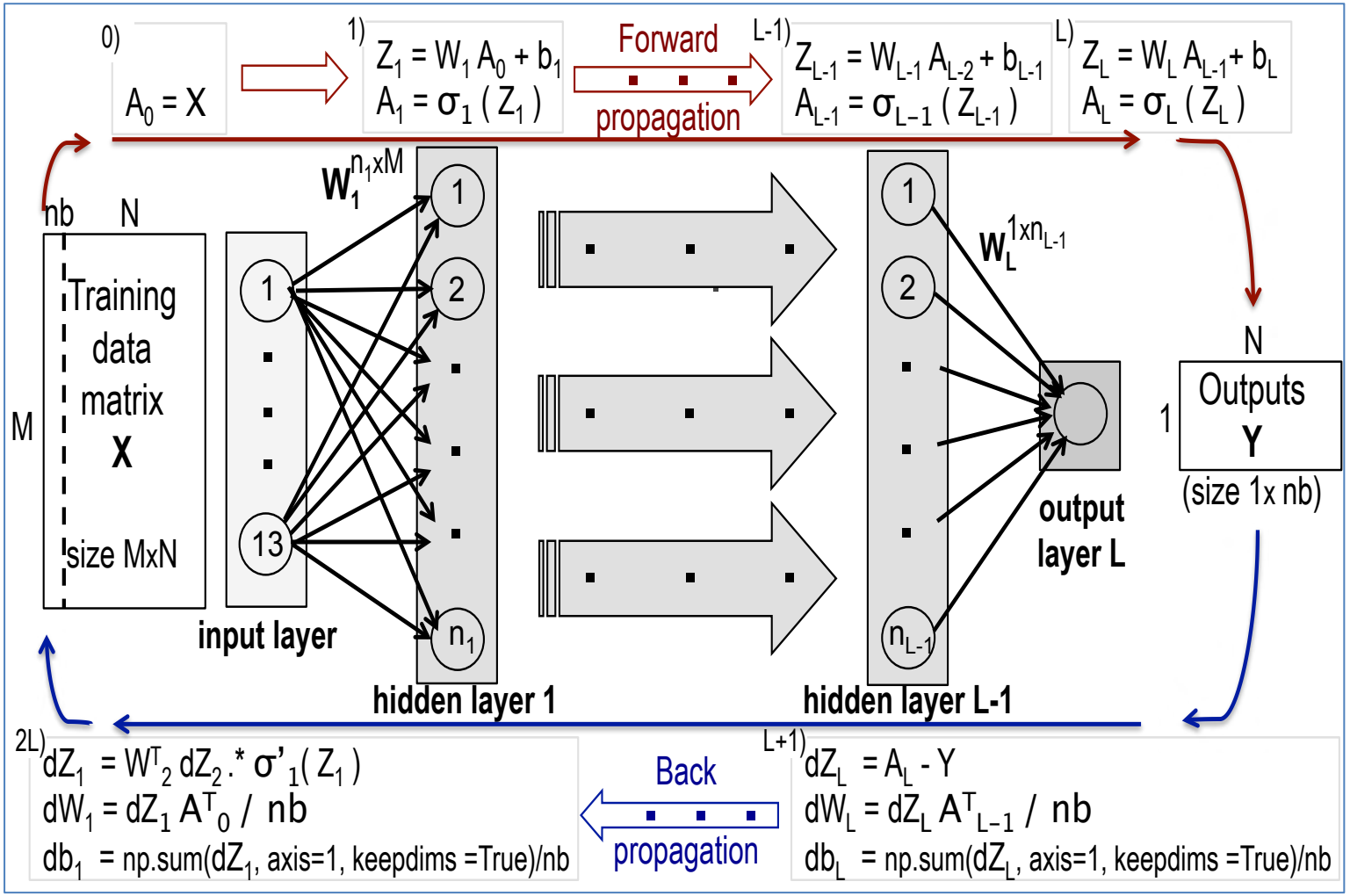
add operator :
Downstream gradient =
upstream gradient

function operator :
downstream gradient =
Upstream gradient * local function gradient

Summary : Flow of NN

```
model.fit(x_train, y_train, epochs=3, batch_size=10)
```

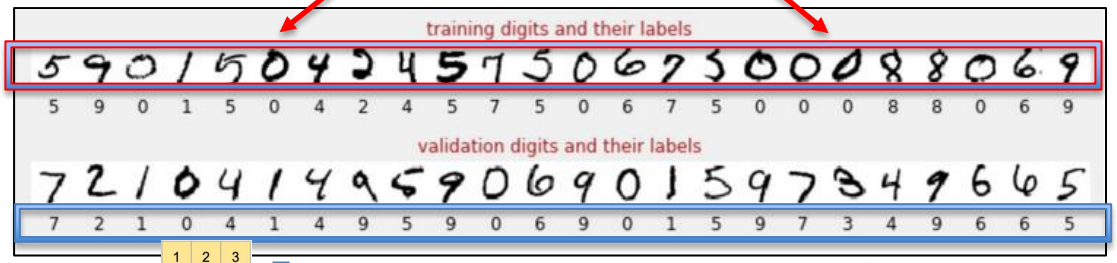




MNIST Example (28x28 pixels image)

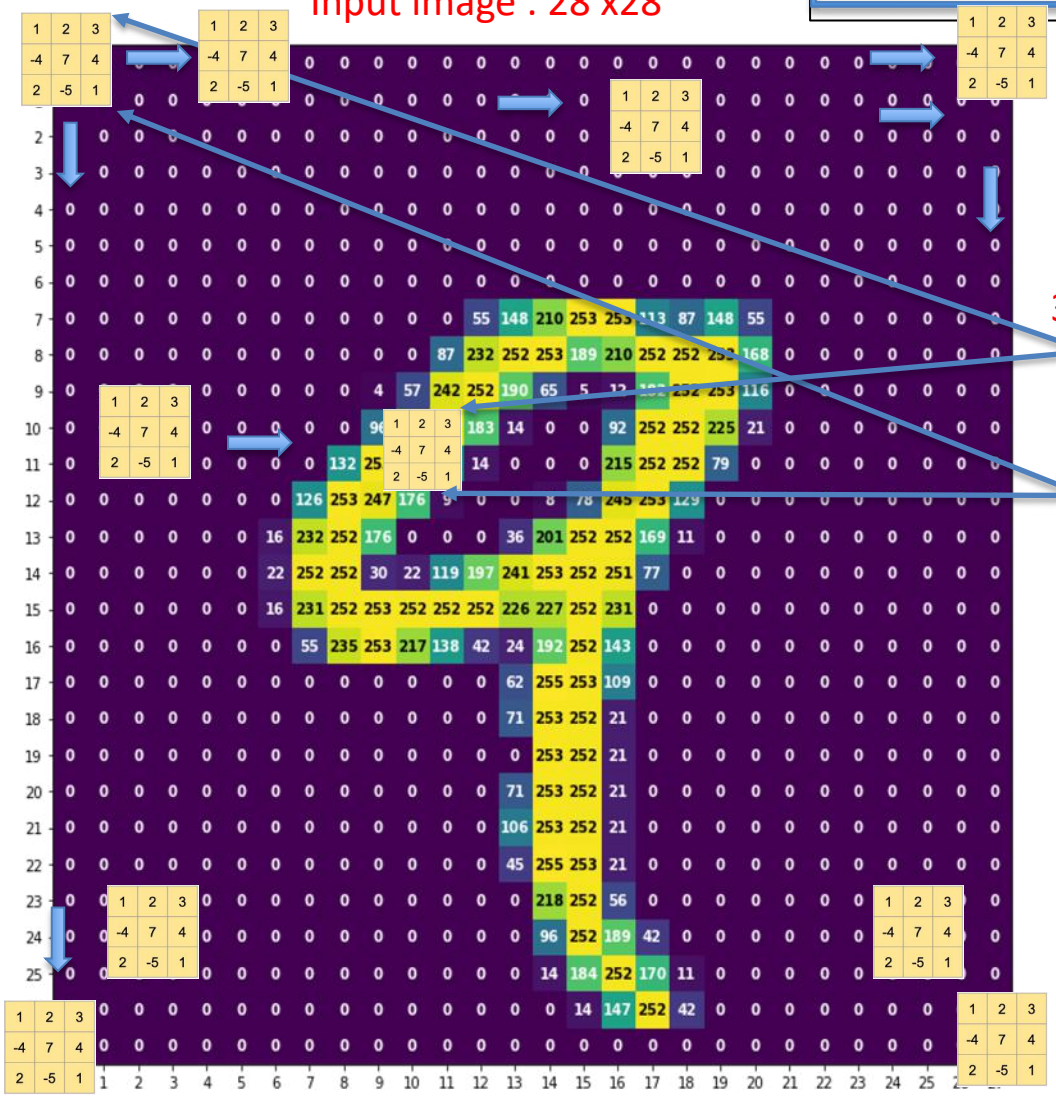
Handwritten digits in the MNIST dataset are 28x28 pixel grayscale images, use the original image (28x28 matrix).

Image : input



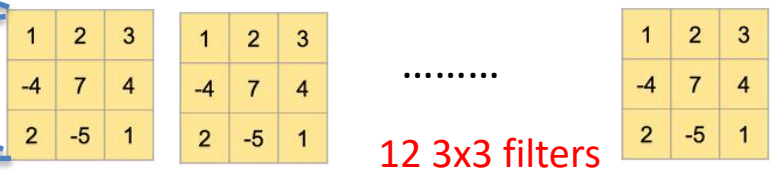
Labels : target

Input image : 28 x28

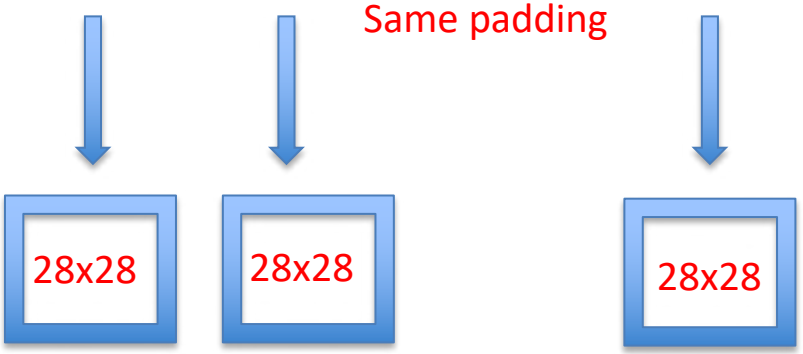


12 filters, each one acts on an image until it is fully covered.

3 x 3 filter (kernel)

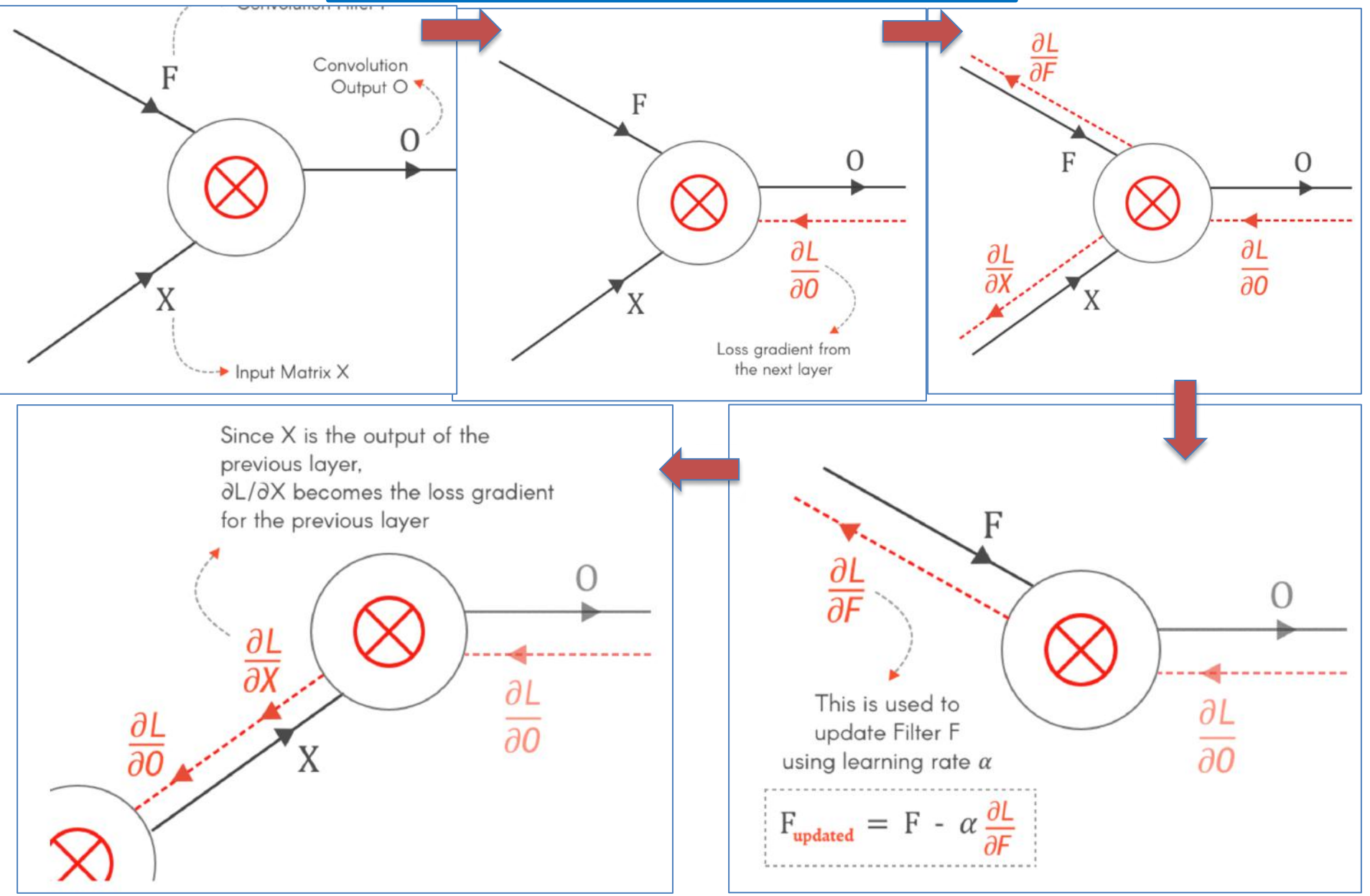


12 3x3 filters
Same padding



12 Output features : each 28 x28 matrix

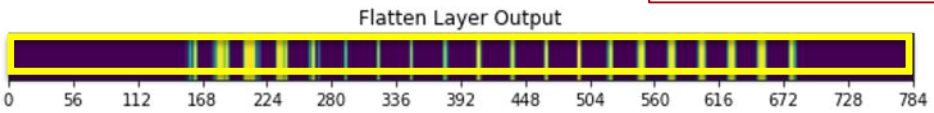
Forward path : backward path

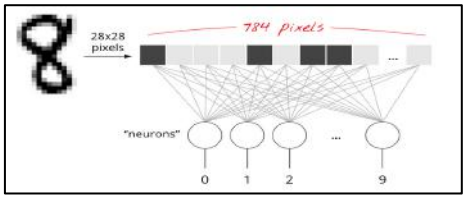


https://medium.com/@pavisj/convolutions-and-backpropagations-46026a8f5d2c

Summary : Flow of MLP Dense layer NN

1 image = a vector of 784

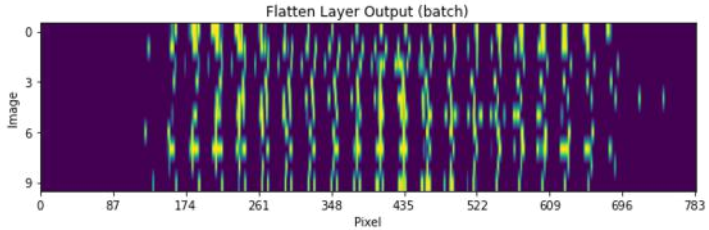




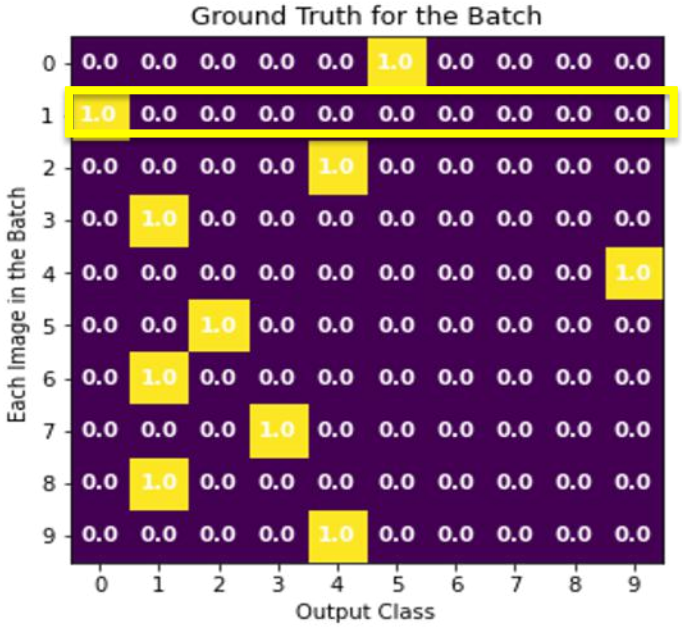
An image of 28x28 pixels

Input to 1st layer

10 images : batch of 10
10 x 784 matrix

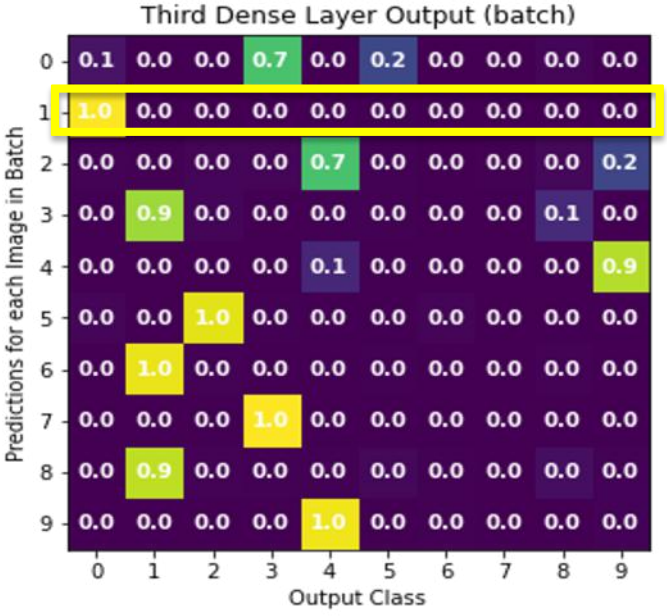


10 x 10 labeled matrix

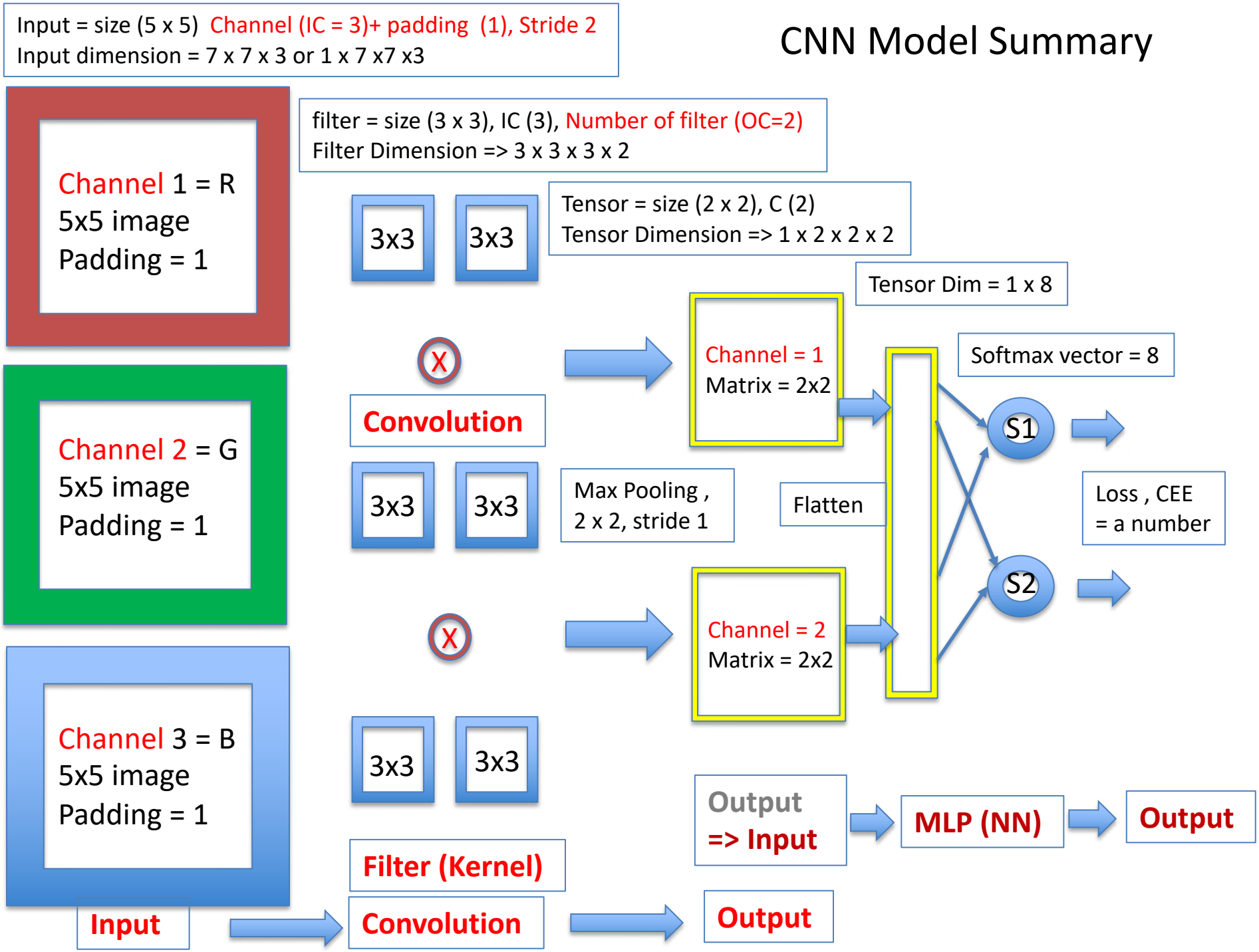


Comparing Computed NN results To Labeled results (target)

10 x 10 output matrix



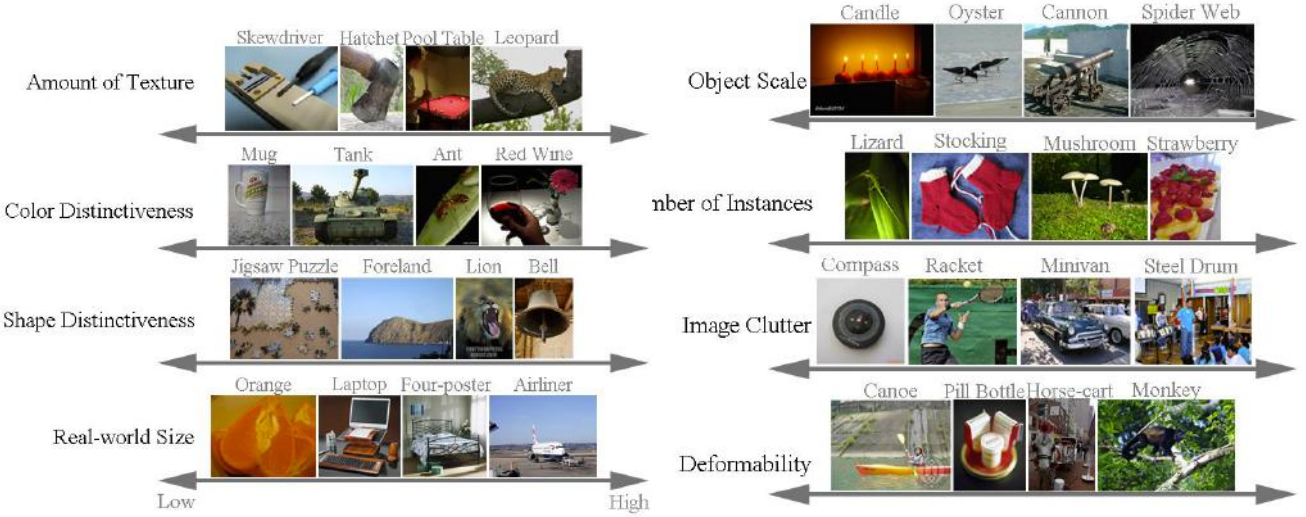
CNN Model Summary

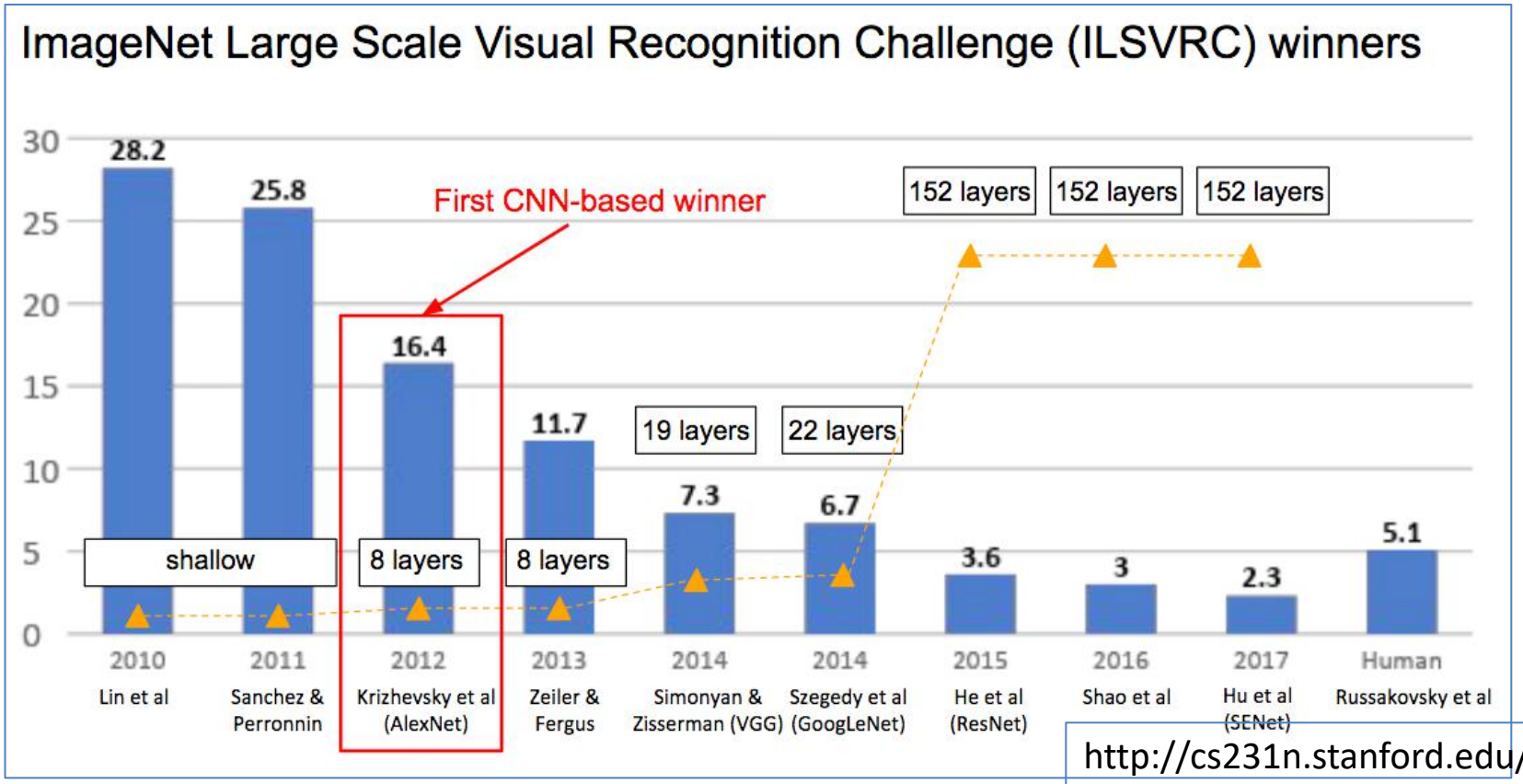


Topics of Unit 11

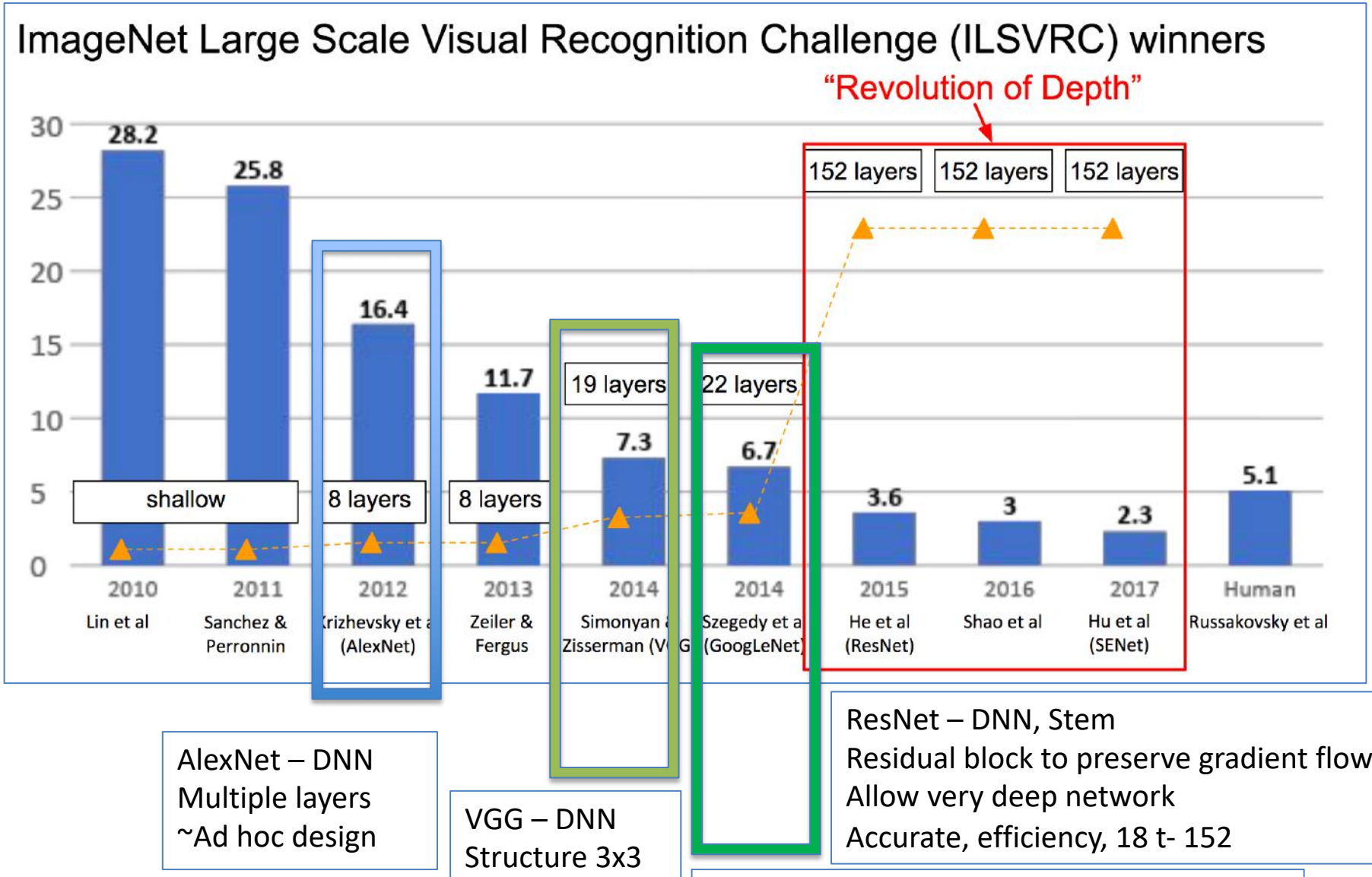
- ✓ **UNIT 11 : DNN Computing**
  - **DNN Principles: Mathematics**
  - **CNN Model: ResNet**
  - **DNN Computing: Essential Practices**
  - **DNN Usage: I/O, Transfer Learning**
  - **CNN Applications: Unet, Object Detection, Autonomous Vehicle**
  - **Advanced Topics: GAN, Reinforcement Learning**

- ✓ 1000 classes
- ✓ RGB images
- ✓ Cluttered Images (one category)
- ✓ Full resolution images
- ✓ 1.2 million training images
- ✓ 100 thousand test images





http://cs231n.stanford.edu/



AlexNet – DNN
Multiple layers
~Ad hoc design

VGG – DNN
Structure 3x3
Deep NN,
Lots of parameters
Accuracy

GoogleNet – DNN
Stem to reduce feature size
Block structure for efficiency
Group pooling to reduce parameter count

ResNet – DNN, Stem
Residual block to preserve gradient flow
Allow very deep network
Accurate, efficiency, 18 t- 152

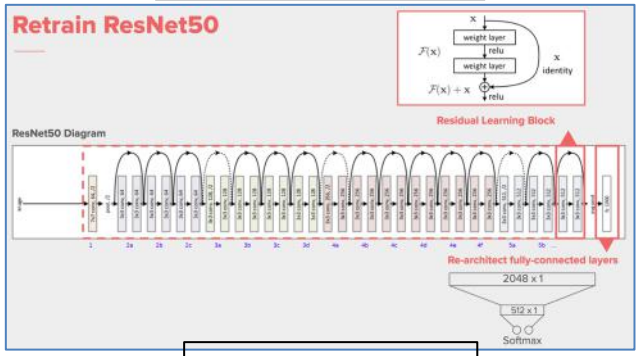
http://cs231n.stanford.edu/

Neural Network Models

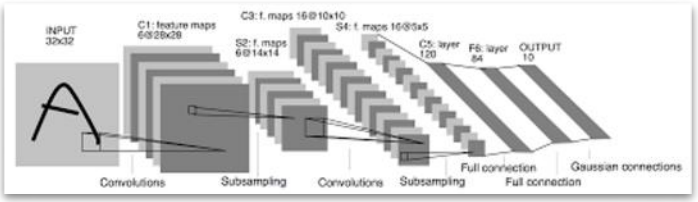
Dataset → Pre-defined Neural Network Model → Fully connected layer # neurons = # classes → Classification

Examples of Pre-defined Neural Network Models

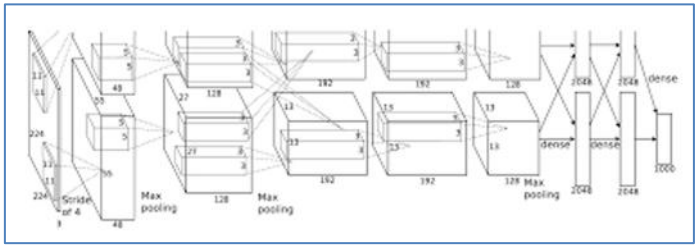
ResNet50



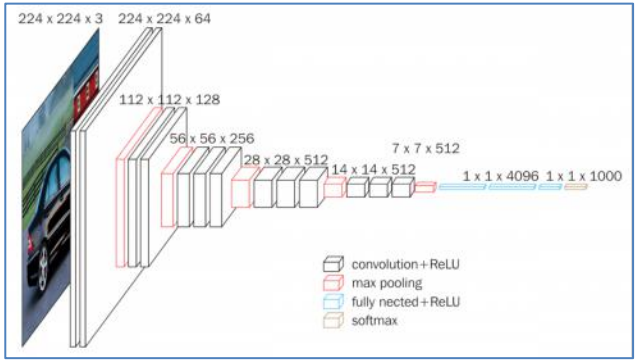
LeNet5



AlexNet

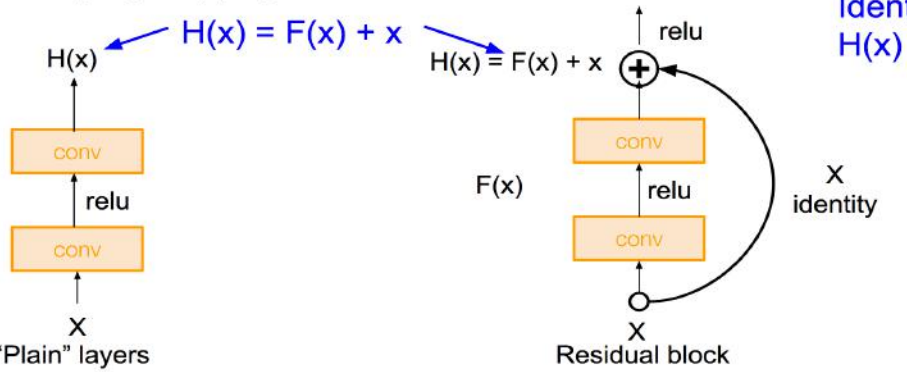


VGG16



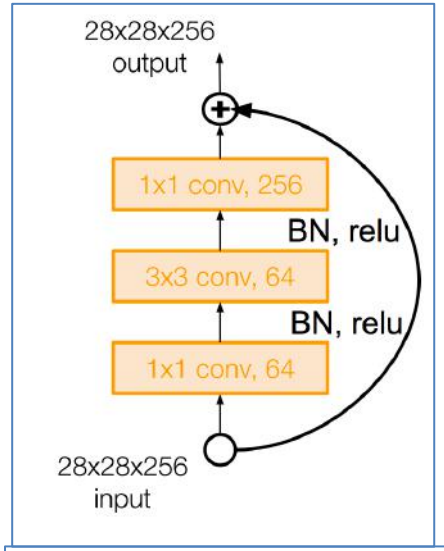
ResNet

Solution: Use network layers to fit a residual mapping instead of directly trying to fit a desired underlying mapping



Identity mapping:
H(x) = x if F(x) = 0

Use layers to fit **residual** F(x) = H(x) - x instead of H(x) directly



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

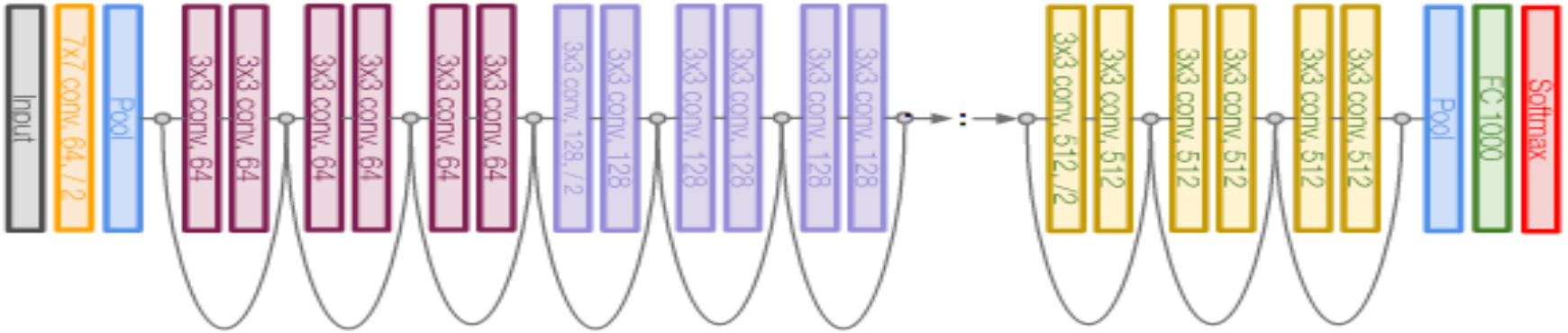
- ✓ Full ResNet architecture:-
- ✓ Stack residual blocks
- ✓ Every residual block has two 3x3 conv layers
- ✓ Periodically, double # of filters and downsample spatially using stride 2 (/2 in each dimension)
- ✓ Additional conv layer at the beginning (stem)
- ✓ No FC layers at the end (only FC 1000 to output classes)

Total depths of 18, 34, 50, 101, or 152 layers for ImageNet

1x1 conv, 256 filters projects back to 256 feature maps (28x28x256)

3x3 conv operates over only 64 feature maps

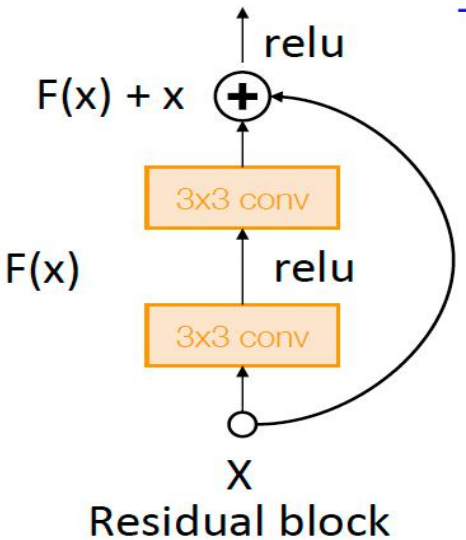
1x1 conv, 64 filters to project to 28x28x64

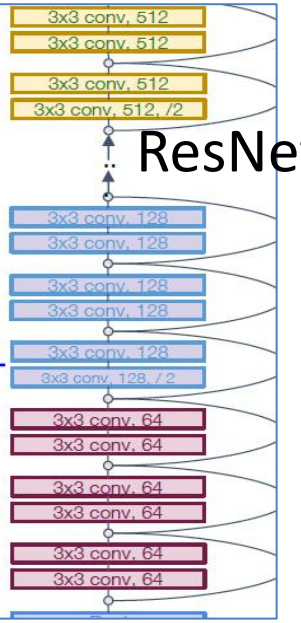


A residual network is a stack of many residual blocks

Regular design, like VGG: each residual block has two 3x3 conv

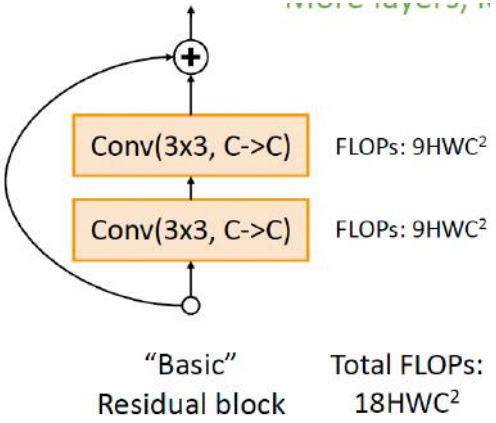
Network is divided into **stages**: the first block of each stage halves the resolution (with stride-2 conv) and doubles the number of channels





ResNet

Basic Block
ResNet18, 34



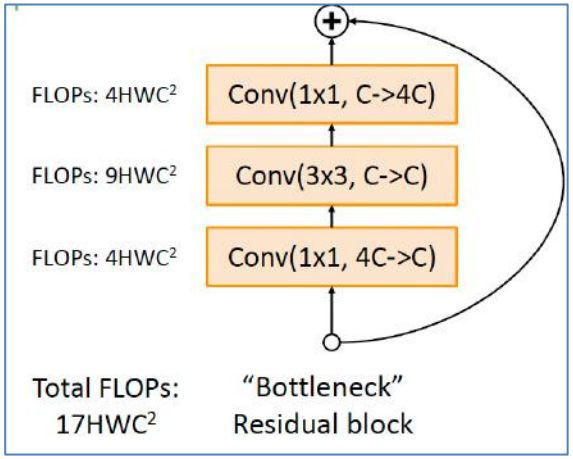
ResNet-18:
Stem: 1 conv layer
Stage 1 (C=64): 2 res. block = 4 conv
Stage 2 (C=128): 2 res. block = 4 conv
Stage 3 (C=256): 2 res. block = 4 conv
Stage 4 (C=512): 2 res. block = 4 conv
Linear

ImageNet top-5 error: 10.92
GFLOP: 1.8

ResNet-34:
Stem: 1 conv layer
Stage 1: 3 res. block = 6 conv
Stage 2: 4 res. block = 8 conv
Stage 3: 6 res. block = 12 conv
Stage 4: 3 res. block = 6 conv
Linear

ImageNet top-5 error: 8.58
GFLOP: 3.6

Bottlenect Block
ResNet50 …



Data Input and Treatment

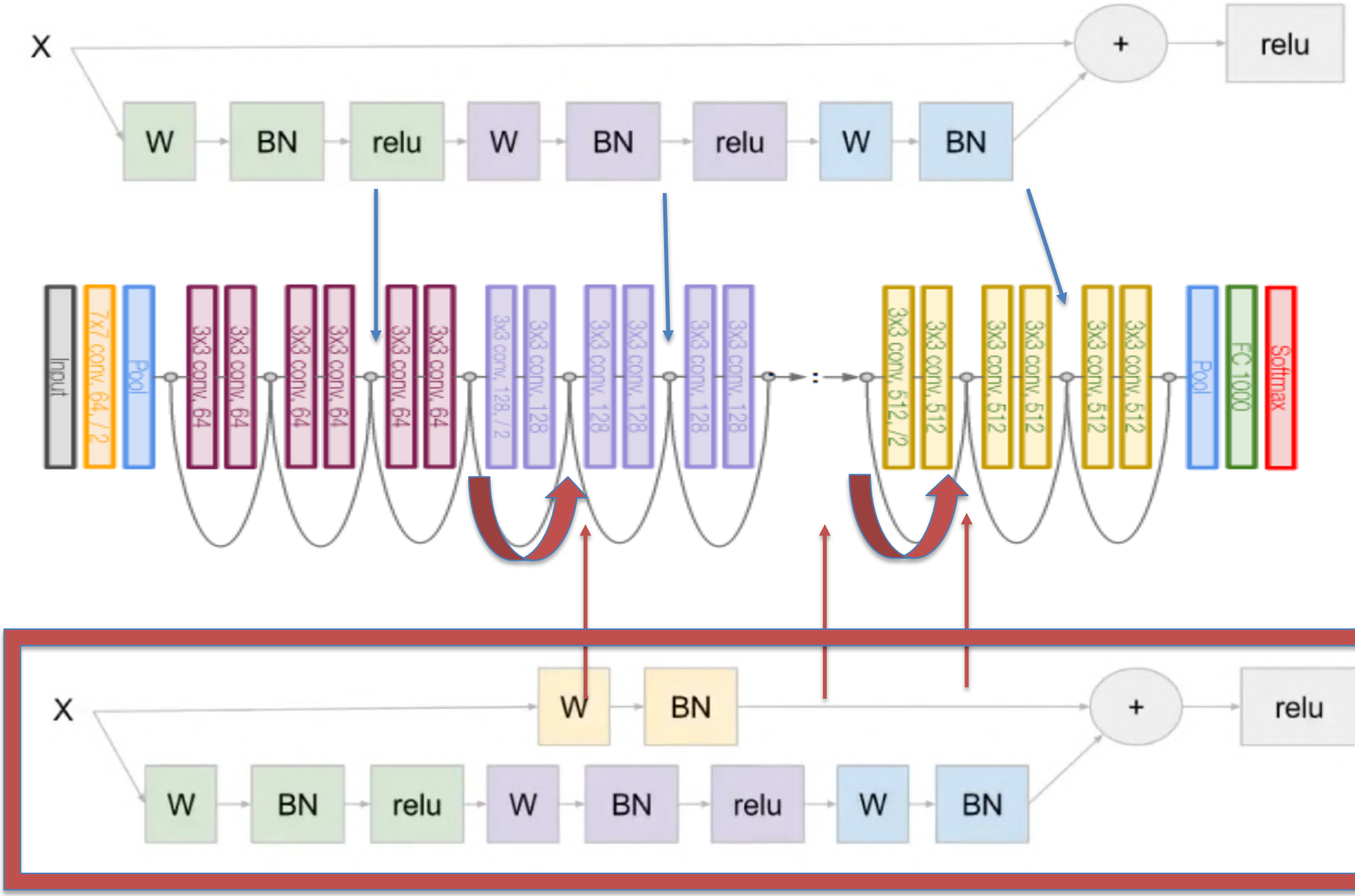
- This code transform the input data in three different ways, by adding a padding border of 4 pixels, by randomly flipping the image horizontally(default value is 0.5), and normalizing the data.

```
transform_train = transforms.Compose([
    transforms.RandomCrop(32, padding=4),   # (output size, border padding size)
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),              # changes to tensor since normalize works on a tensor variable
    transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010)),   # (mean, std)
])
```

- Also in pytorch, we can call torch.nn.DataParallel() to allow for the implementation of data parallelism at the module level. This helps speed up the training significantly
- Cudnn.benchmark will look for the optimal set of algorithms for that particular configuration given that input size is constant. This usually leads to faster runtime.

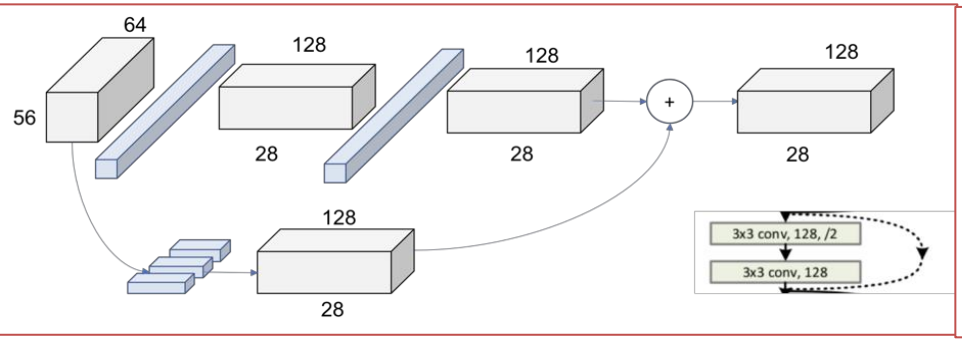
```
if device == 'cuda':
    net = torch.nn.DataParallel(net)
    cudnn.benchmark = True
```

Resnet 18 using Pytorch and Tensorflow

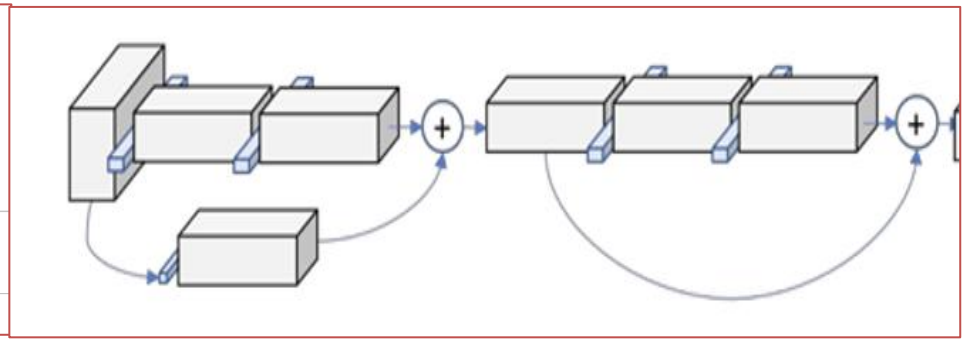


https://www.youtube.com/watch?v=0tBPSxioIZE

https://www.youtube.com/watch?v=jio04YvgraU



Basic (3x3, 3x3) Block



Bottleneck (1x1,3x3, 1x1) Block

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

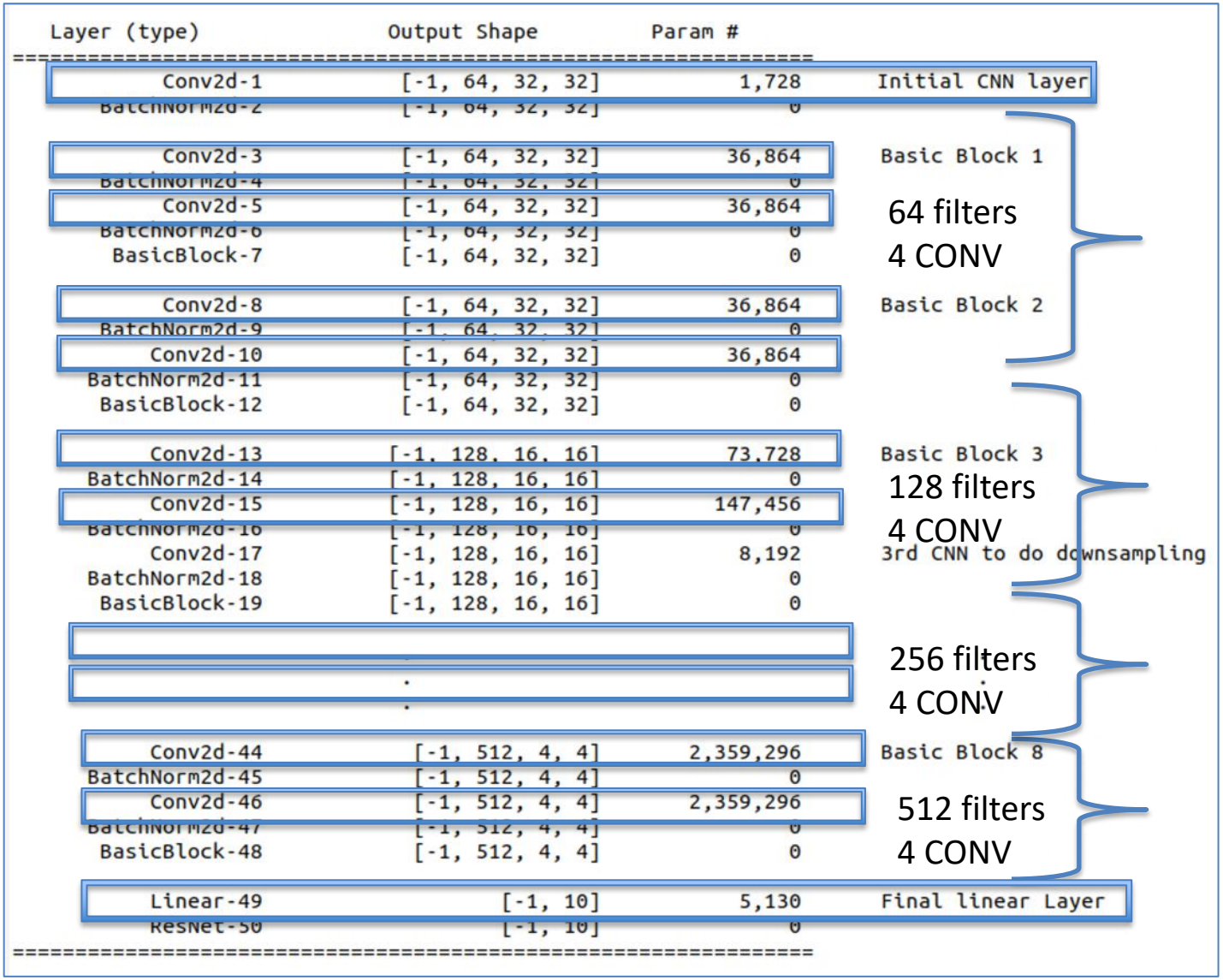
Kaiming He, Deep Residual Learning for Image Recognition

https://arxiv.org/pdf/1512.03385.pdf

```
DataParallel(
  (module): ResNet(
    (conv1): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=False, track_running_stats=True)
    (layer1): Sequential(
      (0): BasicBlock(
        (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=False, track_running_stats=True)
        (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=False, track_running_stats=True)
        (shortcut): Sequential()
      )
      (1): BasicBlock(
        (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=False, track_running_stats=True)
        (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=False, track_running_stats=True)
        (shortcut): Sequential()
      )
    )
                                                .
                                                .
                                                .
    (layer4): Sequential(
      (0): BasicBlock(
        (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=False, track_running_stats=True)
        (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=False, track_running_stats=True)
        (shortcut): Sequential(
          (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
          (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=False, track_running_stats=True)
        )
      )
      (1): BasicBlock(
        (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=False, track_running_stats=True)
        (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=False, track_running_stats=True)
        (shortcut): Sequential()
      )
    )
    (linear): Linear(in_features=512, out_features=10, bias=True)
  )
)
```

Resnet 18 – Model Summary

- ✓ Get this output from torchsummary.
- ✓ Main difference is that there is no pooling layers in this model compared to the original model due to the small size of the CIFAR10 data



Layer (type)	Output Shape	Param #	
Conv2d-1	[-1, 64, 32, 32]	1,728	Initial CNN layer
BatchNorm2d-2	[-1, 64, 32, 32]	0	
Conv2d-3	[-1, 64, 32, 32]	36,864	Basic Block 1
BatchNorm2d-4	[-1, 64, 32, 32]	0	
Conv2d-5	[-1, 64, 32, 32]	36,864	64 filters
BatchNorm2d-6	[-1, 64, 32, 32]	0	
BasicBlock-7	[-1, 64, 32, 32]	0	4 CONV
Conv2d-8	[-1, 64, 32, 32]	36,864	Basic Block 2
BatchNorm2d-9	[-1, 64, 32, 32]	0	
Conv2d-10	[-1, 64, 32, 32]	36,864	
BatchNorm2d-11	[-1, 64, 32, 32]	0	
BasicBlock-12	[-1, 64, 32, 32]	0	
Conv2d-13	[-1, 128, 16, 16]	73,728	Basic Block 3
BatchNorm2d-14	[-1, 128, 16, 16]	0	
Conv2d-15	[-1, 128, 16, 16]	147,456	128 filters
BatchNorm2d-16	[-1, 128, 16, 16]	0	4 CONV
Conv2d-17	[-1, 128, 16, 16]	8,192	3rd CNN to do downsampling
BatchNorm2d-18	[-1, 128, 16, 16]	0	
BasicBlock-19	[-1, 128, 16, 16]	0	
			256 filters
	.		4 CONV
	.		
Conv2d-44	[-1, 512, 4, 4]	2,359,296	Basic Block 8
BatchNorm2d-45	[-1, 512, 4, 4]	0	
Conv2d-46	[-1, 512, 4, 4]	2,359,296	512 filters
BatchNorm2d-47	[-1, 512, 4, 4]	0	
BasicBlock-48	[-1, 512, 4, 4]	0	4 CONV
Linear-49	[-1, 10]	5,130	Final linear Layer
ResNet-50	[-1, 10]	0	

resnet18-keras.ipynb

```
class ResnetBlock(Model):
#A standard resnet block.
  def __init__(self, channels: int, down_sample=False):
      #channels: same as number of convolution kernels
      super().__init__()

      self.__channels = channels
      self.__down_sample = down_sample
      self.__strides = [2, 1] if down_sample else [1, 1]

      KERNEL_SIZE = (3, 3)
      # use He initialization, instead of Xavier (a.k.a 'glorot_uniform' in Keras
      INIT_SCHEME = "he_normal"
      self.conv_1 = Conv2D(self.__channels, strides=self.__strides[0],
      kernel_size=KERNEL_SIZE, padding="same", kernel_initializer=INIT_SCHEME)
      self.bn_1 = BatchNormalization()
      self.conv_2 = Conv2D(self.__channels, strides=self.__strides[1],
      kernel_size=KERNEL_SIZE, padding="same", kernel_initializer=INIT_SCHEME)
      self.bn_2 = BatchNormalization()
      self.merge = Add()
      if self.__down_sample:
            # perform down sampling using stride of 2, according to [1].
            self.res_conv = Conv2D(
            self.__channels, strides=2, kernel_size=(1, 1), kernel_initializer=INIT_SCHEME,  padding="same")
            self.res_bn = BatchNormalization()

  def call(self, inputs):
      res = inputs
      x = self.conv_1(inputs)
      x = self.bn_1(x)
      x = tf.nn.relu(x)
      x = self.conv_2(x)
      x = self.bn_2(x)
      if self.__down_sample:
            res = self.res_conv(res)
            res = self.res_bn(res)

      # if not perform down sample, then add a shortcut directly
      x = self.merge([x, res])
      out = tf.nn.relu(x)
      return out
```

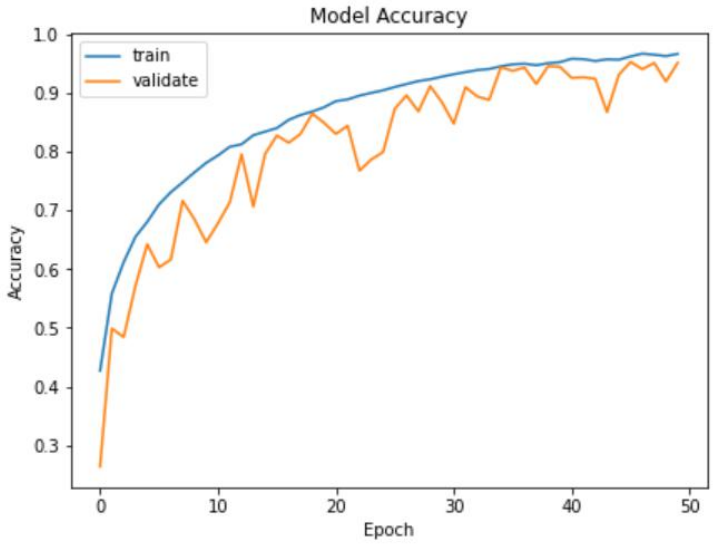
resnet18-keras.ipynb

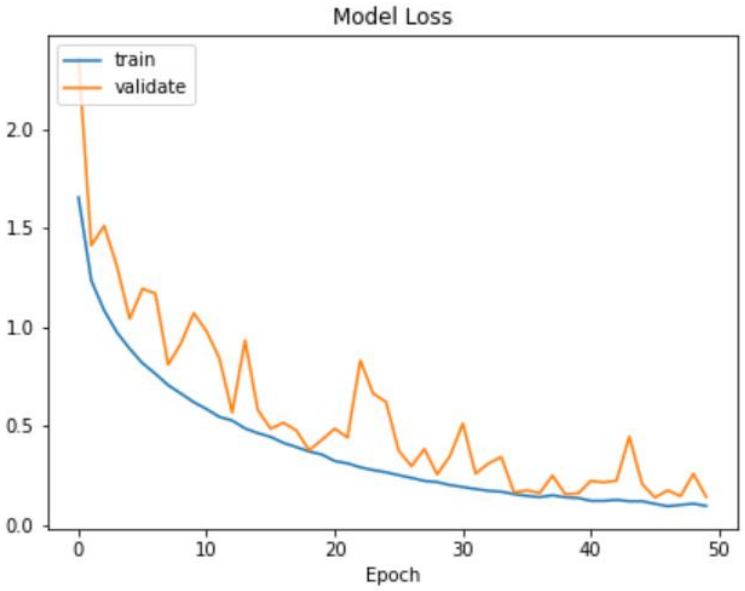
```
class ResNet18(Model):

def __init__(self, num_classes, **kwargs):
      #num_classes: number of classes in specific classification task.
      super().__init__(**kwargs)
      self.conv_1 = Conv2D(64, (7, 7), strides=2,
      padding="same", kernel_initializer="he_normal")
      self.init_bn = BatchNormalization()
      self.pool_2 = MaxPool2D(pool_size=(2, 2), strides=2, padding="same")
      self.res_1_1 = ResnetBlock(64)
      self.res_1_2 = ResnetBlock(64)
      self.res_2_1 = ResnetBlock(128, down_sample=True)
      self.res_2_2 = ResnetBlock(128)
      self.res_3_1 = ResnetBlock(256, down_sample=True)
      self.res_3_2 = ResnetBlock(256)
      self.res_4_1 = ResnetBlock(512, down_sample=True)
      self.res_4_2 = ResnetBlock(512)
      self.avg_pool = GlobalAveragePooling2D()
      self.flat = Flatten()
      self.fc = Dense(num_classes, activation="softmax")

  def call(self, inputs):
      out = self.conv_1(inputs)
      out = self.init_bn(out)
      out = tf.nn.relu(out)
      out = self.pool_2(out)
      for res_block in [self.res_1_1, self.res_1_2, self.res_2_1, self.res_2_2, self.res_3_1, self.res_3_2,
            self.res_4_1, self.res_4_2]:
            out = res_block(out)
            out = self.avg_pool(out)
            out = self.flat(out)
            out = self.fc(out)
      return out
```

```
model = ResNet18(10)
model.build(input_shape = (None,32,32,3))
#use categorical_crossentropy since the label is one-hot encoded
SGD = tf.keras.optimizers.SGD
# opt = SGD(learning_rate=0.1,momentum=0.9,decay = 1e-04) #parameters suggested by He [1]
model.compile(optimizer = "adam",loss='categorical_crossentropy', metrics=["accuracy"])
model.summary()
```

Google drive : QA -3 : resnet18-keras.ipynb

```
Model: "res_net18"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              multiple                  9472
_________________________________________________________________
batch_normalization (BatchNo multiple                  256
_________________________________________________________________
max_pooling2d (MaxPooling2D) multiple                  0
_________________________________________________________________
resnet_block (ResnetBlock)   multiple                  74368
_________________________________________________________________
resnet_block_1 (ResnetBlock) multiple                  74368
_________________________________________________________________
resnet_block_2 (ResnetBlock) multiple                  231296
_________________________________________________________________
resnet_block_3 (ResnetBlock) multiple                  296192
_________________________________________________________________
resnet_block_4 (ResnetBlock) multiple                  921344
_________________________________________________________________
resnet_block_5 (ResnetBlock) multiple                  1182208
_________________________________________________________________
resnet_block_6 (ResnetBlock) multiple                  3677696
_________________________________________________________________
resnet_block_7 (ResnetBlock) multiple                  4723712
_________________________________________________________________
global_average_pooling2d (Gl multiple                  0
_________________________________________________________________
flatten (Flatten)            multiple                  0
_________________________________________________________________
dense (Dense)                multiple                  5130
=================================================================
Total params: 11,196,042
Trainable params: 11,186,442
Non-trainable params: 9,600
```





Topics of Unit 11

- ✓ **UNIT 11 : DNN Computing**
  - **DNN Principles: Mathematics**
  - **CNN Model: ResNet**
  - **DNN Computing: Essential Practices**
  - **DNN Usage: I/O, Transfer Learning**
  - **CNN Applications: Unet, Object Detection, Autonomous Vehicle**
  - **Advanced Topics: GAN, Reinforcement Learning**

Network Initialization

w = np.random.randn(n) / sqrt(n)

Calibrating the variances with 1/sqrt(n). For deep network, the problem is that the distribution of the outputs from a randomly initialized neuron has a variance that grows with the number of inputs. It turns out that we can normalize the variance of each neuron's output to 1 by scaling its weight vector by the square root of its *fan-in* (i.e. its number of inputs). That is, the recommended heuristic is to initialize each neuron's weight vector as: w = np.random.randn(n) / sqrt(n), where n is the number of its inputs. This ensures that all neurons in the network initially have approximately the same output distribution and empirically improves the rate of convergence.

NO, NO: All zero initialization. : if every neuron in the network computes the same output, then the derivative with respect to loss function is the same for every w, thus all weights have the same value in subsequent iterations.

NOT GOOD : Small random numbers : A Neural Network layer that has very small weights will during backpropagation compute very small gradients on its data. This could greatly diminish the "gradient signal" flowing backward through a network, and could become a concern for deep networks.

An initialization specifically for ReLU neurons, reaching the conclusion that the variance of neurons in the network should be $2.0/n$. This gives the initialization w = np.random.randn(n) * sqrt(2.0/n) and is the current recommendation for use in practice in the specific case of neural networks with ReLU neurons.

w = np.random.randn(n) *2/ sqrt(n)

Neural Network Initialization

Weight Initialization: "Xavier" Initialization

```
dims = [4096] * 7
hs = []
x = np.random.randn(16, dims[0])
for Din, Dout in zip(dims[:-1], dims[1:]):
    W = np.random.randn(Din, Dout) / np.sqrt(Din)
    x = np.tanh(x.dot(W))
    hs.append(x)
```

"Xavier" initialization:
std = 1/sqrt(Din)

"Just right": Activations are nicely scaled for all layers!

For conv layers, Din is kernel_size^2 * input_channels

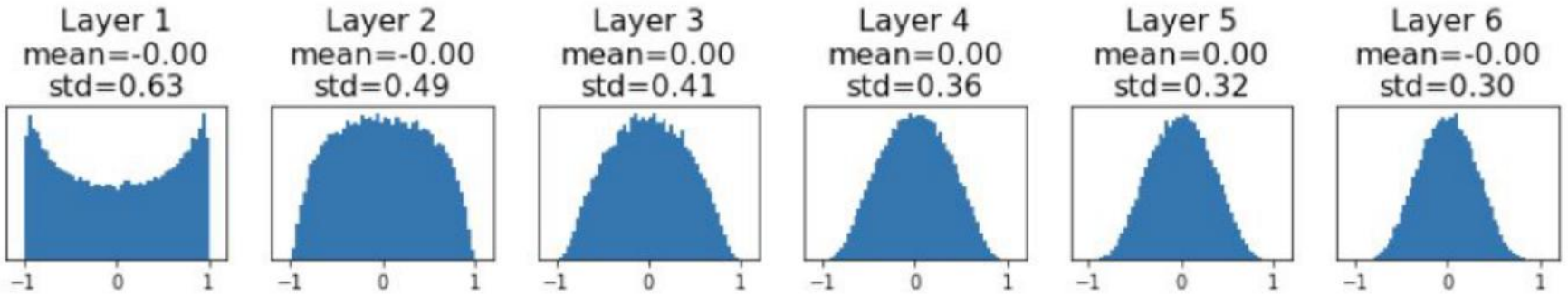
Derivation:

$y = Wx$
$h = f(y)$

$$\begin{aligned} Var(y_i) &= Din * Var(x_i w_i) && \text{[Assume x, w are iid]} \\ &= Din * (E[x_i^2]E[w_i^2] - E[x_i]^2 E[w_i]^2) && \text{[Assume x, w independant]} \\ &= Din * Var(x_i) * Var(w_i) && \text{[Assume x, w are zero-mean]} \end{aligned}$$

If $Var(w_i) = 1/Din$ then $Var(y_i) = Var(x_i)$

Glorot and Bengio, "Understanding the difficulty of training deep feedforward neural networks", AISTAT 2010



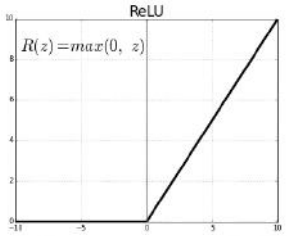
http://cs231n.stanford.edu/slides/2020/lecture_7.pdf

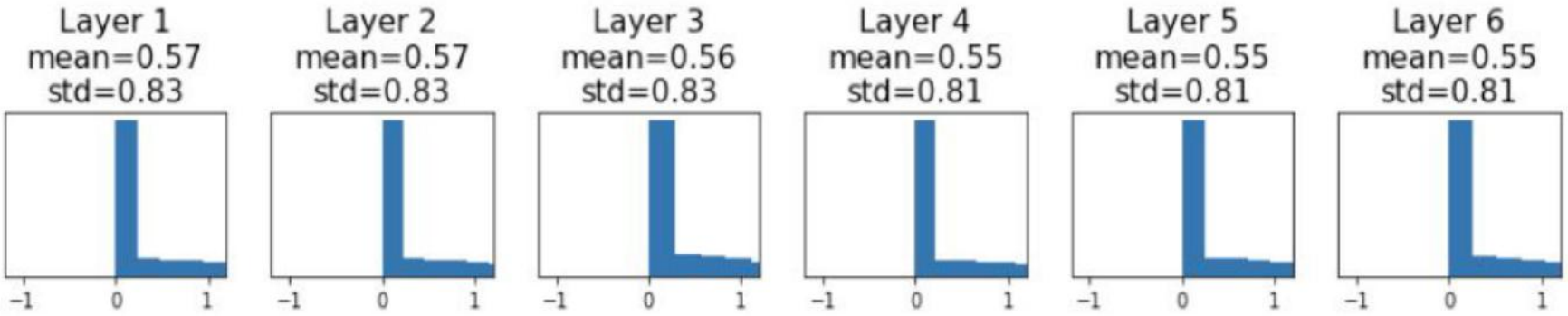
Weight Initialization: Kaiming / MSRA Initialization

```
dims = [4096] * 7
hs = []
x = np.random.randn(16, dims[0])
for Din, Dout in zip(dims[:-1], dims[1:]):
    W = np.random.randn(Din, Dout) * np.sqrt(2/Din)
    x = np.maximum(0, x.dot(W))
    hs.append(x)
```

ReLU correction: std = sqrt(2 / Din)

"Just right": Activations are nicely scaled for all layers!



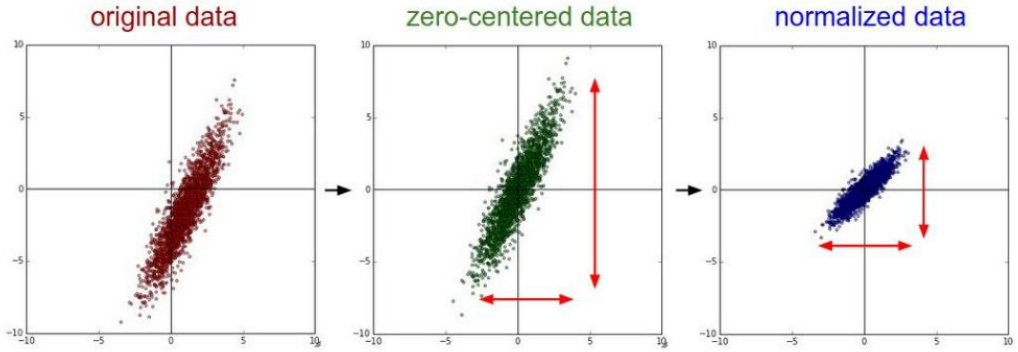


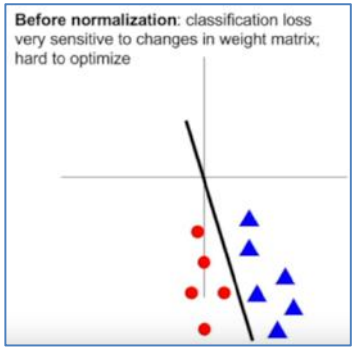
He et al, "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification", ICCV 2015

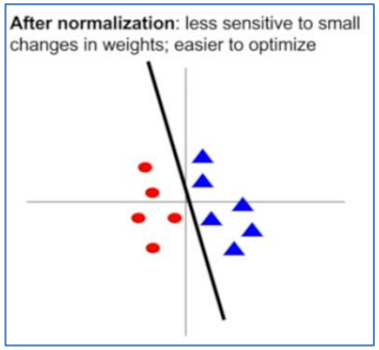
Initializing the biases. It is possible and common to initialize the biases to be zero. For ReLU non-linearities, some people like to use small constant value such as 0.01 for all biases because this ensures that all ReLU units fire in the beginning and therefore obtain and propagate some gradient.

http://cs231n.stanford.edu/slides/2020/lecture_7.pdf

Batched Normalization







- ✓ Batch normalization is one of the reasons why deep learning (CNN) has made such outstanding progress in recent years. Batch normalization enables the use of higher learning rates, greatly accelerating the learning process. It also enabled the training of deep neural networks with sigmoid activations that were previously deemed too difficult to train due to the vanishing gradient problem.
- ✓ Covariate shift refers to the change in the distribution of the input values to a learning algorithm. Since the activations of a previous layer are the inputs of the next layer, each layer in the neural network is faced with a situation where **the input distribution changes with each step**.
- ✓ The basic idea behind batch normalization is to limit covariate shift by normalizing the activations of each layer (transforming the inputs to be mean 0 and unit variance). This, supposedly, allows each layer to learn on a more stable distribution of inputs, and would thus accelerate the training of the network.
- ✓ Normalization is a simple differentiable operation

https://mlexplained.com/2018/01/10/an-intuitive-explanation-of-why-batch-normalization-really-works-normalization-in-deep-learning-part-1/

X = layers. BatchNormalization()(x)

Batch Normalization has 4 parameters per set of filters (feature)
Number of parameters = 4 x number of filters (bias) = 128, 256

As you can read there, in order to make the batch normalization work during training, they need to keep track of the distributions of each normalized dimensions. To do so, since you are in mode=0 by default, they compute 4 parameters per feature on the previous layer. Those parameters are making sure that you properly propagate and backpropagate the information. These parameters are in fact [gamma weights, beta weights, moving_mean(non-trainable), moving_variance(non-trainable)], of the size of the input layer.

https://en.wikipedia.org/wiki/Batch_normalization

input matrix = 80 x 80 x 64

X = layers.MaxPooling2D(3, stride= 2, padding='same')(x)
Reduce size by half because of stride, paddling

output matrix = 40 x 40 x 64

Batch Normalization

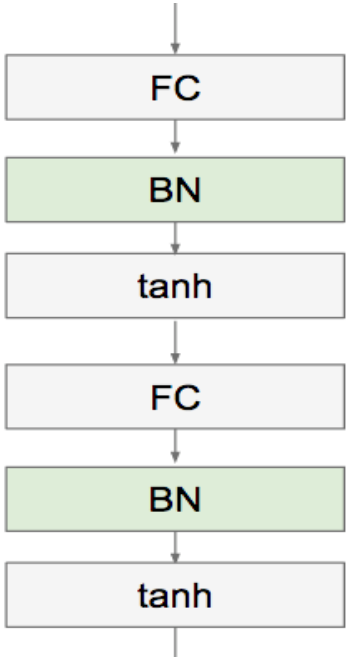
- ✓ Initializing neural networks by explicitly forcing the activations throughout a network to take on a unit Gaussian distribution at the beginning of the training.
- ✓ To make each dimension zero mean and unit-variance activation at a batch of activation at some layer
- ✓ Compute the empirical mean and variance independently for each dimension at every forward pass, then normalize the input
- ✓ Usually inserted after FC or Convolutional layers and before activation

Normalize:

$$\hat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

And then allow the network to squash the range if it wants to:

$$y^{(k)} = \gamma^{(k)}\hat{x}^{(k)} + \beta^{(k)}$$



$$\hat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

Note, the network can learn:

$$\gamma^{(k)} = \sqrt{\mathrm{Var}[x^{(k)}]}$$

$$\beta^{(k)} = \mathrm{E}[x^{(k)}]$$

to recover the identity mapping.

https://arxiv.org/pdf/1502.03167.pdf

http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture6.pdf

Batch Normalization

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

- ✓ Make Deep CNN to converge in training
- ✓ Improves gradient flow through the network
- ✓ Allows higher learning rates
- ✓ Reduces the strong dependence on initialization, more robust to initialization
- ✓ Acts as a form of regularization in a way (sort of randomization), and perhaps slightly reduces the need for dropout.

Note: at test time BatchNorm layer functions differently:

Batch Normalization: Test-Time

Estimates depend on minibatch; can't do this at test-time!

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

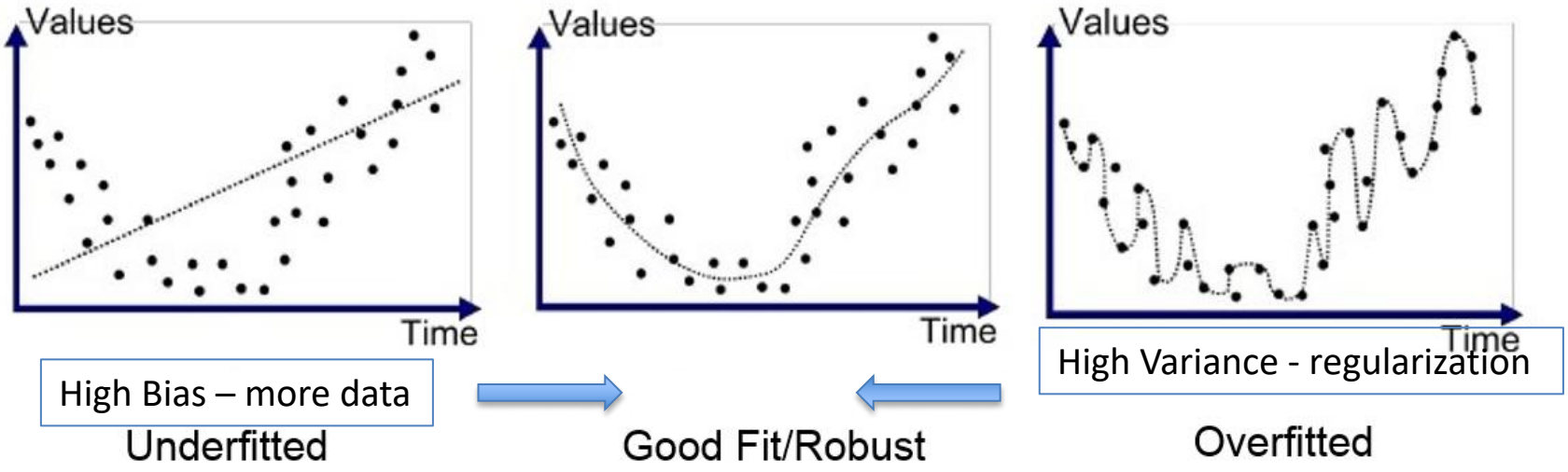
$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$ Normalized x, Shape is N x D

$$y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j$$ Output, Shape is N x D

The mean/std are not computed based on the batch. Instead, a single fixed empirical mean of activations during training is used.

Underfit/Overfit

$Err(x) = Bias^2 + variance + irreducible\ error$



https://towardsdatascience.com/regularization-techniques-for-neural-networks-e55f295f2866

- ✓ **W**hen you attend a deep learning lecture, you often hear the term overfitting and underfitting. These terms describe the state of the accuracy of the deep learning model.
- ✓ In underfit models, the training and validation accuracy are both very low. Underfitting is generally a result of not having enough data, need data augmentation. This is a problem of high bias.
- ✓ Overfitting refers to a model that learns the training data too well. It literally mugged up the training data. In overfit models, the training accuracy is very high and validation accuracy is very low. This is a issue with high variance
- ✓ Methods to avoid ovefitting : L2 regularization, Early stopping, Bagging, Dropout

Bias/Variance

- ✓ Bias is the difference between the average prediction of our model and the correct value which we are trying to predict. Model with high bias pays very little attention to the training data and oversimplifies the model. It always leads to high error on training and test data. High bias implies the model is unable to bend to fit the data.
- ✓ Variance is the variability of model prediction for a given data point or a value which tells us spread of our data. Model with high variance pays a lot of attention to training data and does not generalize on the data which it hasn't seen before. As a result, such models perform very well on training data but has high error rates on test data. High variance implies the model follows the data too closely and fails to predict the average.

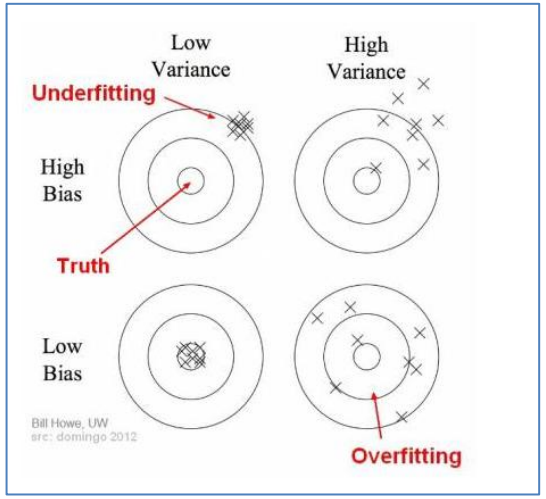
Let the variable we are trying to predict as Y and other covariates as X. We assume there is a relationship between the two such that,

$$Y=f(X) + e$$

where e is the error term and it's normally distributed with a mean of 0. Using linear regression, the expected square error is

$$Err(x) = \left(E[\hat{f}(x)] - f(x)\right)^2 + E\left[\left(\hat{f}(x) - E[\hat{f}(x)]\right)^2\right] + \sigma_e^2$$

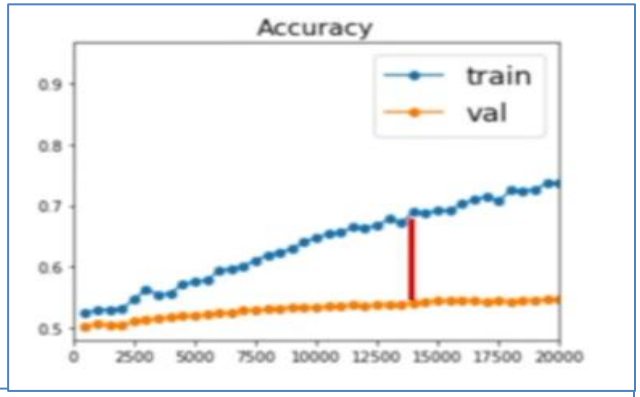
$$Err(x) = \text{Bias}^2 + \text{Variance} + \text{Irreducible Error}$$



https://towardsdatascience.com/understanding-the-bias-variance-tradeoff-165e6942b229

Regularization

$$\tilde{J}(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) + \alpha\Omega(\boldsymbol{\theta})$$



https://www.youtube.com/watch?v=KvtGD37Rm5I
https://towardsdatascience.com/regularization-techniques-for-neural-networks-e55f295f2866

- ✓ Regularization can be defined as any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error.
- ✓ Regularization is often done by putting some extra constraints on a machine learning model, such as adding restrictions on the parameter values or by adding extra terms in the objective function that can be thought of as corresponding to a soft constraint on the parameter values.
- ✓ An effective regularizer is said to be the one that makes a profitable trade by reducing variance significantly while not overly increasing the bias.
- ✓ Where alpha (lamda) is a hyperparameter that weighs the relative contribution of the norm penalty omega. Setting alpha to 0 means no regularization and larger values of alpha correspond to more regularization.
- ✓ For Neural networks, regularization applies mainly to penalizes on the weights of the affine transformations and leave biases untouched. This is because of the fact that biases require lesser data to fit accurately than the weights.
- ✓ In general add some kind of randomness at training time and average it out at test time. CFD : Artificial viscosity, diffusion, penalty

L2 Regularization

$$\tilde{J}(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) + \alpha\Omega(\boldsymbol{\theta})$$

$$\Omega(\boldsymbol{\theta}) = \frac{1}{2}\|\boldsymbol{w}\|_2^2$$

$$J(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda\sum_{j=1}^{n}\theta_j^2\right]$$

$$\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \frac{\alpha}{2}\boldsymbol{w}^\top\boldsymbol{w} + J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}),$$

```
# Loss function using L2 Regularization
regularizer = tf.nn.l2_loss(weights)
loss = tf.reduce_mean(loss + beta * regularizer)
```

$$\nabla_{\boldsymbol{w}}\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \alpha\boldsymbol{w} + \nabla_{\boldsymbol{w}}J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}).$$

$$\boldsymbol{w} \leftarrow (1 - \epsilon\alpha)\boldsymbol{w} - \epsilon\nabla_{\boldsymbol{w}}J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}).$$

We can see that the weight decay term is now multiplicatively shrinking the weight vector by a constant factor on each step, before performing the usual gradient update.

There are several ways of controlling the capacity of Neural Networks to prevent overfitting:

- ✓ **L2 regularization** uses the squared magnitude of all parameters to penalize directly in the objective.
- ✓ The L2 regularization has the intuitive interpretation of heavily penalizing peaky weight vectors and preferring diffuse weight vectors.
- ✓ due to multiplicative interactions between weights and inputs the L2 regularization has the appealing property of encouraging the network to use all of its inputs a little rather than some of its inputs a lot.
- ✓ Lastly, notice that during gradient descent parameter update, using the L2 regularization ultimately means that every weight is decayed linearly: W += -lambda * W towards zero.

L1 and Max Norm Regularization

$$\Omega(\boldsymbol{\theta}) = ||\boldsymbol{w}||_1 = \sum_i |w_i|,$$

$$\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \alpha||\boldsymbol{w}||_1 + J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}),$$

The gradient is scaled by the constant factor with a sign equal to sign(wi).

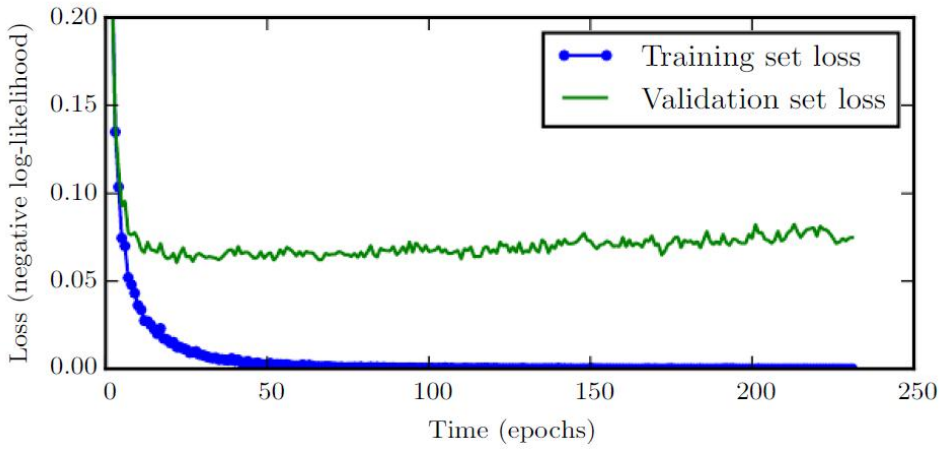
$$\nabla_{\boldsymbol{w}} \tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \alpha \text{sign}(\boldsymbol{w}) + \nabla_{\boldsymbol{w}} J(\boldsymbol{X}, \boldsymbol{y}; \boldsymbol{w})$$

- ✓ **In L1 regularization**, each weight w is added the term $\lambda|w|$ to the objective.
- ✓ The L1 regularization leads the weight vectors to become sparse during optimization (a lot of zero weights). In other words, neurons with L1 regularization end up using only a sparse subset of their most important inputs and become nearly invariant to the "noisy" inputs.
- ✓ Combining the L1 regularization with the L2 regularization is called Elastic net regularization.
- ✓ In comparison, final weight vectors from L2 regularization are usually diffuse. In practice, if you are not concerned with explicit feature selection, L2 regularization can be expected to give superior performance over L1

Max norm constraints.
Another form of regularization is to enforce an absolute upper bound on the magnitude of the weight vector for every neuron and use projected gradient descent to enforce the constraint. In practice, this corresponds to performing the parameter update as normal, and then enforcing the constraint by clamping the weight vector $\vec{w}$ of every neuron to satisfy $\|\vec{w}\|_2 < c$. Typical values of c are on orders of 3 or 4. One of its appealing properties is that network cannot "explode" even when the learning rates are set too high because the updates are always bounded.

Regularization : Early Stopping

When training a large model on a sufficiently large dataset, if the training is done for a long amount of time rather than increasing the generalization capability of the model, it increases the overfitting. As in the training process, the training error keeps on reducing but after a certain point, the validation error starts to increase hence signifying that our model has started to overfit.



This requires babysitting the training and in general is a technique in hyperparameter tuning

One way to think of early stopping is as a very efficient hyperparameter selection algorithm. The idea of early stopping of training is that as soon as the validation error starts to increase we freeze the parameters and stop the training process. Or we can also store the copy of model parameters every time the error on the validation set improves and return these parameters when the training terminates rather than the latest parameters.

Ensemble Learning : Bagging

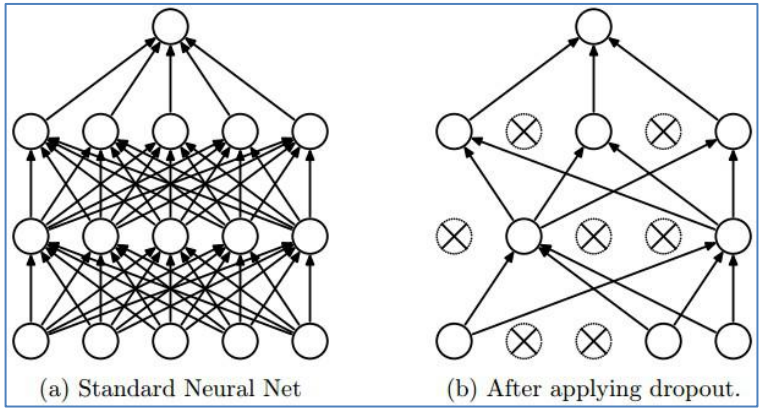
- ✓ In machine learning theory, it is a well-known fact that most models underperform when exposed to datasets that deviate from the training data. From that perspective, combining different models on an ensemble is an efficient way to expand the capacity of a model and reduce its generalization error, leading to methods of regularization.
- ✓ A solution to mitigate the high variance of neural networks is to train multiple models and combine their predictions.
- ✓ Generally, ensemble learning involves training more than one network on the same dataset, then using each of the trained models to make a prediction before combining the predictions in some way to make a final outcome or prediction.
- ✓ **Or one popular way is called Bootstrap Aggregation or Bagging**
  a) It involves resampling the training dataset with replacement, then training a network using the resampled dataset. The resampling procedure means that the composition of each training dataset is different with the possibility of duplicated examples allowing the model trained on the dataset to have a slightly different expectation of the density of the samples, and in turn different generalization error.
  b) The essence of Bagging is to train a deep learning model using variations of the training dataset. Each variation is built by sampling subsets of the original training dataset and replacing some of the missing entries with duplicates from the original selection. That translates into a collection of datasets that are based on entries from the training data replicated multiple times. Let's use an example and assume that our original dataset is the vector [1,2,3,4,5]. A Bagging technique will produce datasets such as [1,1,2,3,5], [1,2,2,3,4], [1,2,3,5,5]… and many others. Bagging will use the generated datasets to train the original model and combine the results. Different trainings should specialize the model on different areas of the target data.

Regularization : Dropout

- ✓ If Bagging trains the model on variations of the input dataset, Dropout works by generating subnetworks of the original deep neural network. Essentially, Dropout creates subnetworks by removing some of the non-output units and then train those subnetworks using the original dataset. The nature of Dropout enables to create an ensemble capabilities of the original model which can be incredibly efficient
- ✓ Dropout is a computationally inexpensive but powerful regularization method, dropout can be thought of as a method of making bagging practical for ensembles of very many large neural networks. The method of bagging cannot be directly applied to large neural networks as it involves training multiple models, and evaluating multiple models on each test example. since training and evaluating such networks is costly in terms of runtime and memory, this method is impractical for neural networks. Dropout provides an inexpensive approximation to training and evaluating a bagged ensemble of exponentially many neural networks. Dropout trains the ensemble consisting of all sub-networks that can be formed by removing non-output units from an underlying base network.
- ✓ In most modern neural networks, based on a series of affine transformations and nonlinearities, we can effectively remove a unit from a network by multiplying its output value by zero. The dropout algorithm in terms of multiplication by zero for simplicity, but it can be trivially modified to work with other operations that remove a unit from the network.
- ✓ Dropout falls into a more general category of methods that introduce stochastic behavior in the forward pass of the network.

Dropout

Dropout is an extremely effective and simple regularization technique by Srivastava et al. It complements the other methods (L1, L2, maxnorm). While training, dropout is implemented by only keeping a neuron active with some probability p or setting it to zero otherwise. In each forward pass, randomly set some neurons (activation) to zero. The probability p of dropout is a hyperparameter; 0.5 is common.



(a) Standard Neural Net (b) After applying dropout.

During training, Dropout can be interpreted as sampling a Neural Network within the full Neural Network, and only updating the parameters of the sampled network based on the input data. During testing there is no dropout applied, with the interpretation of evaluating an averaged prediction across the exponentially-sized ensemble of all sub-networks.

https://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf

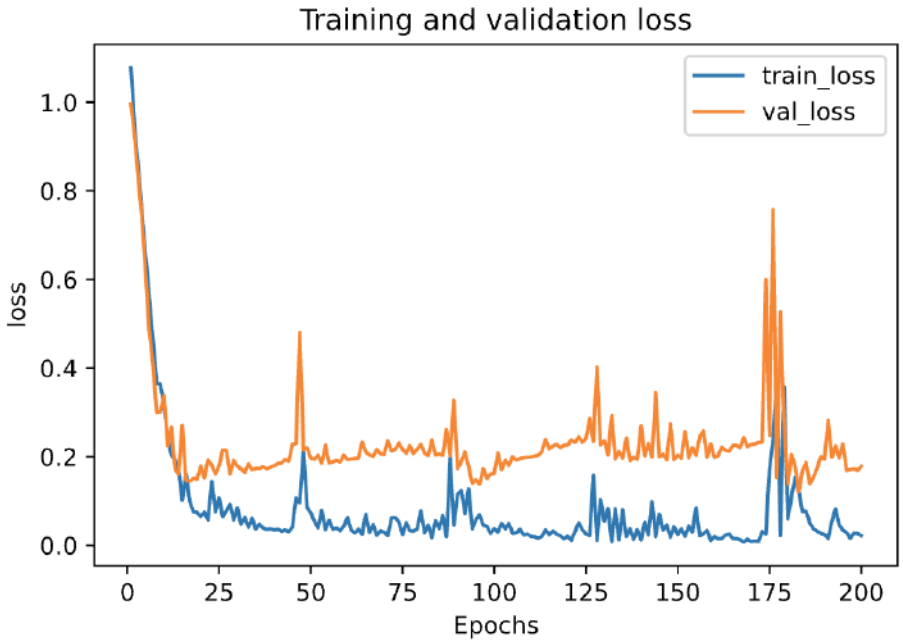
```
def predict(X):
  # ensembled forward pass
  H1 = np.maximum(0, np.dot(W1, X) + b1) * p # NOTE: scale the activations
  H2 = np.maximum(0, np.dot(W2, H1) + b2) * p # NOTE: scale the activations
  out = np.dot(W3, H2) + b3
```

```
"""
Inverted Dropout: Recommended implementation example.
We drop and scale at train time and don't do anything at test time.
"""

p = 0.5 # probability of keeping a unit active. higher = less dropout

def train_step(X):
  # forward pass for example 3-layer neural network
  H1 = np.maximum(0, np.dot(W1, X) + b1)
  U1 = (np.random.rand(*H1.shape) < p) / p # first dropout mask. Notice /p!
  H1 *= U1 # drop!
  H2 = np.maximum(0, np.dot(W2, H1) + b2)
  U2 = (np.random.rand(*H2.shape) < p) / p # second dropout mask. Notice /p
  H2 *= U2 # drop!
  out = np.dot(W3, H2) + b3

  # backward pass: compute gradients... (not shown)
  # perform parameter update... (not shown)

def predict(X):
  # ensembled forward pass
  H1 = np.maximum(0, np.dot(W1, X) + b1) # no scaling necessary
  H2 = np.maximum(0, np.dot(W2, H1) + b2)
  out = np.dot(W3, H2) + b3
```

Regularization - TensorFlow

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
def create_model():
        model = Sequential([
                Dense(64, activation='relu', input_shape=(4,)),
                Dense(128, activation='relu'),
                Dense(128, activation='relu'),
                Dense(128, activation='relu'),
                Dense(64, activation='relu'),
                Dense(64, activation='relu'),
                Dense(64, activation='relu'),
                Dense(3, activation='softmax')
        ])
        return model

model = create_model()
model.summary()

model.compile(
        optimizer='adam',
        loss='categorical_crossentropy',
        metrics=['accuracy']
)

history = model.fit(
X_train,
y_train,
epochs=200,
validation_split=0.25,
batch_size=40,
verbose=2
)
plot_metric(history, 'loss')
```

```
%matplotlib inline
%config InlineBackend.figure_format = 'svg'

def plot_metric(history, metric):
        train_metrics = history.history[metric]
        val_metrics = history.history['val_'+metric]
        epochs = range(1, len(train_metrics) + 1)
        plt.plot(epochs, train_metrics)
        plt.plot(epochs, val_metrics)
        plt.title('Training and validation '+ metric)
        plt.xlabel("Epochs")
        plt.ylabel(metric)
        plt.legend(["train_"+metric, 'val_'+metric])
        plt.show()
```



Regularization - TensorFlow

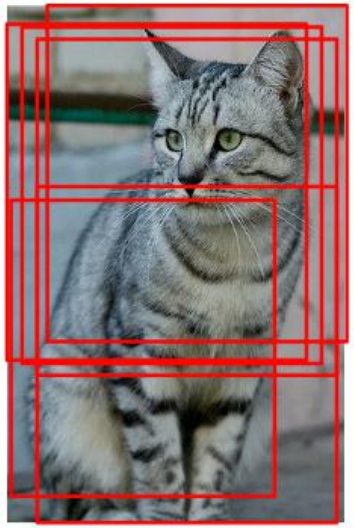
```
from tensorflow.keras.layers import Dropout
from tensorflow.keras.regularizers import l2
def create_regularized_model(factor, rate):
model = Sequential([
        Dense(64, kernel_regularizer=l2(factor), activation="relu", input_shape=(4,)), Dropout(rate),
        Dense(128, kernel_regularizer=l2(factor), activation="relu"), Dropout(rate),
        Dense(128, kernel_regularizer=l2(factor), activation="relu"), Dropout(rate),
        Dense(128, kernel_regularizer=l2(factor), activation="relu"), Dropout(rate),
        Dense(64, kernel_regularizer=l2(factor), activation="relu"), Dropout(rate),
        Dense(64, kernel_regularizer=l2(factor), activation="relu"), Dropout(rate),
        Dense(64, kernel_regularizer=l2(factor), activation="relu"), Dropout(rate),
        Dense(3, activation='softmax')
])
return model
```

```
model = create_regularized_model(1e-5, 0.3)
model.summary()
# First configure model using model.compile()
model.compile(
        optimizer='adam',
        loss='categorical_crossentropy',
        metrics=['accuracy']
)# Then, train the model with fit()
history = model.fit(
        X_train,
        y_train,
        epochs=200,
        validation_split=0.25,
        batch_size=40,
        verbose=2
)
plot_metric(history, 'loss')
```

https://towardsdatascience.com/machine-learning-model-regularization-in-practice-an-example-with-keras-and-tensorflow-2-0-52a96746123e

Regularization : Data Augmentation



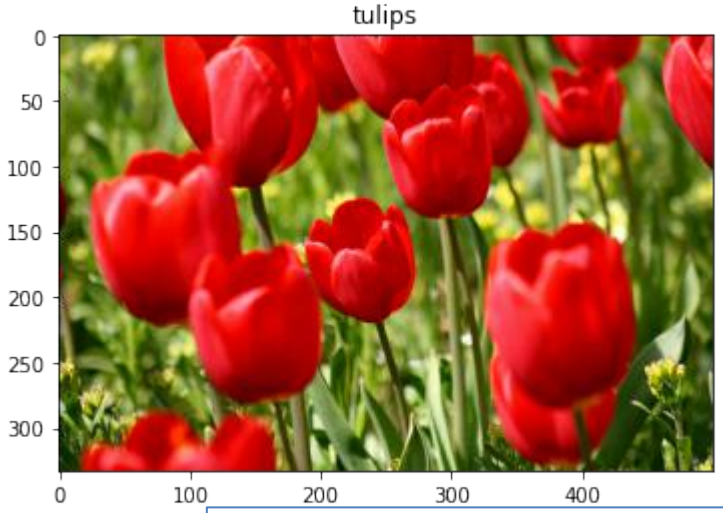



Horizontal Flips

contrast and brightness

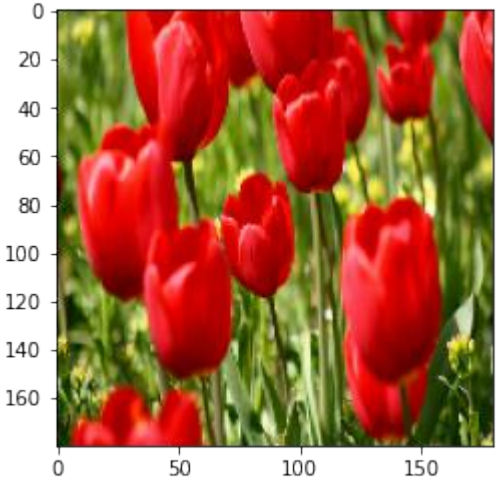
Random crops and scales

Random mix of : translation, rotation, stretching, shearing, lens distortions, etc..



TensorFlow/Keras code

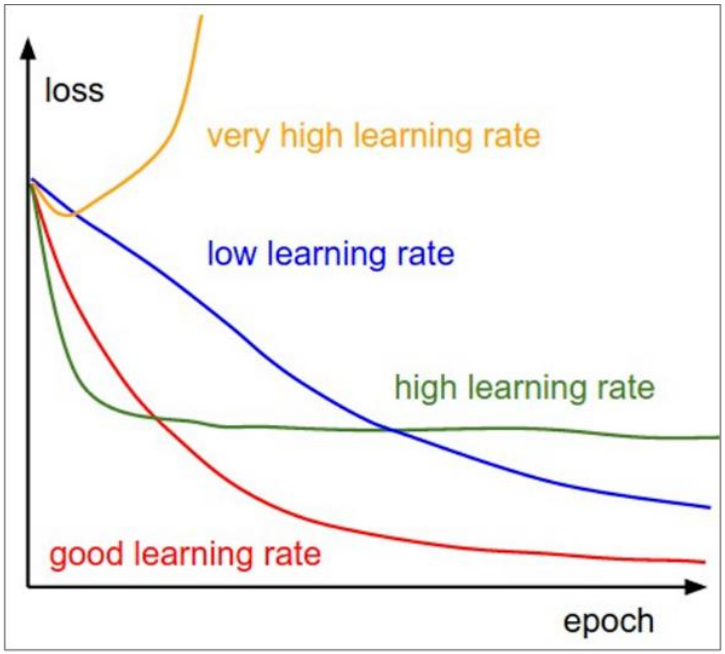
```
resize_and_rescale =
tf.keras.Sequential([
  layers.experimental.preprocessing.Resizing(IMG_SIZE, IMG_SIZE),
  layers.experimental.preprocessing.Rescaling(1./255)
])
```

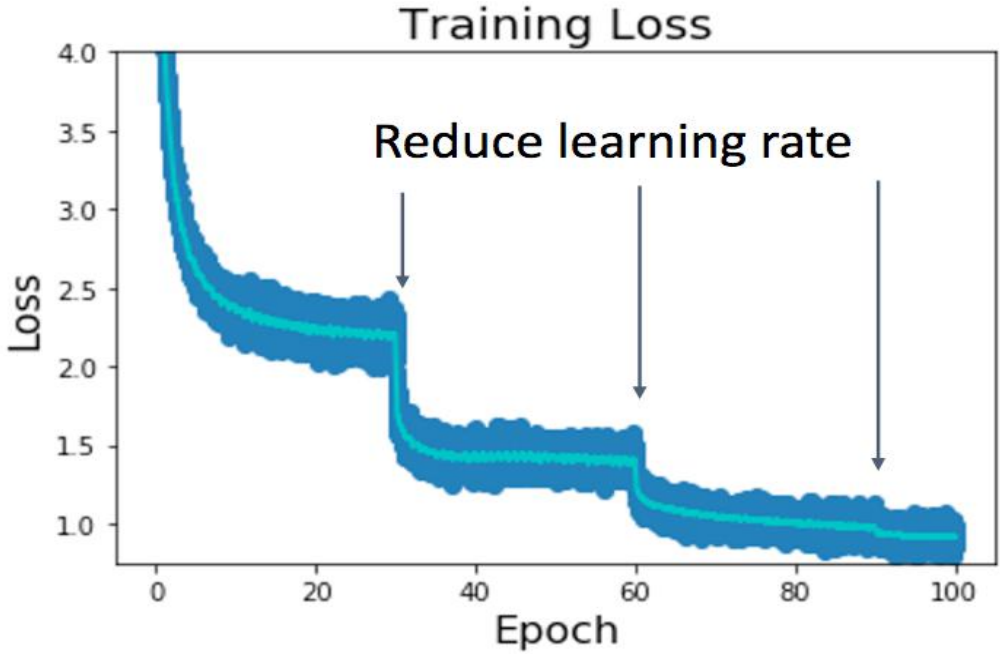


There are a variety of preprocessing layers you can use for data augmentation including layers.RandomContrast, layers.RandomCrop, layers.RandomZoom, and others.

Hyperparameter

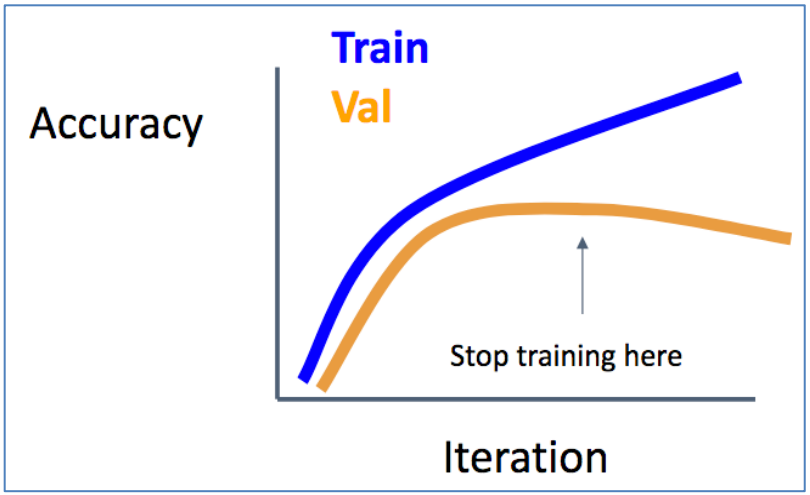
- ✓ **Hyperparameter** are neural network "presets" like network architecture, learning rate, batch size, and more. A poor choice of hyperparameters can cause a network's accuracy to converge slowly or not at all. Tuning learning rate is important.
- ✓ **What are some standard hyperparameter optimization techniques?**
  - Selection: Grid Search, Random Search
  - Early stopping : Learning Curve Matching, (LCM), Successive Halving Algorithm (SHA) and Hyperband
  - Modern Approaches: Evolutionary Algorithms, such as population based training (PBT) and swarm optimization.

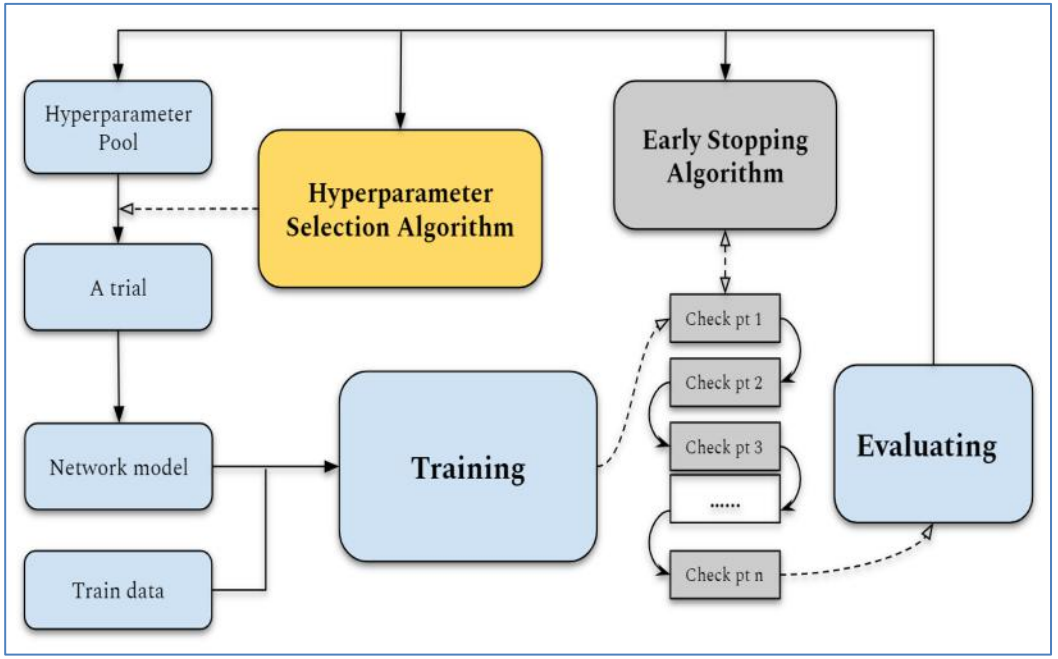




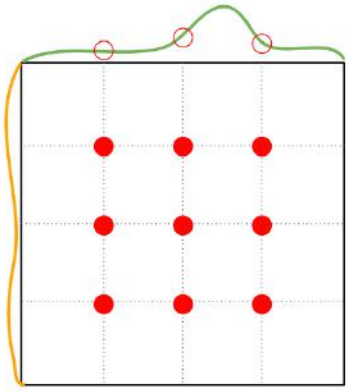
Step: Reduce learning rate at a few fixed points. E.g. for ResNets, multiply LR by 0.1 after epochs 30, 60, and 90.

Early Stopping





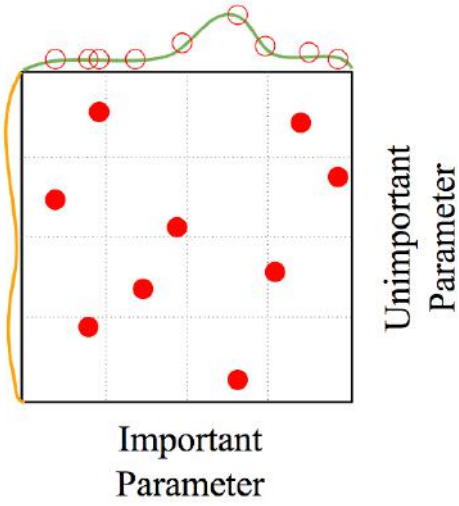
Grid Layout



Choose several values for each hyperparameter (Often space choices log-linearly) Example:
Weight decay: [$1x10^{-4}$, $1x10^{-3}$, $1x10^{-2}$, $1x10^{-1}$]
Learning rate: [$1x10^{-4}$, $1x10^{-3}$, $1x10^{-2}$, $1x10^{-1}$]
Evaluate all possible choices on this hyperparameter grid

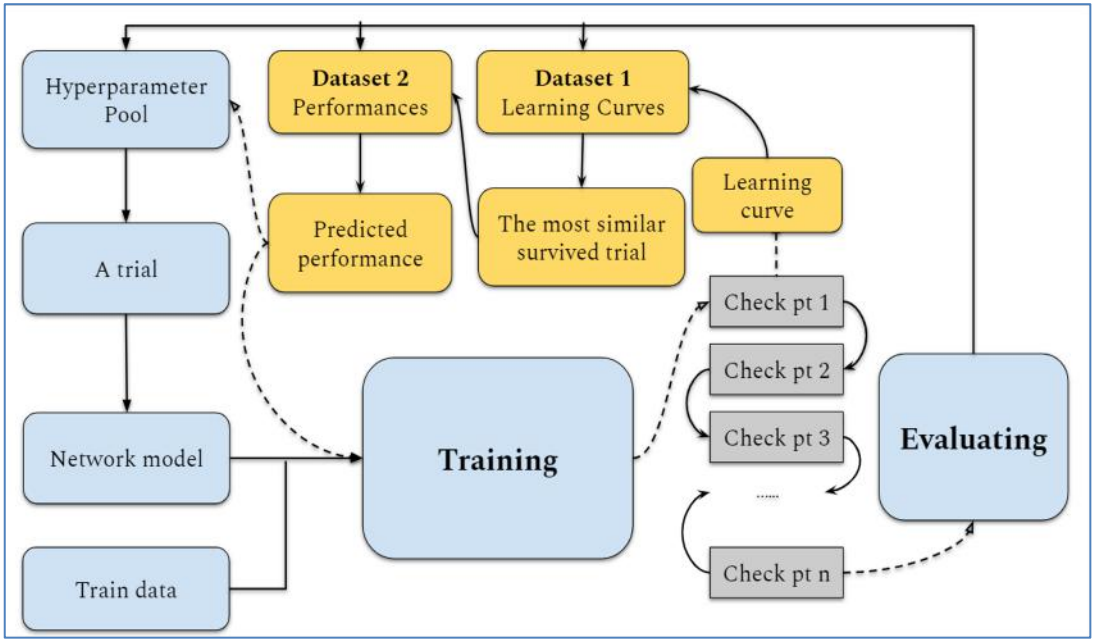
Choose several values for each hyperparameter (Often space choices log-linearly) Example:
Weight decay: log-uniform on [$1x10^{-4}$, $1x10^{-1}$]
Learning rate: log-uniform on [$1x10^{-4}$, $1x10^{-1}$]
Run many different random trials

Random Layout



Learning Curves matching LCM

- ✓ ***Trials***: Sets contain a single sample for every hyperparameter.
- ✓ ***Learning Curves***: arrays of the numerical values of the loss function during certain stages of a single training.
- ✓ ***Check Points***: points where LCM is applied to decide whether abort the training



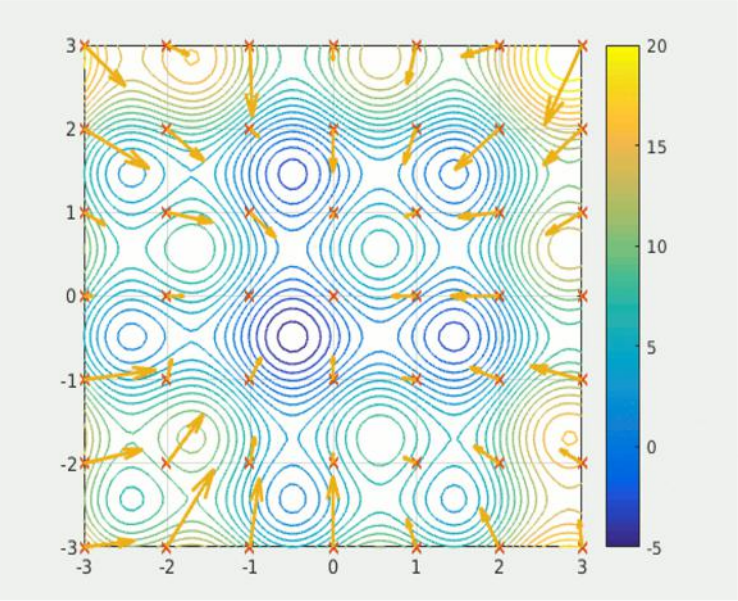
Given a fixed number of trials, we compared two algorithms' computing time and best performances. The same experiments are repeated 12 times.

Name	Trials	Computer Time (S)	Best Performance (%)
LCM	100	8069.08	67.05
Random	100	26498.00	67.26

Remark: in 7 of 12 experiments, two algorithms got the same optimal hyperparameters. CIFAR10 test

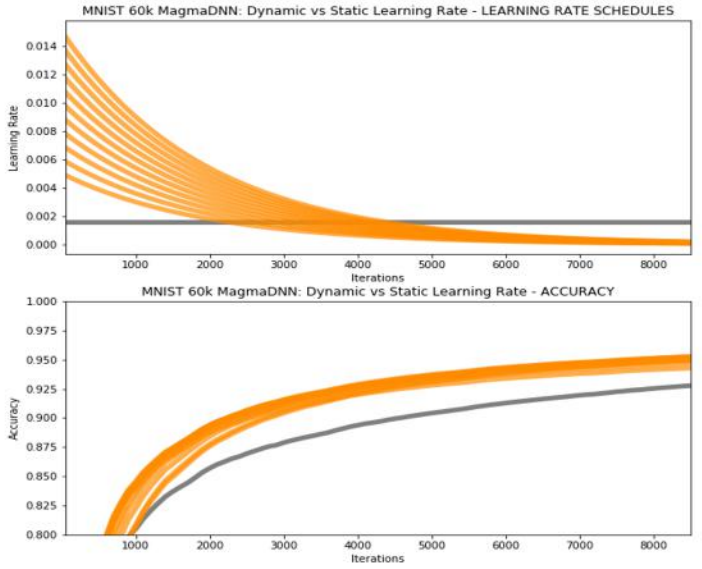
Population Based Training with MagmaDNN

- ✓ **Population Based Training (PBT)** is an evolutionary hyperparameter optimization algorithm.
- ✓ **Evolutionary optimization algorithms** use natural models to inspire a particular approach to traversing a search space to find the minimum of an objective function. One classic case is the Particle Swarm Optimization algorithm, inspired by the swarming behavior of bees.



How does the PBT Algorithm work?

- ✓ Population Model
- ✓ Stochasticity
- ✓ Exploit / Explore
- ✓ Early Stopping
- ✓ Evolution
- ✓ Adaptive Hyperparameter
- ✓ Scheduling

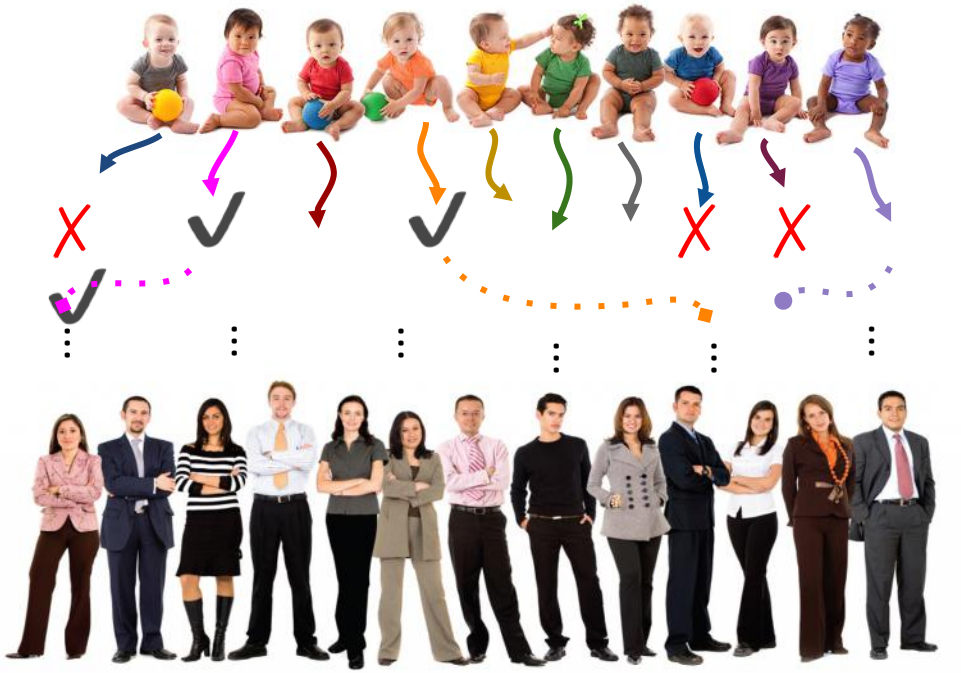


Joint Institute for Computational Sciences

INNOVATIVE COMPUTING LABORATORY

PBT

1) Initialize Networks: Random Weights, Random Hyperparameter
2) Training Period : Networks optimize weights in the usual way (SGD, ADAM, etc.)
3) Rank Fitness : accuracy, loss or other measure determines most and least fit
4) Exploit : Copy the weights and hyperparameters from the most fit to the least
5) Explore : Perturb the updated hyperparameters
6) Repeat : Train -> Exploit -> Explore process until desired convergence
7) End Result: Optimized networks with optimized hyperparameter schedules



	Grid Search	Random Search	PBT
Parallelizability	✓	✓	✓
Stochasticity	X	✓	✓
Early Stopping	X	X	✓
Adaptive Hyperparameters	X	X	✓

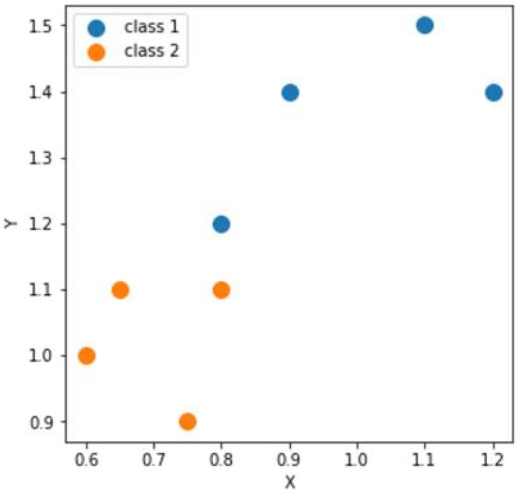
Topics of Unit 11

- ✓ **UNIT 11 : DNN Computing**
  - **DNN Principles: Mathematics**
  - **CNN Model: ResNet**
  - **DNN Computing: Essential Practices**
  - **DNN Usage: I/O, Transfer Learning**
  - **CNN Applications: Unet, Object Detection, Autonomous Vehicle**
  - **Advanced Topics: GAN, Reinforcement Learning**

Building your own Dataset

TensorFlow take data in the format of '*x_train*' and '*y_train*'.
'*x_train*' - features in the dataset
y_train **- contains the classification or labels for those features**

X	Y	Class
0.8	1.2	1
0.9	1.4	1
1.2	1.4	1
1.1	1.5	1
0.8	1.1	2
0.6	1	2
0.65	1.1	2
0.75	0.9	2




```
In:     x_train = []
        x_train.append([0.80, 1.2])
        x_train.append([0.90, 1.4])
        x_train.append([1.20, 1.4])
        x_train.append([1.10, 1.5])
        x_train.append([0.80, 1.1])
        x_train.append([0.60, 1.0])
        x_train.append([0.65, 1.1])
        x_train.append([0.75, 0.9])
        X_train
Out:        [[0.8, 1.2],
         [0.9, 1.4],
         [1.2, 1.4],
         [1.1, 1.5],
         [0.8, 1.1],
         [0.6, 1.0],
         [0.65, 1.1],
         [0.75, 0.9]]
In:     x_train.shape
Out:    (8,2)
```

```
In:     y_train = []
        y_train.append([1])
        y_train.append([1])
        y_train.append([1])
        y_train.append([1])
        y_train.append([2])
        y_train.append([2])
        y_train.append([2])
        y_train.append([2])
        Y_train

Out:    [[1], [1], [1],
[1], [2], [2], [2], [2]]

In:     y_train.shape

Out:    (8,)
```

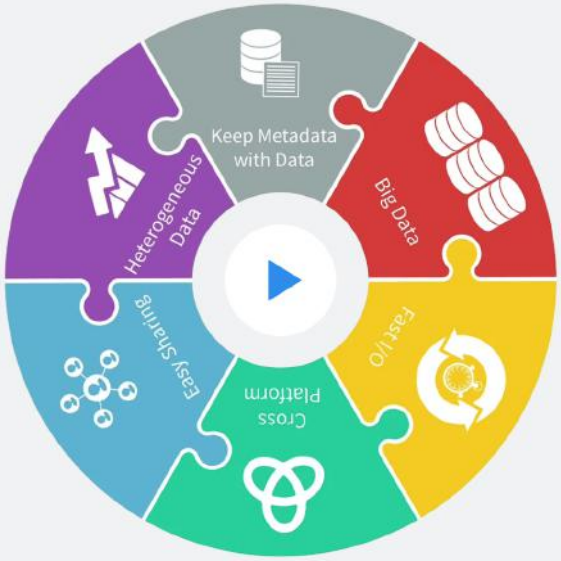
https://www.hdfgroup.org/solutions/hdf5/

The HDF Group

About Us | Solutions | Community | Downloads | Documentation | Support Portal | Register | HDF Lab

THE HDF5® LIBRARY & FILE FORMAT

HDF5®



High-performance data management and storage suite

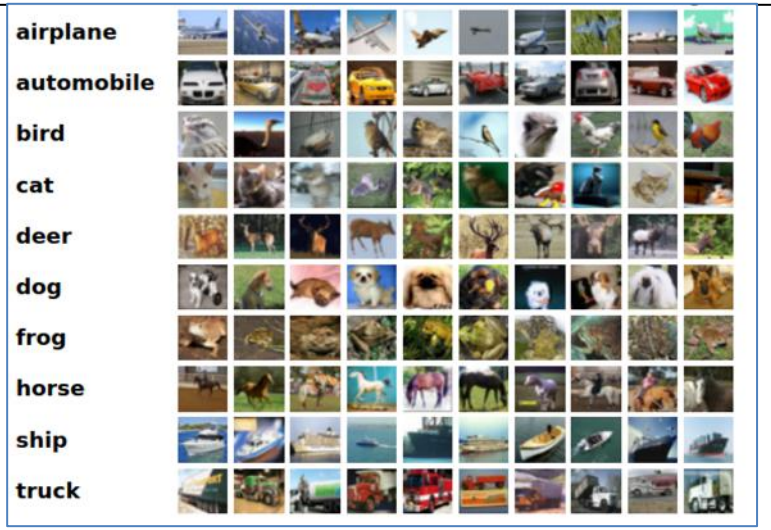
Utilize the HDF5 high performance data software library and file format to manage, proc
fast I/O processing and storage.

Download HDF5 | Download Sell Sheet (PDF) | Documentation

Accessing Popular Datasets

```
import tensorflow as tf
import matplotlib.pyplot as plt

cifar10 = tf.keras.datasets.cifar10 #download the images

tf.keras.datasets.
                    {} boston_housing
(x_train, y_train)  {} cifar10
                    {} cifar100
print("number of i {} fashion_mnist
image_classes = [" {} imdb
                    {} mnist
model = tf.keras.m  {} reuters
```

Easy access to 7 datasets can be made directly through TensorFlow without needing to install extra modules. Datasets will be automatically downloaded and saved for future use.




```
import tensorflow as tf
cifar10 = tf.keras.datasets.cifar10 #download the images
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
```

boston_housing module: Boston housing price regression dataset.

cifar10 module: CIFAR10 small images classification dataset.

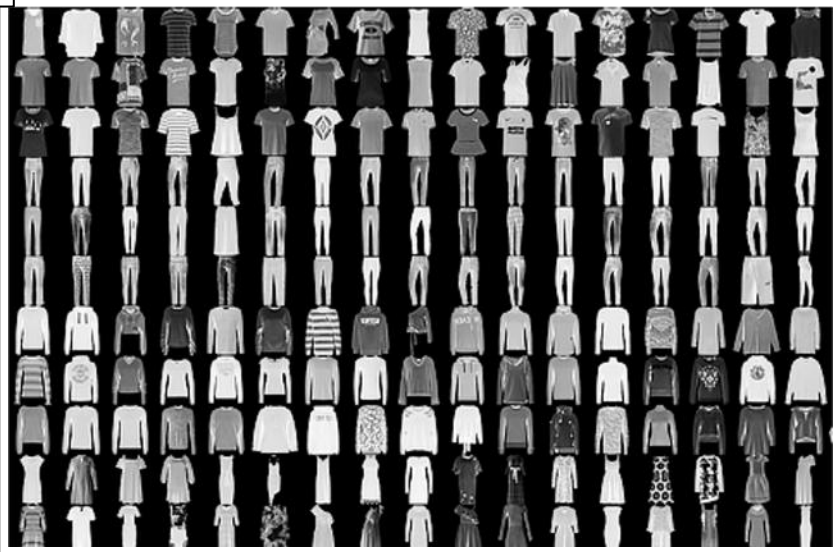
cifar100 module: CIFAR100 small images classification dataset.

fashion_mnist module: Fashion-MNIST dataset.

imdb module: IMDB sentiment classification dataset.

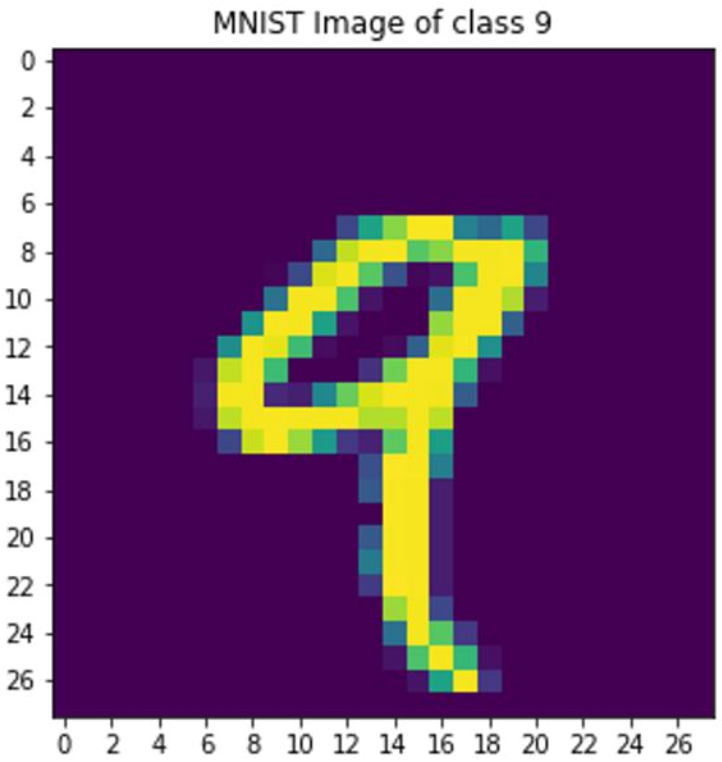
mnist module: MNIST handwritten digits dataset.

reuters module: Reuters topic classification dataset.



Investigating MNIST

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
mnist = tf.keras.datasets.mnist #download images
(x_train, y_train), (x_test, y_test) =
mnist.load_data()
img_idx = 4 #the image index to use as an example

plt.imshow(x_train[img_idx]) #plot the image

plt.xticks(np.arange(0,len(x_train[img_idx]),3))
plt.yticks(np.arange(0,len(x_train[img_idx]),3))
plt.title("MNIST Image of class
{}".format(y_train[img_idx]))
plt.show()
```



Accessing label data for plot title

Accessing Other Datasets

Access to 150+ datasets can be gained through the use of the “TensorFlow Datasets” Python module. Datasets will be downloaded from the database.

```
import tensorflow as tf
import tensorflow_datasets as tfds

# Construct a tf.data.Dataset
ds = tfds.load('mnist', split='train',
shuffle_files=True)

# Build your input pipeline
ds = ds.shuffle(1024).batch(32)
for example in ds.take(2):
 image, label = example["image"], example["label"]
 print(image.shape)
```

Above code prints:

(32, 28, 28, 1)

(32, 28, 28, 1)

The dimensions correspond to:

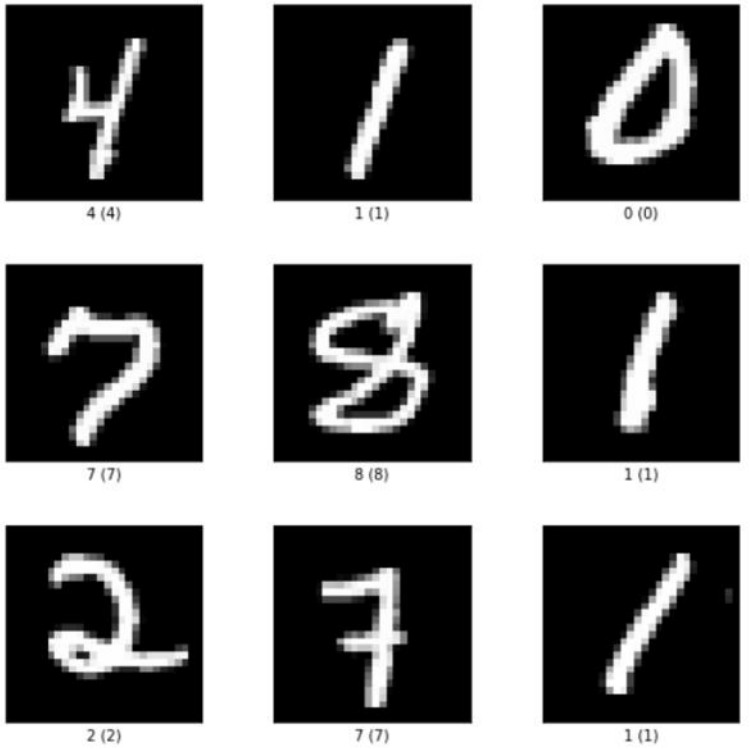
(batch_size,image_height,image_width,image_channels)

conda install tensorflow-datasets

Datasets are stored in:

~/tensorflow_datasets/

```
ds, info = tfds.load('mnist', split='train', with_info=True)

fig = tfds.show_examples(ds, info)
```



Accessing ImageNet Dataset

The ImageNet dataset can be used through tensorflow-datasets. However, you need to manually download it through image-net.org after receiving access.

After getting access, the two files: **ILSVRC2012_img_train.tar** and **ILSVRC2012_img_val.tar** must be downloaded and placed in the following directory:

~/tensorflow_datasets/downloads/manual/

1. Download the 2012 test split available here.
2. Download the October 10, 2019 patch. There is a Google Drive link to the patch provided on the same page.
3. Combine the two tar-balls, manually overwriting any images in the original archive with images from the patch. According to the instructions on image-net.org, this procedure overwrites just a few images.

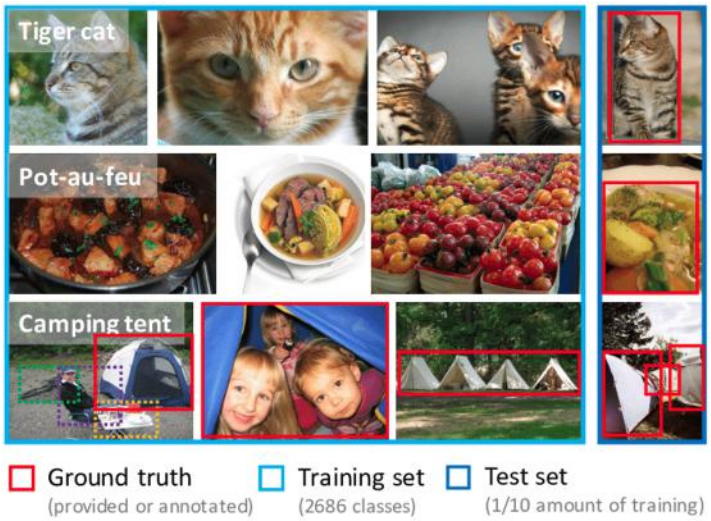
```
import tensorflow as tf
import tensorflow_datasets as tfds

# Construct a tf.data.Dataset
ds = tfds.load('imagenet2012', split='train',
shuffle_files=True)
```

conda install tensorflow-datasets

IMAGENET

- **Dataset size: 155.84 GiB**
- **Classes: 1000**

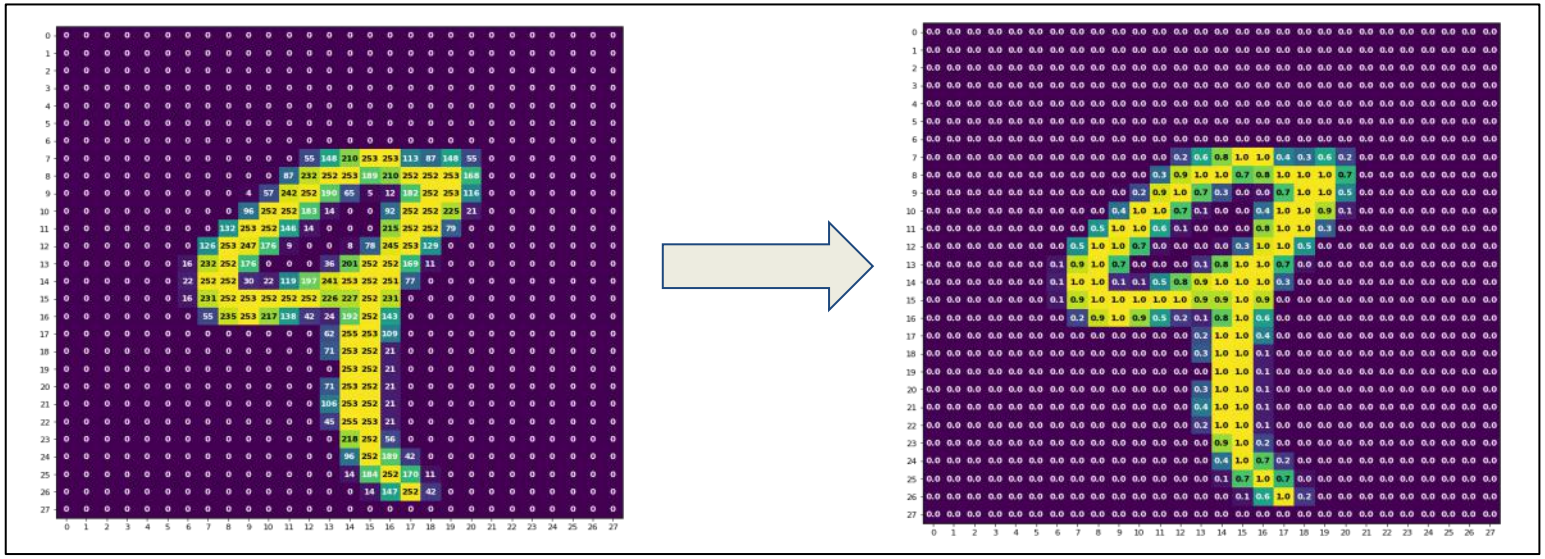
Split	# of Images
'Test'	100,000
'Train'	1,281,167
'Validation'	50,000



Using TensorFlow Datasets Map

```
def scale(image, label):
      image = tf.cast(image, tf.float32)
      image /= 255
      return image, label
BATCH_SIZE = 64
datasets = tfds.load("mnist")
ds_train, ds_test = datasets['train'],
datasets['test']
train_dataset = ds_train.map(scale).batch(BATCH_SIZE)
eval_dataset = ds_test.map(scale).batch(BATCH_SIZE)
```

Map is passed a function which will do data preprocessing such as resizing, casting, and normalization.

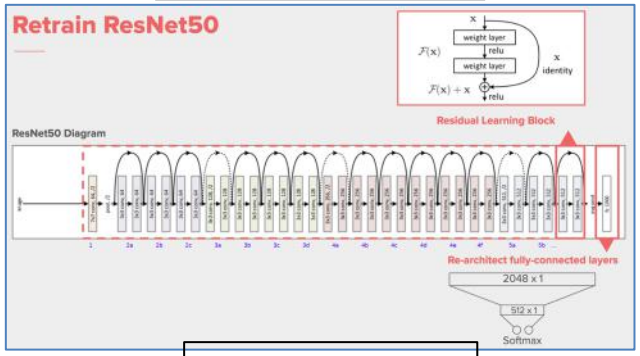


Neural Network Models

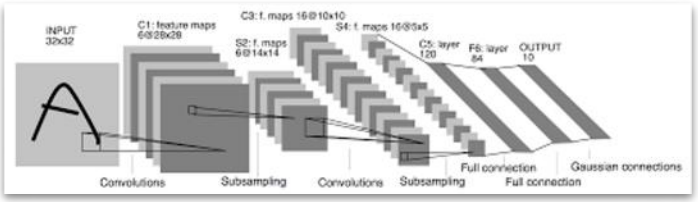
Dataset → Pre-defined Neural Network Model → Fully connected layer # neurons = # classes → Classification

Examples of Pre-defined Neural Network Models

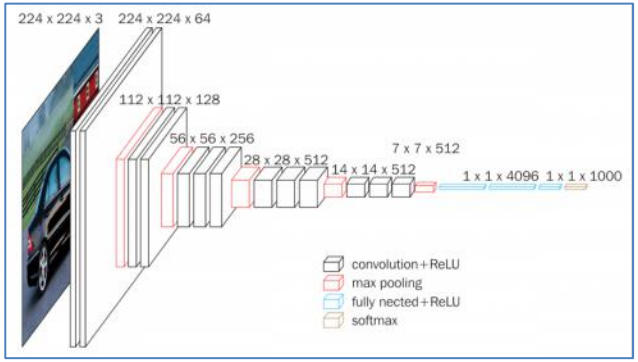
ResNet50



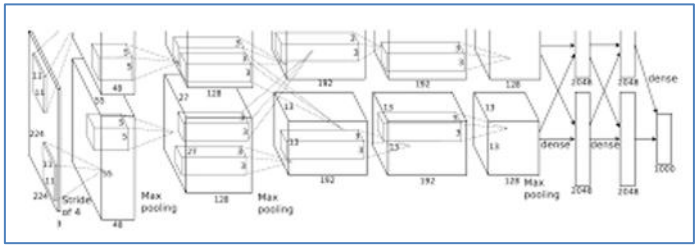
LeNet5



VGG16



AlexNet



Understanding Transfer Learning

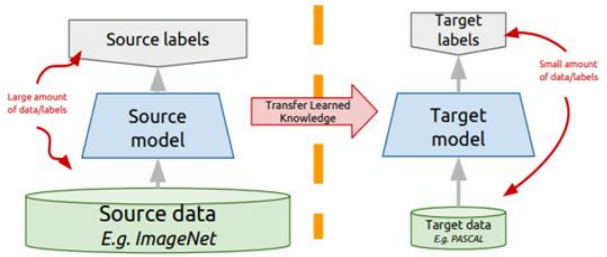
Transfer learning: idea

Instead of training a deep network from scratch for your task:

- Take a network trained on a different domain for a different **source task**
- Adapt it for your domain and your **target task**

Variations:

- Same domain, different task
- Different domain, same task



- ✓ Traditional learning occurs purely based on specific tasks, datasets and training models.
- ✓ Transfer learning is a technique that can leverage knowledge (features, weights etc) from previously trained models for training newer models and tackle problems like having less data for the newer task!
- ✓ Transfer learning utilizes knowledge from previously learned tasks and apply them to newer, related ones.

Transfer learning consists of taking features learned on one problem, and leveraging them on a new, similar problem. Transfer learning is usually done for tasks where the dataset has too little data to train a full-scale model from scratch. In general the workflow of transfer learning in DNN is :

- ✓ Take layers from a previously trained model.
- ✓ Freeze them, so as to avoid destroying any of the information they contain during future training rounds.
- ✓ Add some new, trainable layers on top of the frozen layers. They will learn to turn the old features into predictions on a new dataset.
- ✓ Train the new layers on a new dataset.

A last, optional step, is **fine-tuning**, which consists of unfreezing the entire model or part of it, and re-training it on the new data with a very low learning rate. This can potentially achieve meaningful improvements, by incrementally adapting the pretrained features to the new data.

https://www.tensorflow.org/guide/keras/transfer_learning

Transfer Learning

Freeze or fine-tune?

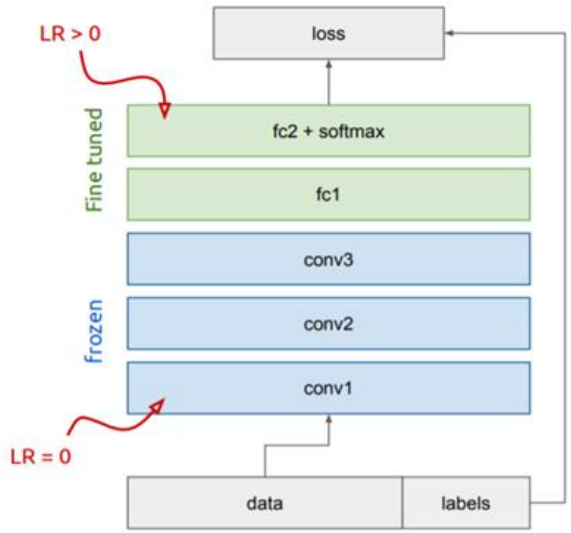
Bottom *n* layers can be frozen or fine tuned.

- **Frozen**: not updated during backprop
- **Fine-tuned**: updated during backprop

Which to do depends on target task:

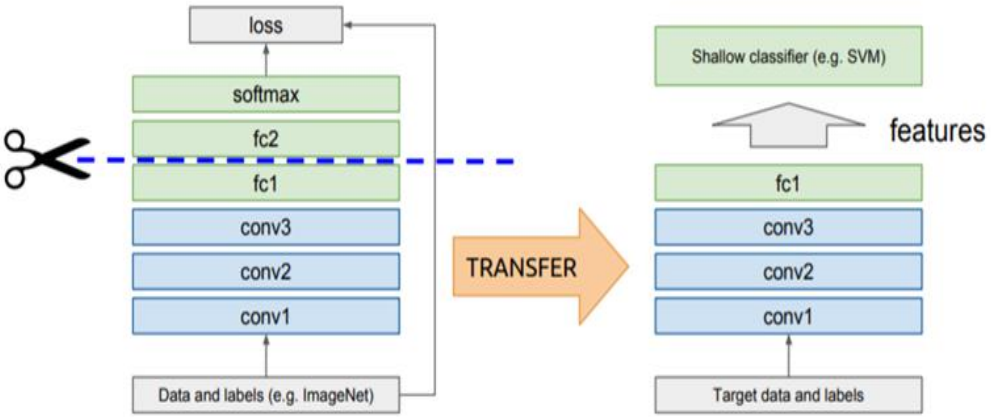
- **Freeze**: target task labels are scarce, and we want to avoid overfitting
- **Fine-tune**: target task labels are more plentiful

In general, we can set learning rates to be different for each layer to find a tradeoff between freezing and fine tuning



Idea: use outputs of one or more layers of a network trained on a different task as generic feature detectors. Train a new shallow model on these features.

Assumes that $D_S = D_T$



- ✓ Find a very large dataset that has similar data, train a big ConvNet
- ✓ Transfer learning parameters to your dataset

A pre-trained model is a saved network that was previously trained on a large dataset, typically on a large-scale image-classification task. You either use the pretrained model as is or use transfer learning to customize this model to a given task.

The intuition behind transfer learning for image classification is that if a model is trained on a large and general enough dataset, this model will effectively serve as a generic model of the visual world. You can then take advantage of these learned feature maps without having to start from scratch.

Understanding Transfer Learning

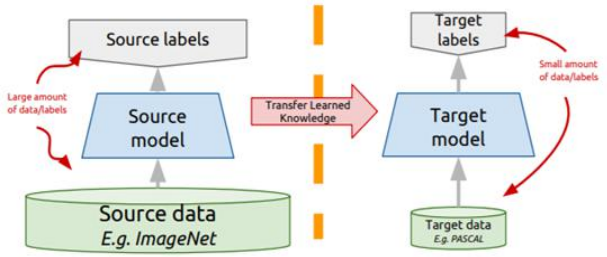
Transfer learning: idea

Instead of training a deep network from scratch for your task:

- Take a network trained on a different domain for a different **source task**
- Adapt it for your domain and your **target task**

Variations:

- Same domain, different task
- Different domain, same task



- ✓ Traditional learning occurs purely based on specific tasks, datasets and training models.
- ✓ Transfer learning is a technique that can leverage knowledge (features, weights etc) from previously trained models for training newer models and tackle problems like having less data for the newer task!
- ✓ Transfer learning utilizes knowledge from previously learned tasks and apply them to newer, related ones.

Transfer learning consists of taking features learned on one problem, and leveraging them on a new, similar problem. Transfer learning is usually done for tasks where the dataset has too little data to train a full-scale model from scratch. In general the workflow of transfer learning in DNN is :

- ✓ Take layers from a previously trained model.
- ✓ Freeze them, so as to avoid destroying any of the information they contain during future training rounds.
- ✓ Add some new, trainable layers on top of the frozen layers. They will learn to turn the old features into predictions on a new dataset.
- ✓ Train the new layers on a new dataset.

A last, optional step, is **fine-tuning**, which consists of unfreezing the entire model or part of it, and re-training it on the new data with a very low learning rate. This can potentially achieve meaningful improvements, by incrementally adapting the pretrained features to the new data.

https://www.tensorflow.org/guide/keras/transfer_learning

Transfer Learning

Freeze or fine-tune?

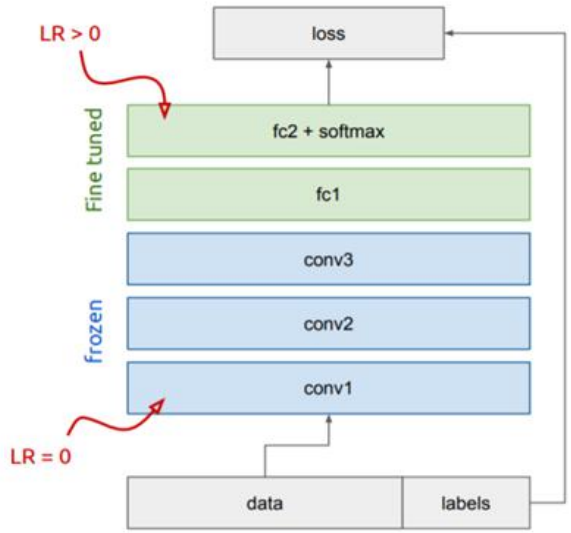
Bottom *n* layers can be frozen or fine tuned.

- **Frozen**: not updated during backprop
- **Fine-tuned**: updated during backprop

Which to do depends on target task:

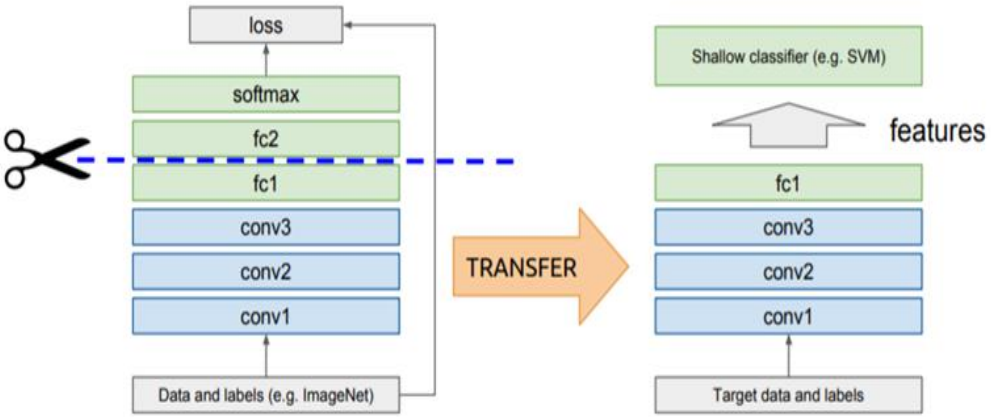
- **Freeze**: target task labels are scarce, and we want to avoid overfitting
- **Fine-tune**: target task labels are more plentiful

In general, we can set learning rates to be different for each layer to find a tradeoff between freezing and fine tuning



Idea: use outputs of one or more layers of a network trained on a different task as generic feature detectors. Train a new shallow model on these features.

Assumes that $D_S = D_T$



- ✓ Find a very large dataset that has similar data, train a big ConvNet
- ✓ Transfer learning parameters to your dataset

A pre-trained model is a saved network that was previously trained on a large dataset, typically on a large-scale image-classification task. You either use the pretrained model as is or use transfer learning to customize this model to a given task.

The intuition behind transfer learning for image classification is that if a model is trained on a large and general enough dataset, this model will effectively serve as a generic model of the visual world. You can then take advantage of these learned feature maps without having to start from scratch.

Example from Keras

- ✓ Layers & models also feature a boolean attribute trainable. Its value can be changed. Setting layer.trainable to False moves all the layer's weights from trainable to non-trainable. This is called "freezing" the layer: the state of a frozen layer won't be updated during training (either when training with fit() or when training with any custom loop that relies on trainable_weights to apply gradient updates).
- ✓ When a trainable weight becomes non-trainable, its value is no longer updated during training.
- ✓ If you set trainable = False on a model or on any layer that has sublayers, all children layers become non-trainable as well.
- ✓ Once your model has converged on the new data, you can try to unfreeze all or part of the base model and retrain the whole model end-to-end with a very low learning rate.

```
# Make a model with 2 layers
layer1 = keras.layers.Dense(3, activation="relu")
layer2 = keras.layers.Dense(3, activation="sigmoid")
model = keras.Sequential([keras.Input(shape=(3,)), layer1, layer2])

# Freeze the first layer
layer1.trainable = False

# Keep a copy of the weights of layer1 for later reference
initial_layer1_weights_values = layer1.get_weights()

# Train the model
model.compile(optimizer="adam", loss="mse")
model.fit(np.random.random((2, 3)), np.random.random((2, 3)))

# Check that the weights of layer1 have not changed during training
final_layer1_weights_values = layer1.get_weights()
np.testing.assert_allclose(
    initial_layer1_weights_values[0], final_layer1_weights_values[0]
)
np.testing.assert_allclose(
    initial_layer1_weights_values[1], final_layer1_weights_values[1]
)
```

https://www.tensorflow.org/guide/keras/transfer_learning

The typical transfer-learning workflow

A typical transfer learning workflow can be implemented in Keras:

1) Instantiate a base model and load pre-trained weights into it.
2) Freeze all layers in the base model by setting trainable = False.
3) Create a new model on top of the output of one (or several) layers from the base model.
4) Train your new model on your new dataset.

Note that an alternative, more lightweight workflow could also be:

1) Instantiate a base model and load pre-trained weights into it.
2) Run your new dataset through it and record the output of one (or several) layers from the base model. This is called **feature extraction**.
3) Use that output as input data for a new, smaller model.

A key advantage of that second workflow is that you only run the base model once on your data, rather than once per epoch of training. So it's a lot faster & cheaper.

```
base_model = keras.applications.Xception(
    weights='imagenet',  # Load weights pre-trained on ImageNet.
    input_shape=(150, 150, 3),
    include_top=False)  # Do not include the ImageNet classifier at the
top.
base_model.trainable = False
inputs = keras.Input(shape=(150, 150, 3))

# make sure that the base_model is running in inference mode here,
# by passing `training=False`. This is important for fine-tuning
x = base_model(inputs, training=False)

# Convert features of shape `base_model.output_shape[1:]` to vectors
x = keras.layers.GlobalAveragePooling2D()(x)

# A Dense classifier with a single unit (binary classification)
outputs = keras.layers.Dense(1)(x)
model = keras.Model(inputs, outputs)

 model.compile(optimizer=keras.optimizers.Adam(),
           loss=keras.losses.BinaryCrossentropy(from_logits=True),
           metrics=[keras.metrics.BinaryAccuracy()])
model.fit(new_dataset, epochs=20, callbacks=..., validation_data=...)
```

https://www.tensorflow.org/tutorials/images/transfer_learning_with_hub

```
!wget https://download.pytorch.org/tutorial/hymenoptera_data.zip
!unzip -nq hymenoptera_data.zip
# Download the dataset
```

```
import numpy as np
import os
import PIL
import PIL.Image
import tensorflow as tf
data_dir = "hymenoptera_data"

train_ds =
     tf.keras.utils.image_dataset_from_directory(
     data_dir+"/train", image_size=(224, 224),
     #label_mode='categorical', batch_size=4)

val_ds = tf.keras.utils.image_dataset_from_directory(
     data_dir+"/val", image_size=(224, 224),
     #label_mode='categorical',batch_size=4)
print(train_ds.class_names)
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 224, 224, 3)]     0
_________________________________________________________________
rescaling (Rescaling)        (None, 224, 224, 3)       0
_________________________________________________________________
xception (Functional)        (None, 7, 7, 2048)        20861480
_________________________________________________________________
global_average_pooling2d (Gl (None, 2048)              0
_________________________________________________________________
dense (Dense)                (None, 1)                 2049
=================================================================
Total params: 20,863,529
Trainable params: 2,049
Non-trainable params: 20,861,480
```

Google drive - Resnet18-Modeling

```
base_model = tf.keras.applications.Xception(
     weights='imagenet', # Load weights pre-trained on ImageNet.
     input_shape=(224, 224, 3),
     include_top=False) # Do not include the ImageNet classifier at the top.
base_model.trainable = False
inputs = tf.keras.Input(shape=(224, 224, 3))
# rescale input
inputs1 = tf.keras.layers.Rescaling(scale=1 / 127.5, offset=-1)(inputs)
# We make sure that the base_model is running in inference mode here,
# by passing `training=False`. This is important for fine-tuning, as you will
# learn in a few paragraphs.
x = base_model(inputs1, training=False)
# Convert features of shape `base_model.output_shape[1:]` to vectors
x = tf.keras.layers.GlobalAveragePooling2D()(x)
# A Dense classifier with a single unit (binary classification)
outputs = tf.keras.layers.Dense(1)(x)
#utputs = tf.keras.layers.Activation('softmax')(outputs)
model = tf.keras.Model(inputs, outputs)
model.summary()
```

```
model.compile(optimizer=tf.keras.optimizers.Adam(),
loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
metrics=[tf.keras.metrics.BinaryAccuracy()])

model.fit(train_ds, epochs=5, validation_data=val_ds)
```

```
Epoch 1/5 62/62 [==============================] –
80s 1s/step - loss: 0.2754 - binary_accuracy: 0.8980 - val_loss: 0.2117 - val_binary_accuracy: 0.8954
```

```
# Model without transfer learning
base_model = tf.keras.applications.Xception(
weights=None, # Load weights pre-trained on ImageNet.
input_shape=(224, 224, 3),
include_top=False) # Do not include the ImageNet classifier at the top.
base_model.trainable = True
inputs = tf.keras.Input(shape=(224, 224, 3))
# rescale input
inputs1 = tf.keras.layers.Rescaling(scale=1 / 127.5, offset=-1)(inputs)

x = base_model(inputs1, training=True)
# Convert features of shape `base_model.output_shape[1:]` to vectors
x = tf.keras.layers.GlobalAveragePooling2D()(x)
# A Dense classifier with a single unit (binary classification)
outputs = tf.keras.layers.Dense(1)(x)
#utputs = tf.keras.layers.Activation('softmax')(outputs)
model = tf.keras.Model(inputs, outputs)
model.summary()
```

```
Model: "model_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_6 (InputLayer)         [(None, 224, 224, 3)]     0
_________________________________________________________________
rescaling_2 (Rescaling)       (None, 224, 224, 3)       0
_________________________________________________________________
xception (Functional)         (None, 7, 7, 2048)        20861480
_________________________________________________________________
global_average_pooling2d_2 (  (None, 2048)              0
_________________________________________________________________
dense_2 (Dense)               (None, 1)                 2049
=================================================================
Total params: 20,863,529
Trainable params: 20,809,001
Non-trainable params: 54,528
```

```
model.compile(optimizer=tf.keras.optimizers.Adam(),
loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
metrics=[tf.keras.metrics.BinaryAccuracy()])

model.fit(train_ds, epochs=5, validation_data=val_ds)
```

```
Epoch 1/5 62/62 [==============================]
- 265s 4s/step - loss: 1.0894 - binary_accuracy: 0.4490 - val_loss: 0.8420 - val_binary_accuracy: 0.5556
```

Transfer Learning

```
data_dir = 'hymenoptera_data'
image_datasets = {x: datasets.ImageFolder(os.path.join(data_dir, x),
                                          data_transforms[x])
                  for x in ['train', 'val']}
dataloders = {x: torch.utils.data.DataLoader(image_datasets[x], batch_size=4,
                                             shuffle=True, num_workers=4)
              for x in ['train', 'val']}
dataset_sizes = {x: len(image_datasets[x]) for x in ['train', 'val']}
class_names = image_datasets['train'].classes

use_gpu = torch.cuda.is_available()
```

Collects the data from the local directory “hymenoptera_data” and performs data_transforms on them. Gets class names from the subdirectory names in the train and val folder: “bees” and “ants”

Transfer Learning


```
model_ft = models.resnet18(pretrained=True) # load weights from resnet18 network trained on imagenet
num_ftrs = model_ft.fc.in_features
model_ft.fc = nn.Linear(num_ftrs, 2)

if use_gpu:
   model_ft = model_ft.cuda()

criterion = nn.CrossEntropyLoss()

# Observe that all parameters are being optimized
optimizer_ft = optim.SGD(model_ft.parameters(), lr=0.001, momentum=0.9)

# Decay LR by a factor of 0.1 every 7 epochs
exp_lr_scheduler = lr_scheduler.StepLR(optimizer_ft, step_size=7, gamma=0.1)

model_ft = train_model(model_ft, criterion, optimizer_ft, exp_lr_scheduler,
                       num_epochs=25)
```

Loads the pretrained ResNet18 ImageNet model and add a fully connected layer to the end with 2 outputs. We use cross entropy loss and SGD optimization. We the train the model using the train_model function.

Transfer Learning

Epoch 0/24
---
train Loss: 0.1285 Acc: 0.7377
val Loss: 0.0729 Acc: 0.8497

Epoch 1/24
---
train Loss: 0.1418 Acc: 0.7828
val Loss: 0.0808 Acc: 0.8562

Epoch 2/24
---
train Loss: 0.1319 Acc: 0.7828
val Loss: 0.0580 Acc: 0.8954

Epoch 3/24
---
train Loss: 0.1452 Acc: 0.8197
val Loss: 0.0822 Acc: 0.8824

Epoch 21/24
---
train Loss: 0.0669 Acc: 0.8975
val Loss: 0.0640 Acc: 0.9150

Epoch 22/24
---
train Loss: 0.0702 Acc: 0.8852
val Loss: 0.0618 Acc: 0.9085

Epoch 23/24
---
train Loss: 0.0667 Acc: 0.8852
val Loss: 0.0647 Acc: 0.8954

Epoch 24/24
---
train Loss: 0.0779 Acc: 0.8648
val Loss: 0.0633 Acc: 0.9150

Training complete in 2m 27s
Best val Acc: 0.934641

We start with an accuracy of 74% and after training we end with an accuracy of 86%

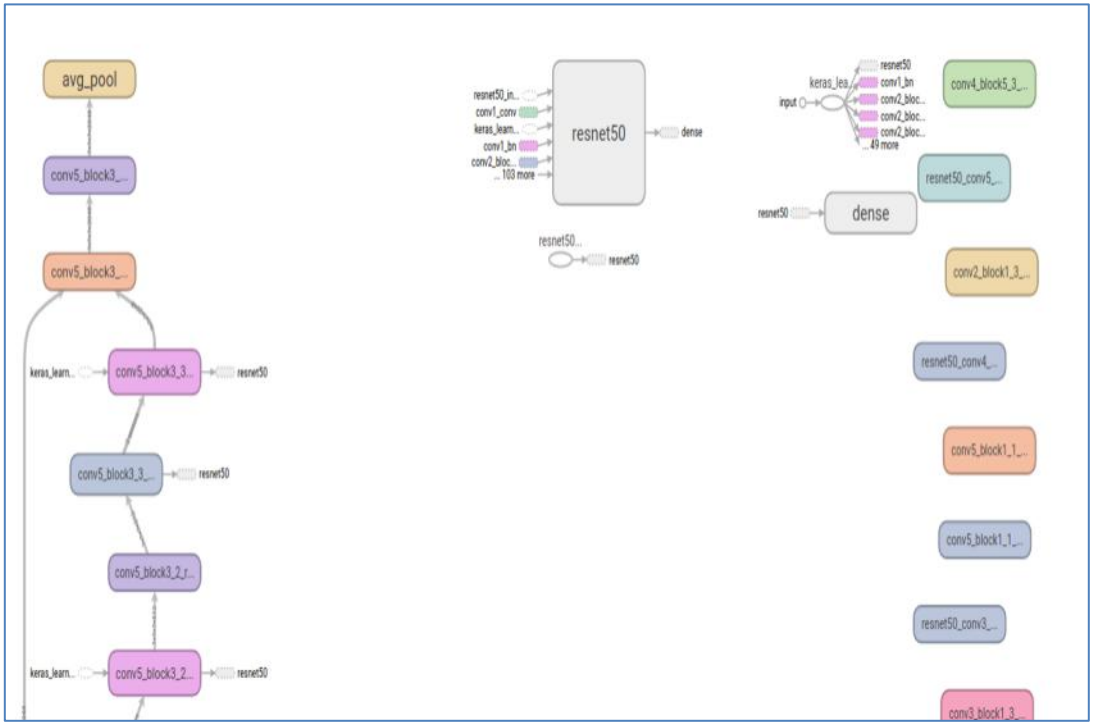
Tensorboard

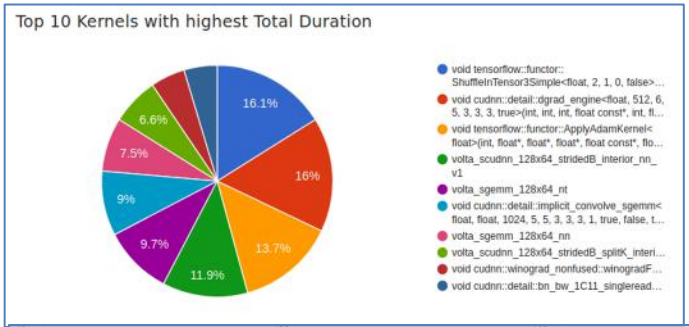
TensorBoard is a visualization, profiling, and debugging tool for TensorFlow. These are examples from TensorBoard after training using ResNet50 on ImageNet

Install TensorBoard
conda -c conda-forge tensorboard

Create callback and add it into model fit.

```
tensorboard_callback =
tf.keras.callbacks.TensorBoard(log_dir=log_dir, histogram_freq=1)
model.fit(x=x_train,
          y=y_train,
          epochs=5,
          validation_data=(x_test, y_test),
          callbacks=[tensorboard_callback])
```





TensorFlow Callbacks

```
class MyCustomCallback(tf.keras.callbacks.Callback):
 def on_epoch_begin(self, epoch, logs=None):
   self.epoch_start = datetime.datetime.now()
   self.num_epochs = self.num_epochs + 1
 def on_epoch_end(self, epoch, logs=None):
   epoch_time = (datetime.datetime.now() - self.epoch_start).total_seconds()
   print('custom callback: time:{}'.format(epoch_time))
   self.total_epoch_time = self.total_epoch_time + epoch_time
... load data + build and compile model
callbacks = [
   MyCustomCallback()
]
model.fit(x=x_train,
          y=y_train,
          epochs=5,
          validation_data=(x_test, y_test),
          callbacks=callbacks)
```

Getting data while training can be done using a callback.

```
Epoch 2/2
781/782 [==============================>.] - ETA: 0s - loss: 2.0883 -
accuracy: 0.3686 custom callback: time:30.128066
```

Program output

ImageNet + Resnet

With a model summary and tensorboard

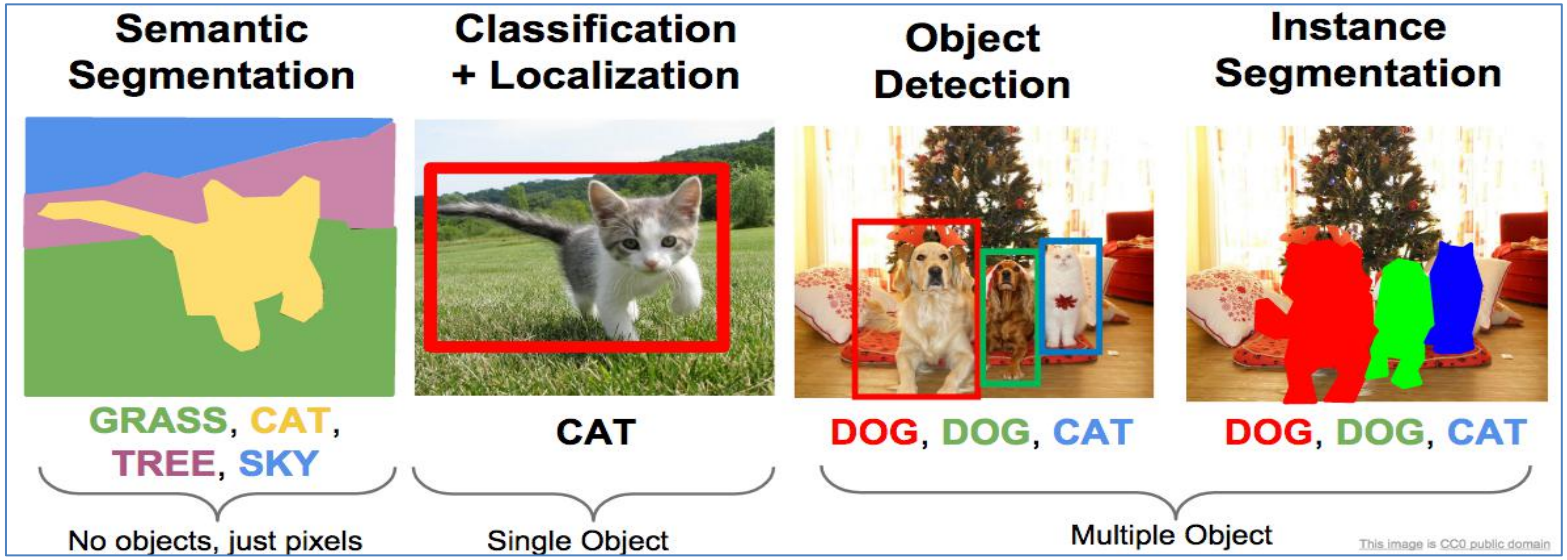
https://colab.research.google.com/drive/1uzKlU2d0g9ADqy4YamV-9AcpnnM7189I?usp=sharing

```
def scale(image, label):
     image = tf.cast(image, tf.float32)
     image /= 255
     return image, label
BATCH_SIZE = 64
datasets = tfds.load('cifar100',as_supervised=True)
ds_train, ds_test = datasets['train'], datasets['test']
train_dataset = ds_train.map(scale).batch(BATCH_SIZE)
eval_dataset = ds_test.map(scale).batch(BATCH_SIZE)
logdir = os.path.join("logs", datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))
tensorboard_callback = tf.keras.callbacks.TensorBoard(logdir, histogram_freq=1)
model = Sequential()
model.add(ResNet50( include_top = False, pooling = 'avg'))
model.add(Dense(100, activation = 'softmax'))
model.summary()
loss_fn = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
model.compile(optimizer='adam',
 loss=loss_fn,
 metrics=['accuracy'])
model.fit(train_dataset, epochs=5, callbacks=[tensorboard_callback])
```

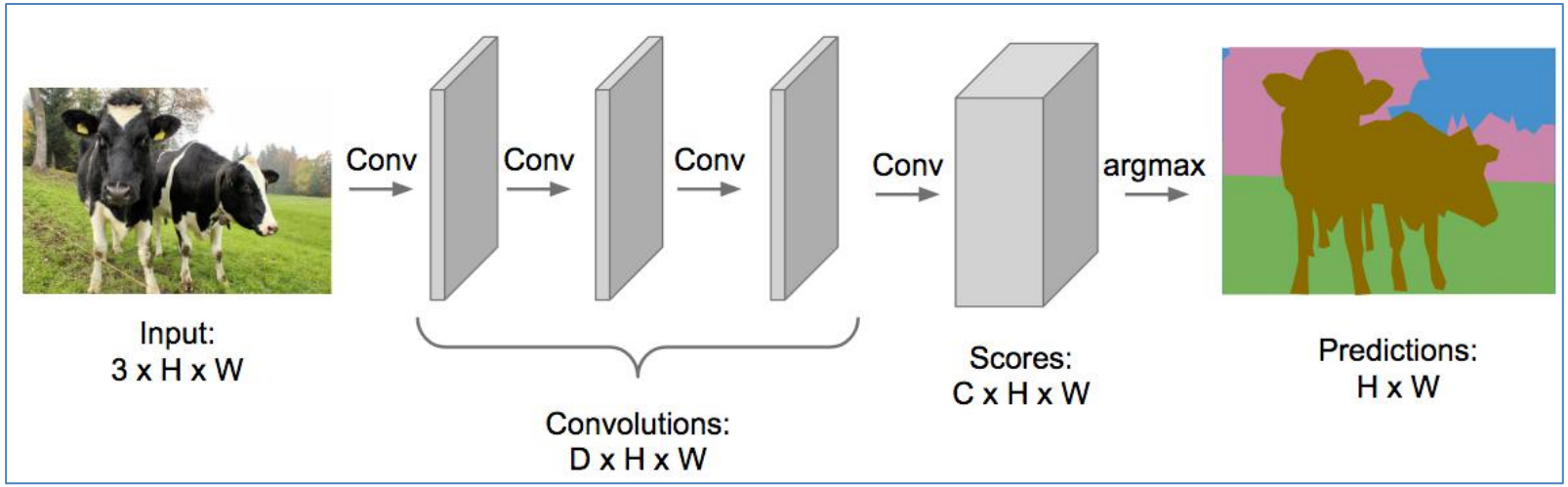
Topics of Unit 11

- ✓ **UNIT 11 : DNN Computing**
  - **DNN Principles: Mathematics**
  - **CNN Model: ResNet**
  - **DNN Computing: Essential Practices**
  - **DNN Usage: I/O, Transfer Learning**
  - **CNN Applications: Unet, Object Detection, Autonomous Vehicle**
  - **Advanced Topics: GAN, Reinforcement Learning**

Segmentation and Detection



Segmentation : Label each pixel in the image with a category label



https://github.com/tensorflow/hub/blob/master/examples/colab/tf2_object_detection.ipynb

http://cs231n.stanford.edu/slides/2020/lecture_12.pdf

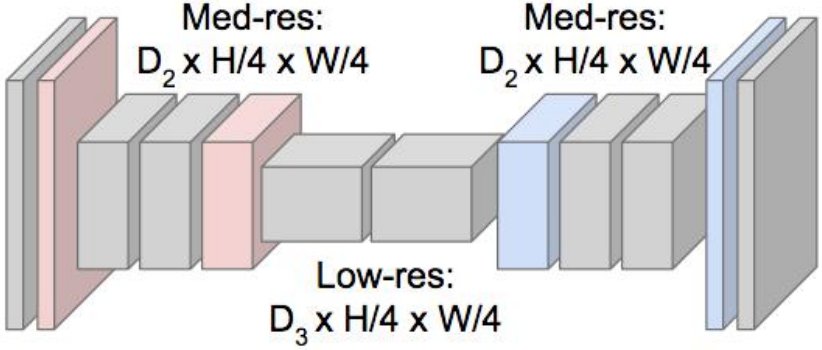
Semantic Segmentation Idea: Fully Convolutional

Downsampling: Pooling, strided convolution

Design network as a bunch of convolutional layers, with **downsampling** and **upsampling** inside the network!

Upsampling: Unpooling or strided transpose convolution

Input: 3 x H x W



High-res: D1 x H/2 x W/2

High-res: D1 x H/2 x W/2

Predictions: H x W

Max Pooling
Remember which element was max!

1	2	6	3
3	5	2	1
1	2	2	1
7	3	4	8

Input: 4 x 4

→

5	6
7	8

Output: 2 x 2

→ ... → Rest of the network

Max Unpooling
Use positions from pooling layer

1	2
3	4

Input: 2 x 2

→

0	0	2	0
0	1	0	0
0	0	0	0
3	0	0	4

Output: 4 x 4

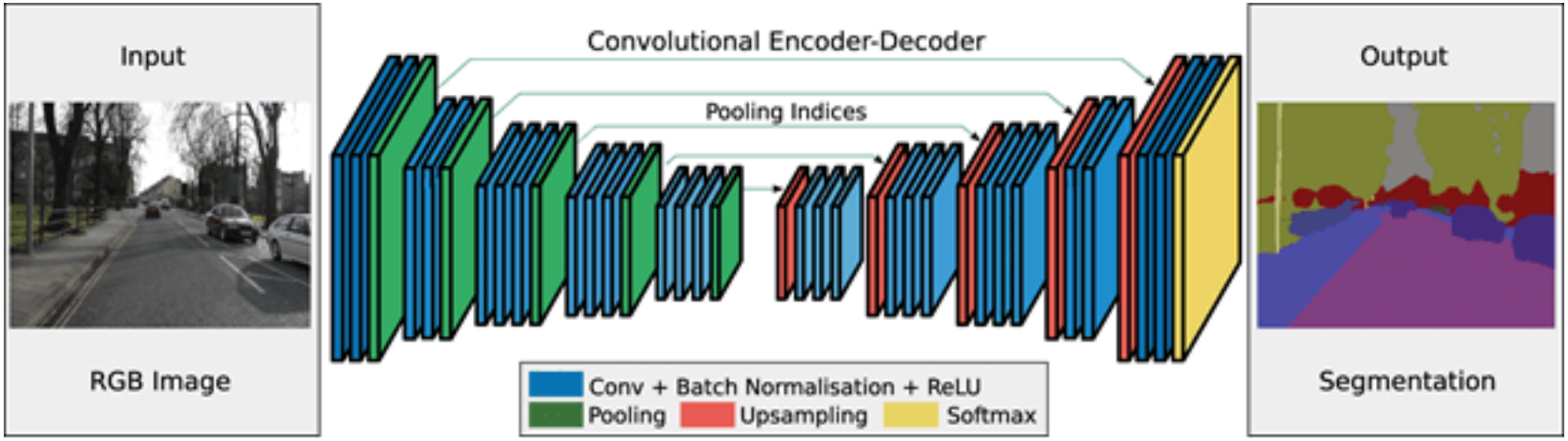
https://web.eecs.umich.edu/~justincj/teaching/eecs498/FA2020/schedule.html

Image segmentation

Divide an image into multiple segments, every pixel in an image is associated with an object, instance segmentation and semantic segmentation :

In semantic segmentation, all objects of the same type are marked using one class label while in instance segmentation similar objects get their own separate labels.

The basic architecture in image segmentation consists of an encoder and a decoder.



The encoder extracts features from the image through filters. The decoder is responsible for generating the final output which is usually a segmentation mask containing the outline of the object. Most of the architectures have this architecture or a variant of it.

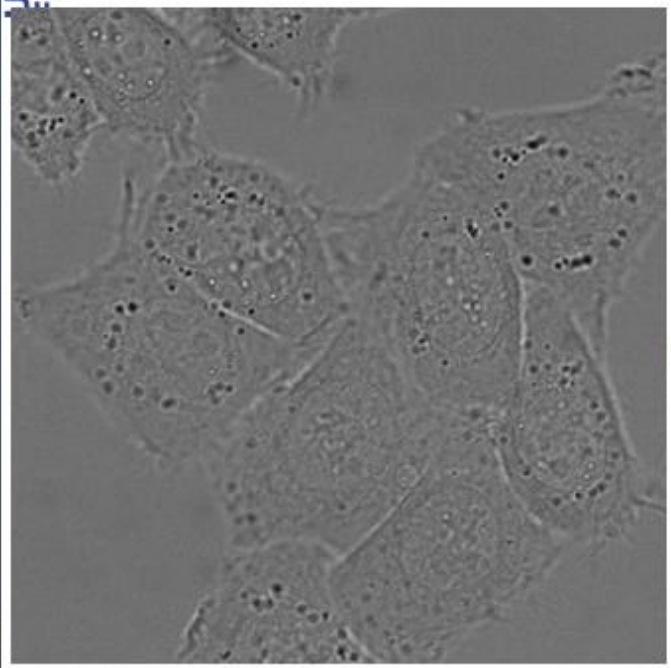
https://neptune.ai/blog/image-segmentation-in-2020

U-net: Convolutional Networks for Biomedical Image Segmentation

Olaf Ronneberger, Philipp Fischer, and Thomas Brox

Computer Science Department and
BIOSS Centre for Biological Signalling Studies
University of Freiburg, Germany

Biomedical Image Segmentation with U-net



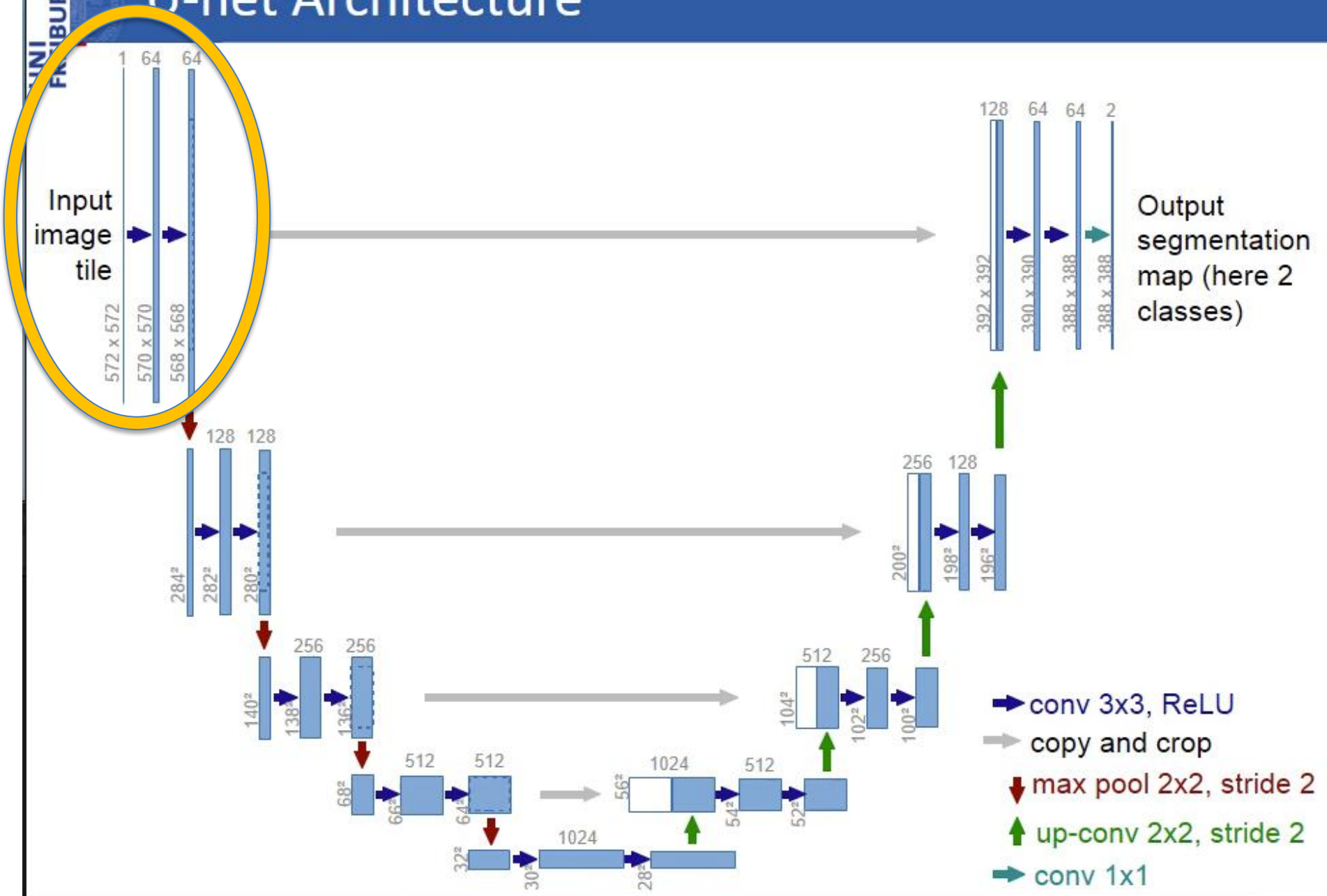
input image

output segmentation map

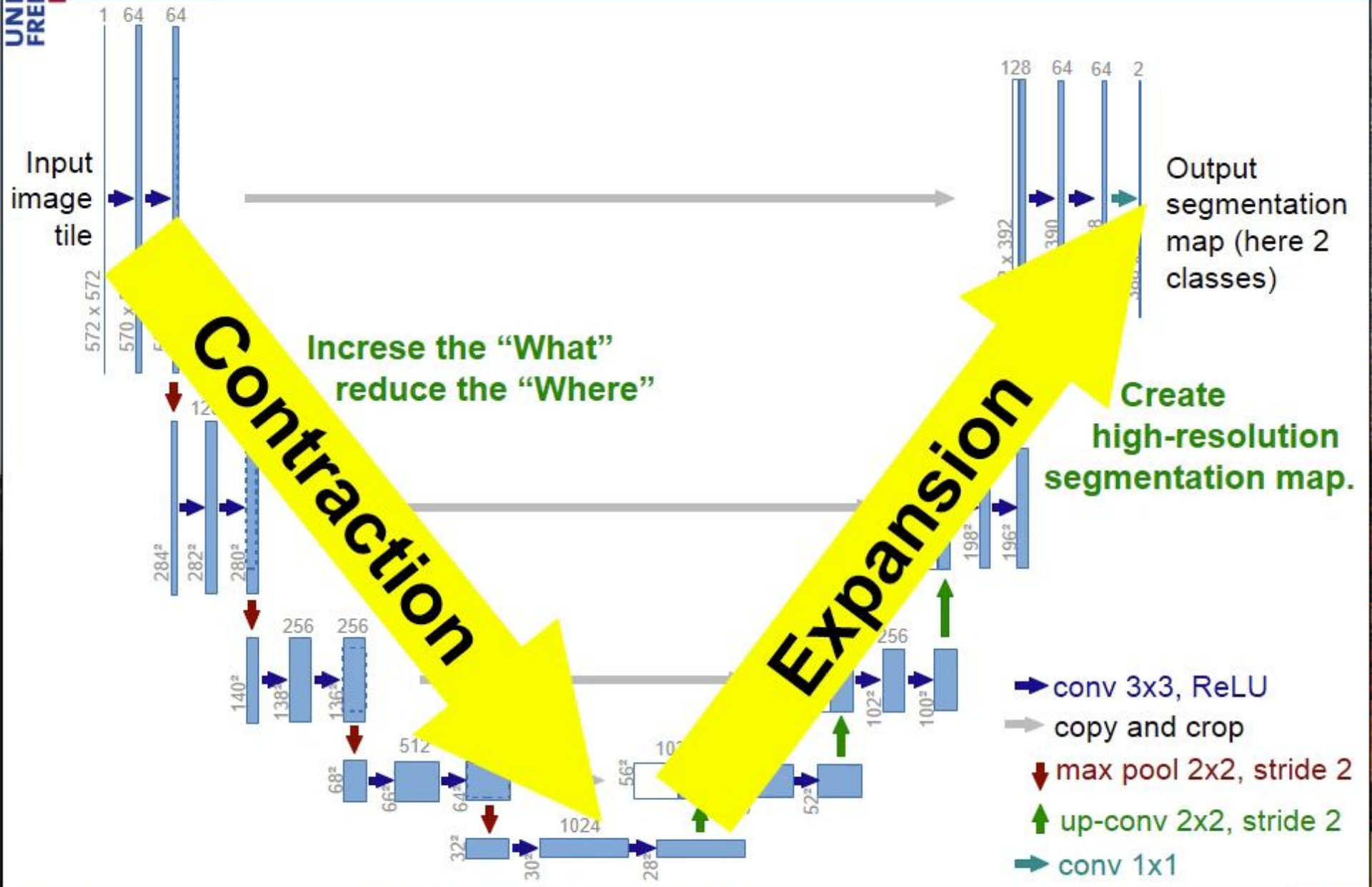
- U-net **learns segmentation** in an end-to-end setting
- **Very few annotated images** (approx. 30 per application)
- **Touching objects** of the same class

[Data provided by Dr. Gert van Cappellen, Erasmus Medical Center. Rotterdam. The Netherlands]

U-net Architecture



U-Net Architecture



CODE explanation
https://keras.io/examples/vision/oxford_pets_image_segmentation

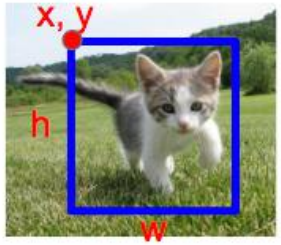
Single Object Detection

Used weighted sum of Losses of Softmax (CE) and L2(MSE) (scalar)

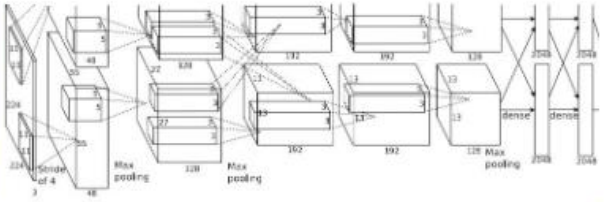
Object Detection: Single Object

(Classification + Localization)

Use pretrain network
Transfer learning from
Imagenet dataset



This image is CC0 public domain



Fully Connected: 4096 to 1000

Class Scores
Cat: 0.9
Dog: 0.05
Car: 0.01
...

Correct label: Cat

Softmax Loss

Multitask Loss

Loss

Vector: 4096

Fully Connected: 4096 to 4

Box Coordinates (x, y, w, h)

L2 Loss

Correct box: (x', y', w', h')

Treat localization as a regression problem!

http://cs231n.stanford.edu/slides/2020/lecture_12.pdf

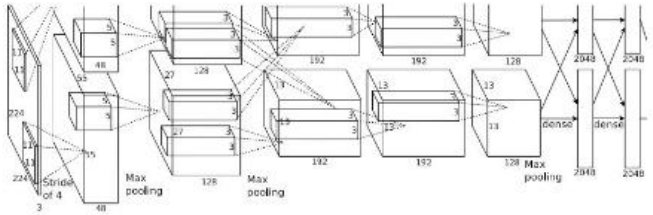
https://github.com/tensorflow/hub/blob/master/examples/colab/tf2_object_detection.ipynb

https://web.eecs.umich.edu/~justincj/teaching/eecs498/FA2020/schedule.html

Object Detection: Multiple Objects Using sliding Windows

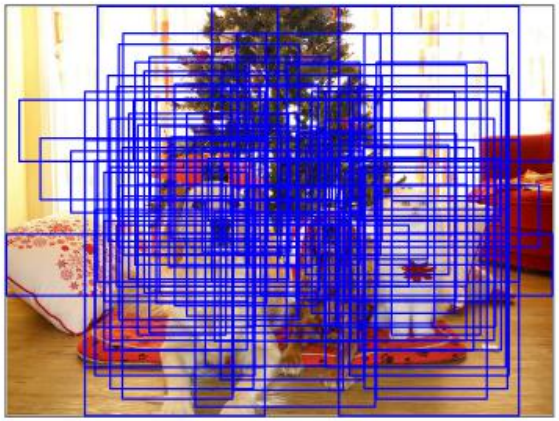
Apply a CNN to many different crops of the image, CNN classifies each crop as object or background

Slide window region -> many inputs -> classification

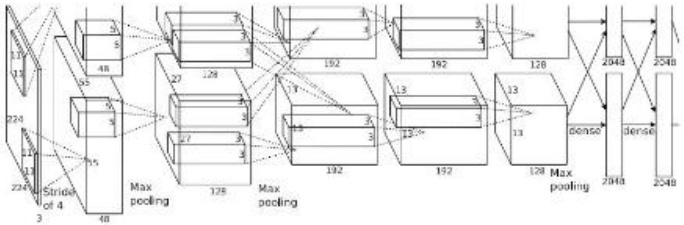


Dog? YES
Cat? NO
Background? NO

https://web.eecs.umich.edu/~justincj/teaching/eecs498/FA2020/schedule.html



Slide window region -> too expensive, NO

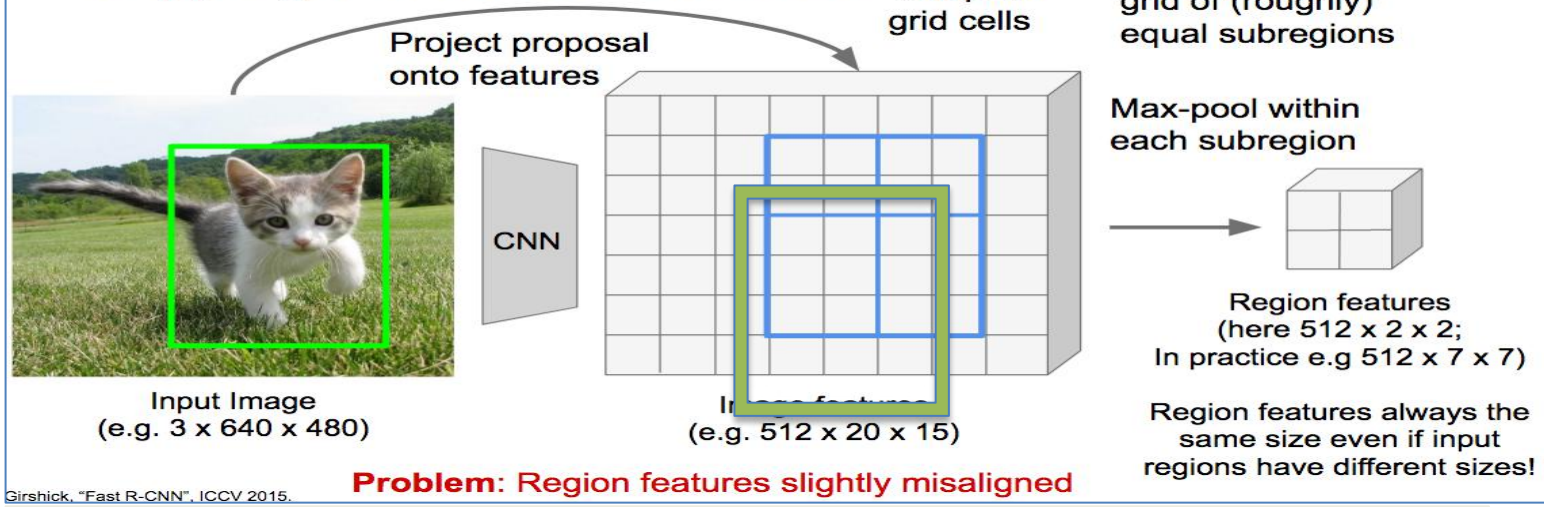


Dog? NO
Cat? YES
Background? NO

Problem: Need to apply CNN to huge number of locations, scales, and aspect ratios, very computationally expensive!

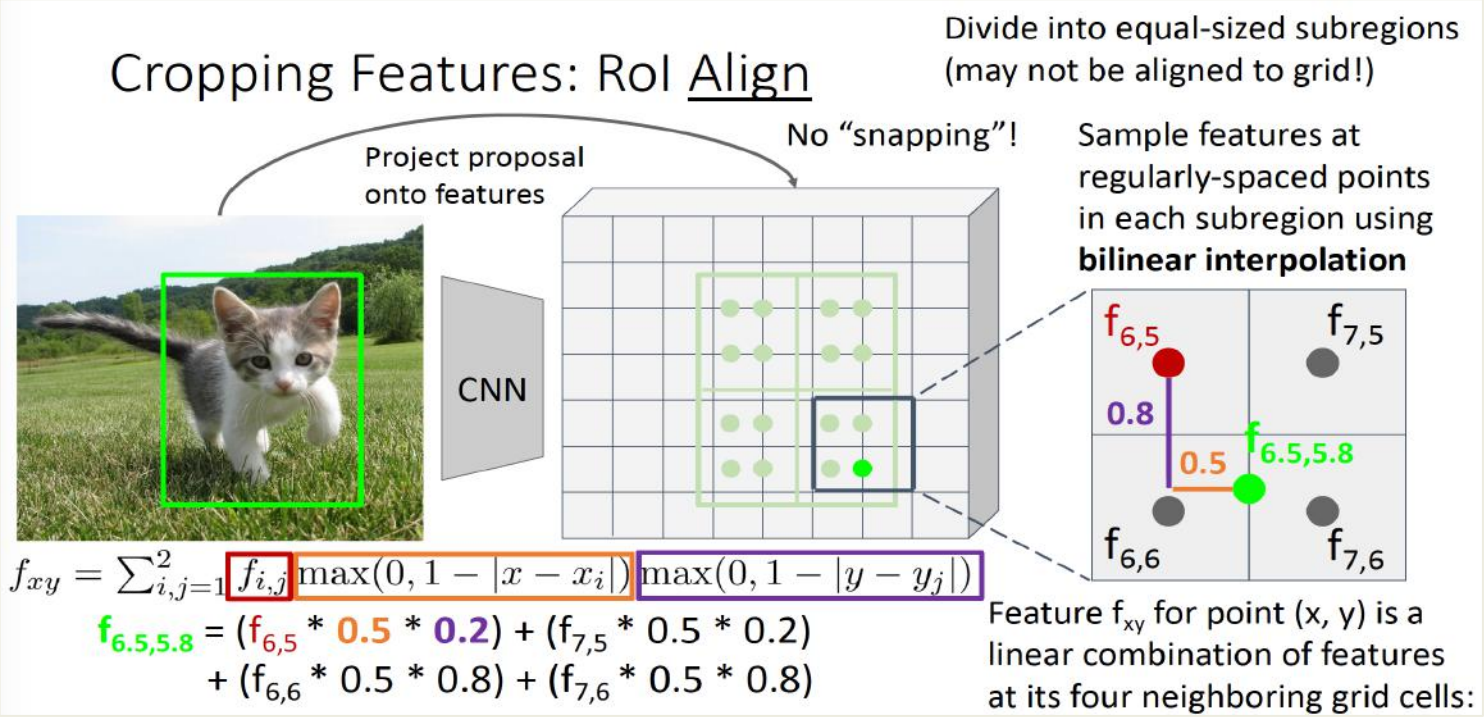
Cropping : Region of Interest

Cropping Features: RoI Pool



Girshick, "Fast R-CNN", ICCV 2015.

Cropping Features: RoI Align



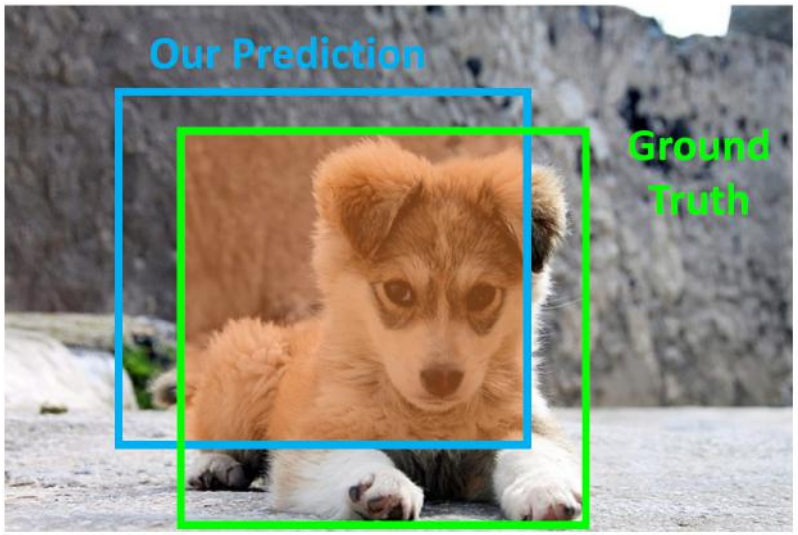
https://web.eecs.umich.edu/~justincj/teaching/eecs498/FA2020/schedule.

Performance Evaluation:
Comparing Boxes: Intersection over Union (IoU)

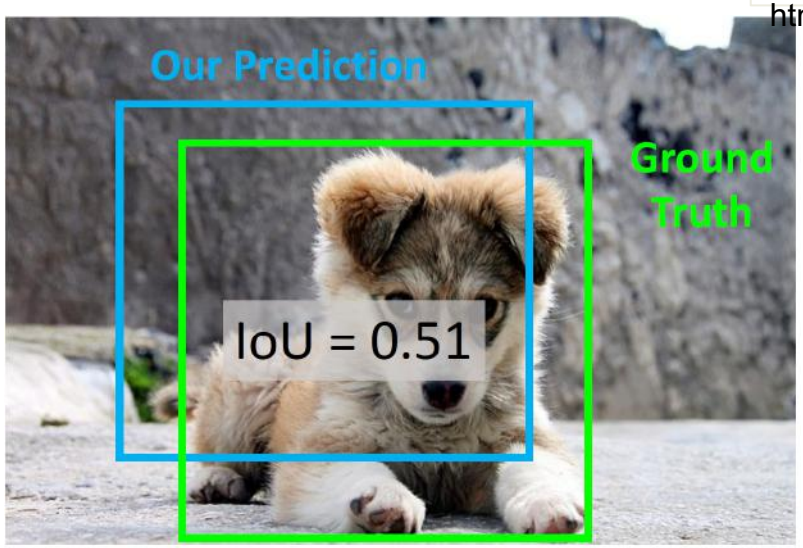
How can we compare our prediction to the ground-truth box?

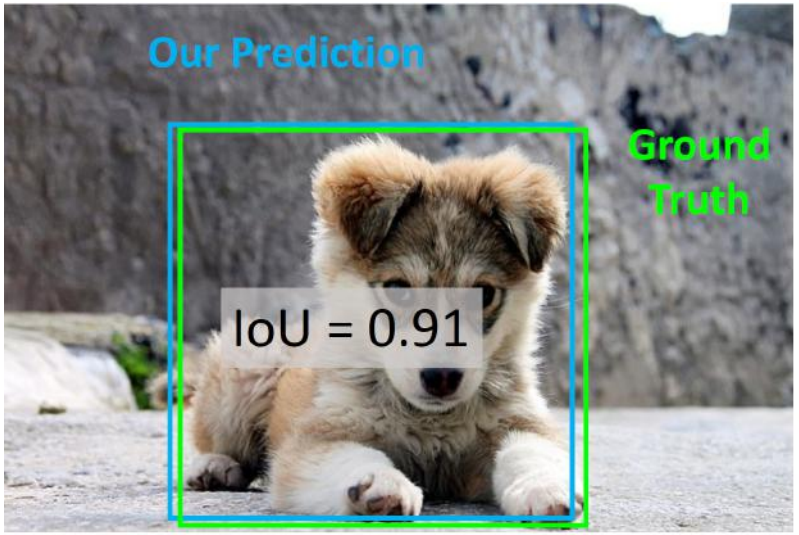
Intersection over Union (IoU)
(Also called “Jaccard similarity” or “Jaccard index”):

$$\frac{\textit{Area of Intersection}}{\textit{Area of Union}}$$

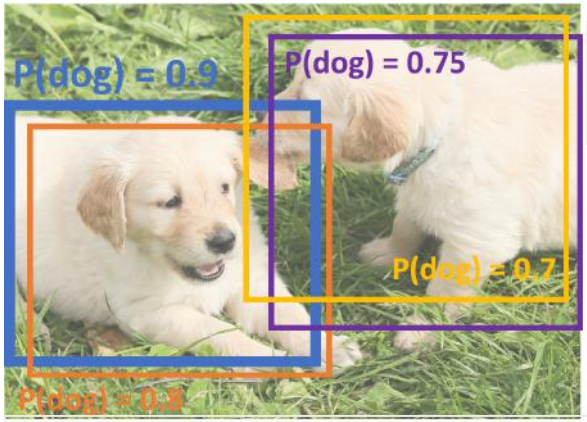


https://web.eecs.umich.edu/~justincj/teaching/eecs498/FA2020/schedule.html





IoU > 0.5 is “decent”, IoU > 0.7 is “pretty good”, IoU > 0.9 is “almost perfect”



Overlapping Boxes: Non-Max Suppression (NMS)

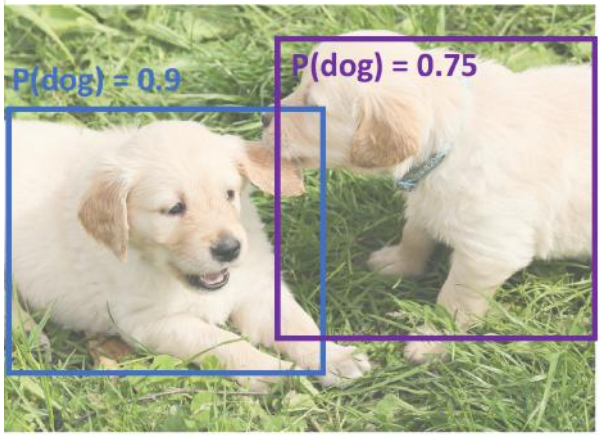
Problem: NMS may eliminate "good" boxes when objects are highly overlapping… no good solution

Problem: Object detectors often output many overlapping detections:

Solution: Post-process raw detections using Non-Max Suppression (NMS)

1. Select next highest-scoring box
2. Eliminate lower-scoring boxes with IoU > threshold (e.g. 0.7)
3. If any boxes remain, GOTO 1

P(dog) = 0.8
P(dog) = 0.75
P(dog) = 0.7
IoU(■, ■) = 0.78
IoU(■, ■) = 0.05
IoU(■, ■) = 0.07



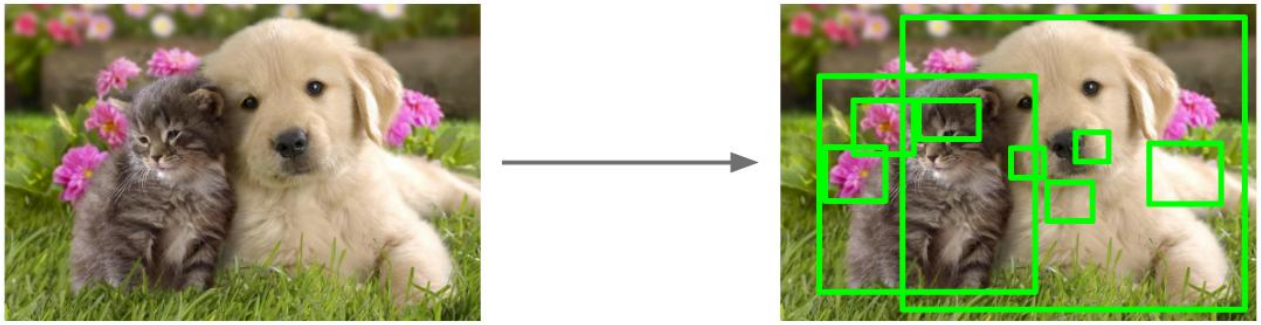
Evaluating Object Detectors: Mean Average Precision (mAP)

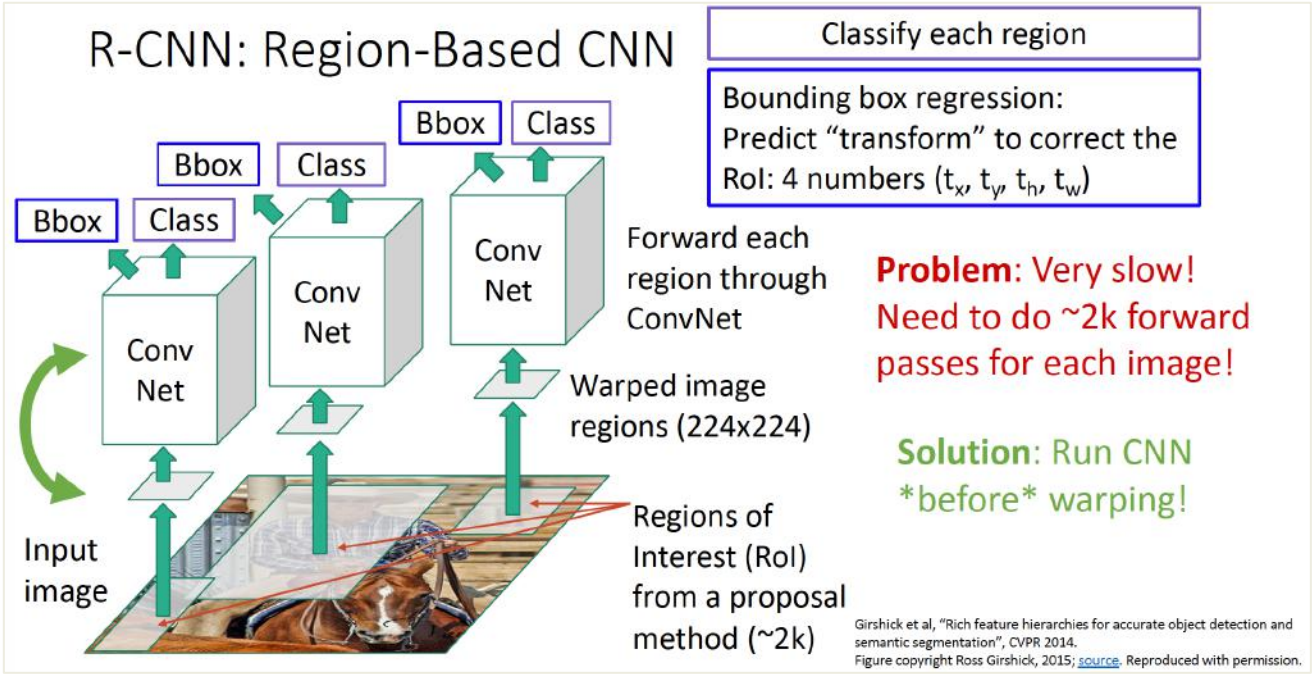
mAP@0.5 = 0.77
mAP@0.55 = 0.71
mAP@0.60 = 0.65
…
mAP@0.95 = 0.2
COCO mAP = 0.4

1. Run object detector on all test images (with NMS)
2. For each category, compute Average Precision (AP) = area under Precision vs Recall Curve
   1. For each detection (highest score to lowest score)
      1. If it matches some GT box with IoU > 0.5, mark it as positive and eliminate the GT
      2. Otherwise mark it as negative
      3. Plot a point on PR Curve
   2. Average Precision (AP) = area under PR curve
3. Mean Average Precision (mAP) = average of AP for each category
4. For "COCO mAP": Compute mAP@thresh for each IoU threshold (0.5, 0.55, 0.6, …, 0.95) and take average

Multiple Object Detection
Region Proposals : Selective Search

- Find "blobby" image regions that are likely to contain objects
- Relatively fast to run; e.g. Selective Search gives 1000 region proposals in a few seconds on CPU

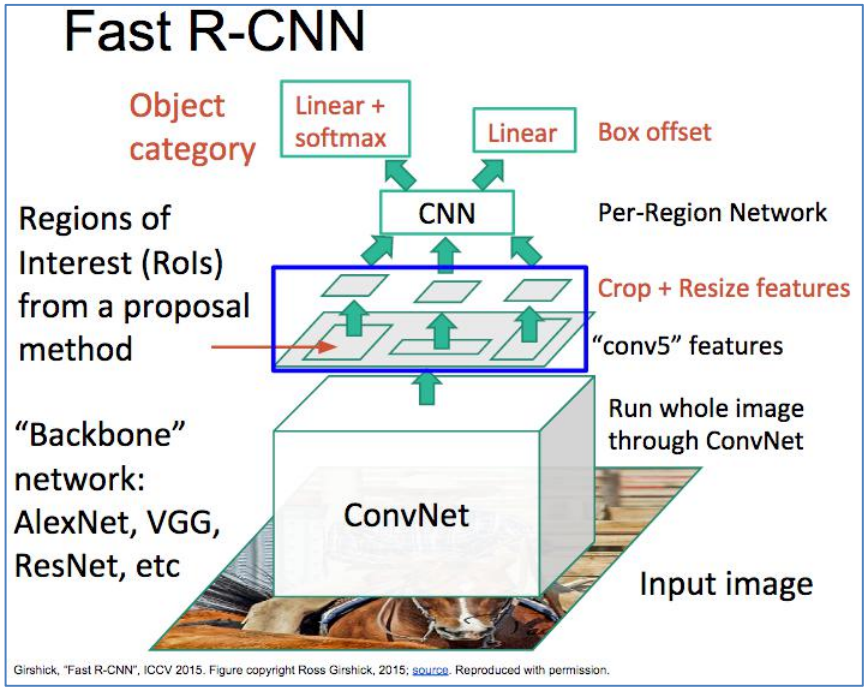


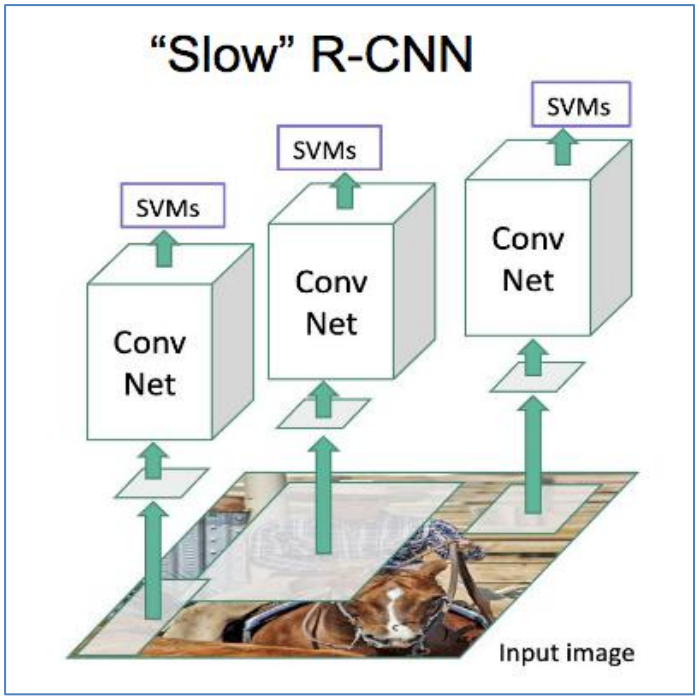


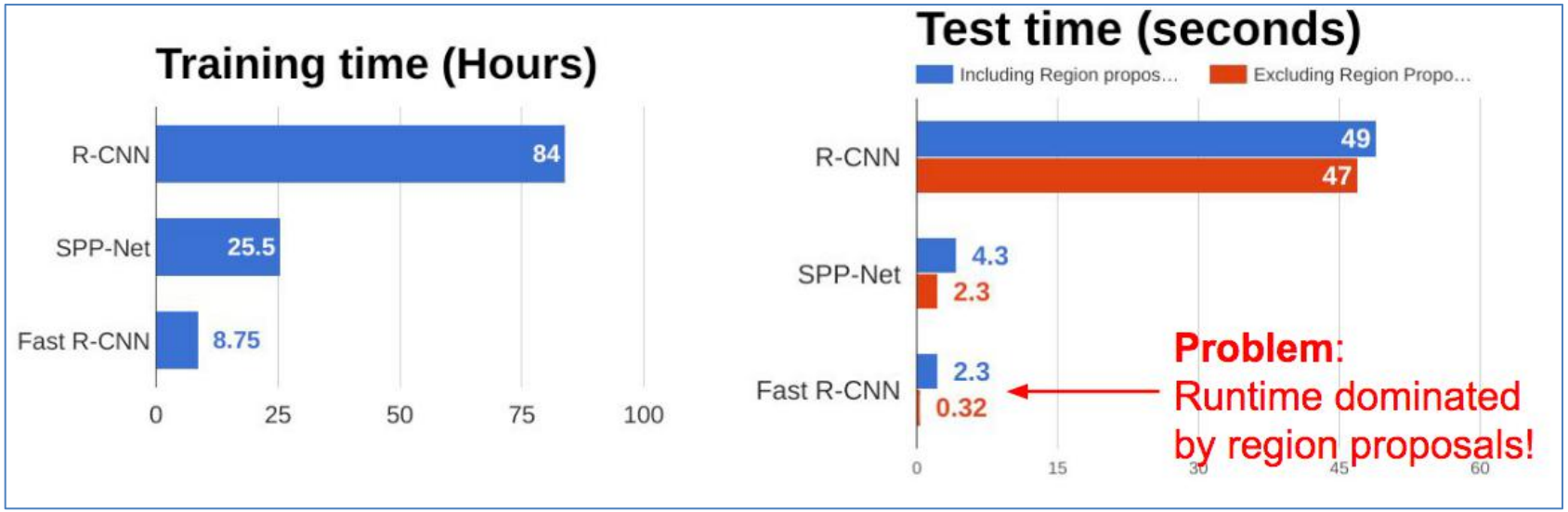
1. Run region proposal method to compute ~2000 region proposals
2. Resize each region to 224x224 and run independently through CNN to predict class scores and bbox transform
3. Use scores to select a subset of region proposals to output
4. (Many choices here: threshold on background, or per-category? Or take top K proposals per image?)
5. Compare with ground-truth boxes
6. Too Slow to compute 2000 CNN

https://web.eecs.umich.edu/~justincj/teaching/eecs498/FA2020/schedule.

Fast, R-CNN



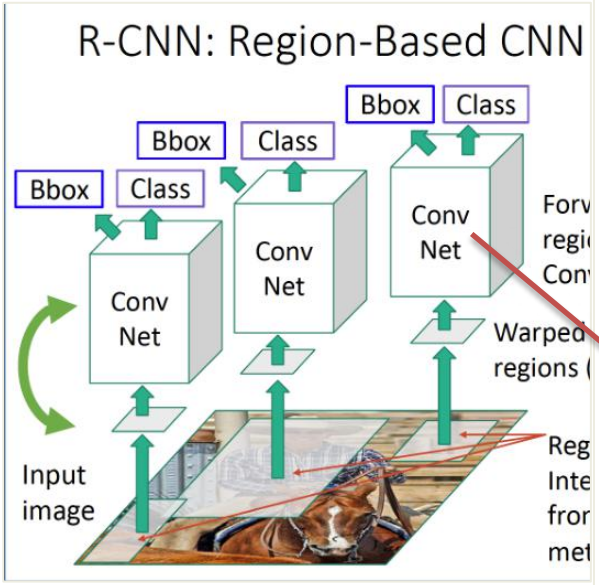




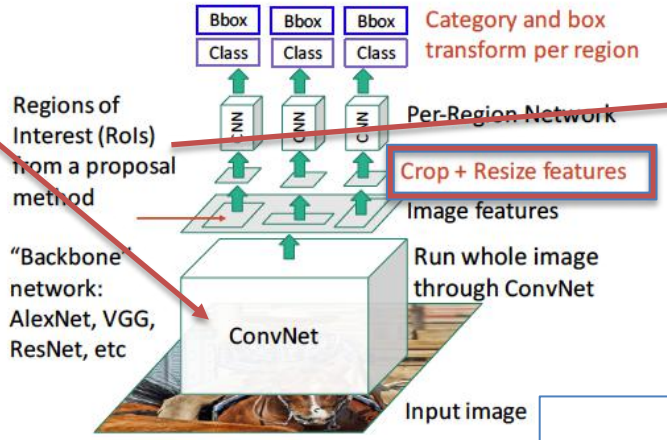
http://cs231n.stanford.edu/slides/2020/lecture_12.pdf

Object Detection

R-CNN: Region-Based CNN

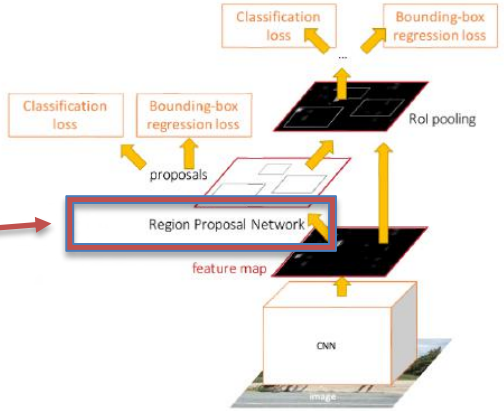


Fast R-CNN: Apply differentiable cropping to shared image features

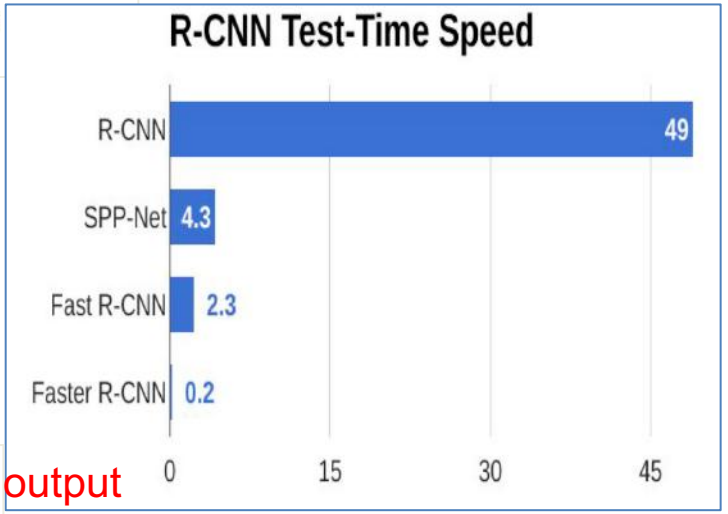


Faster R-CNN: Compute proposals with CNN

Replace Selective Search with CN



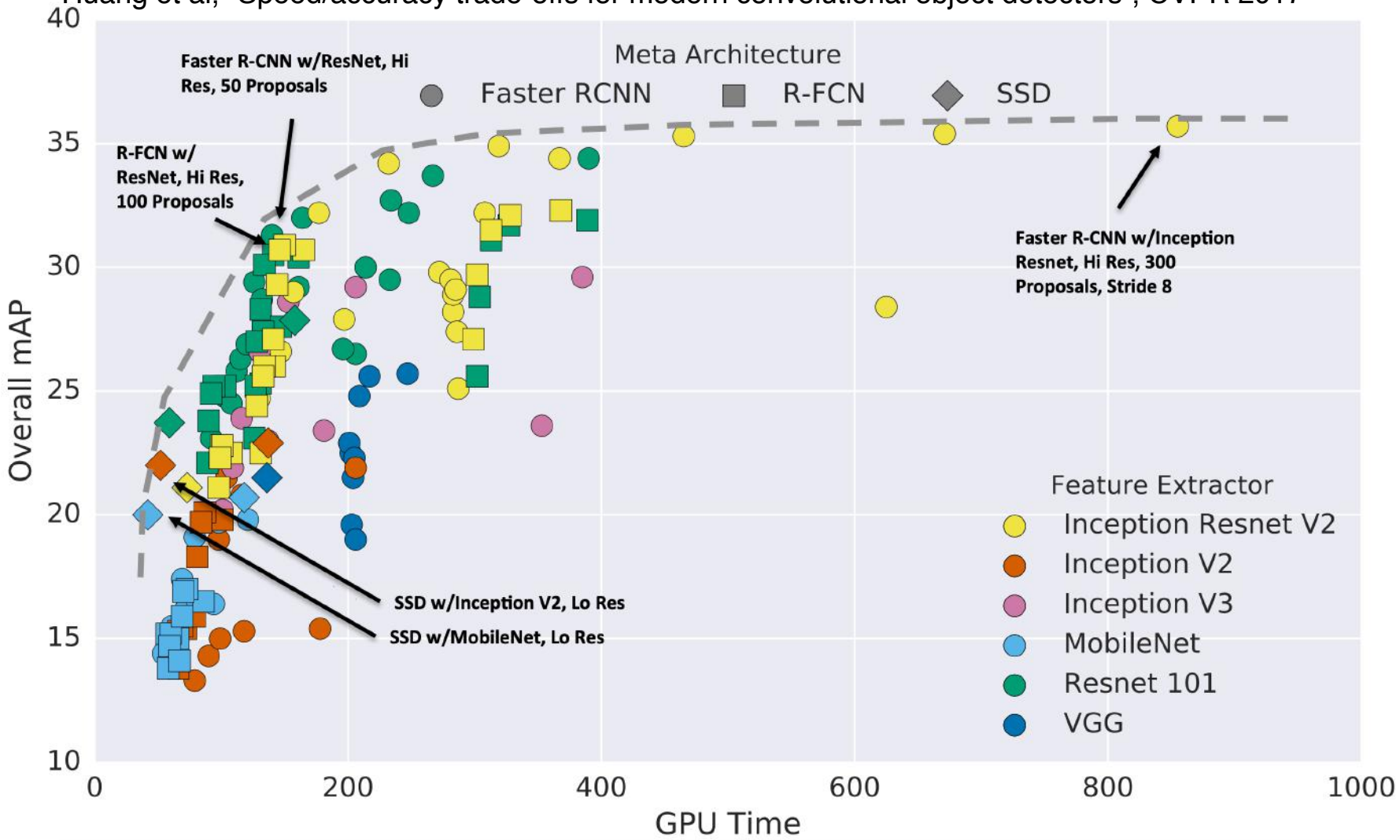
TensorFlow Detection API:
https://github.com/tensorflow/models/tree/master/research/object_detection
Faster R-CNN, SSD, RFCN, Mask R-CNN
Detectron2 (PyTorch):
https://github.com/facebookresearch/detectron2
Fast / Faster / Mask R-CNN, RetinaNet, ImageAI..



Input -> CONV -> Multi-task -stage -> Loss (CE+MSE+..) -> output

https://web.eecs.umich.edu/~justincj/teaching/eecs498/FA2020/schedule.

Huang et al, “Speed/accuracy trade-offs for modern convolutional object detectors”, CVPR 2017

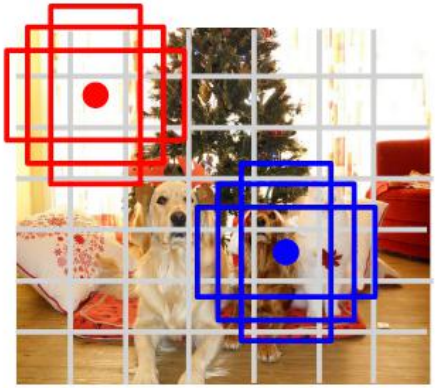


- Two stage method (Faster R-CNN) get the best accuracy, but are slower
- Single-stage methods (SSD) are much faster, but don’t perform as well
- Bigger backbones improve performance, but are slower
- Diminishing returns for slower methods
- Getter faster with larger networks, ensembles, big datasets..

https://web.eecs.umich.edu/~justincj/teaching/eecs498/FA2020/schedule.

YOLO/SSD

Input image
3 x H x W



Divide image into grid
7 x 7

Image a set of **base boxes** centered at each grid cell
Here B = 3

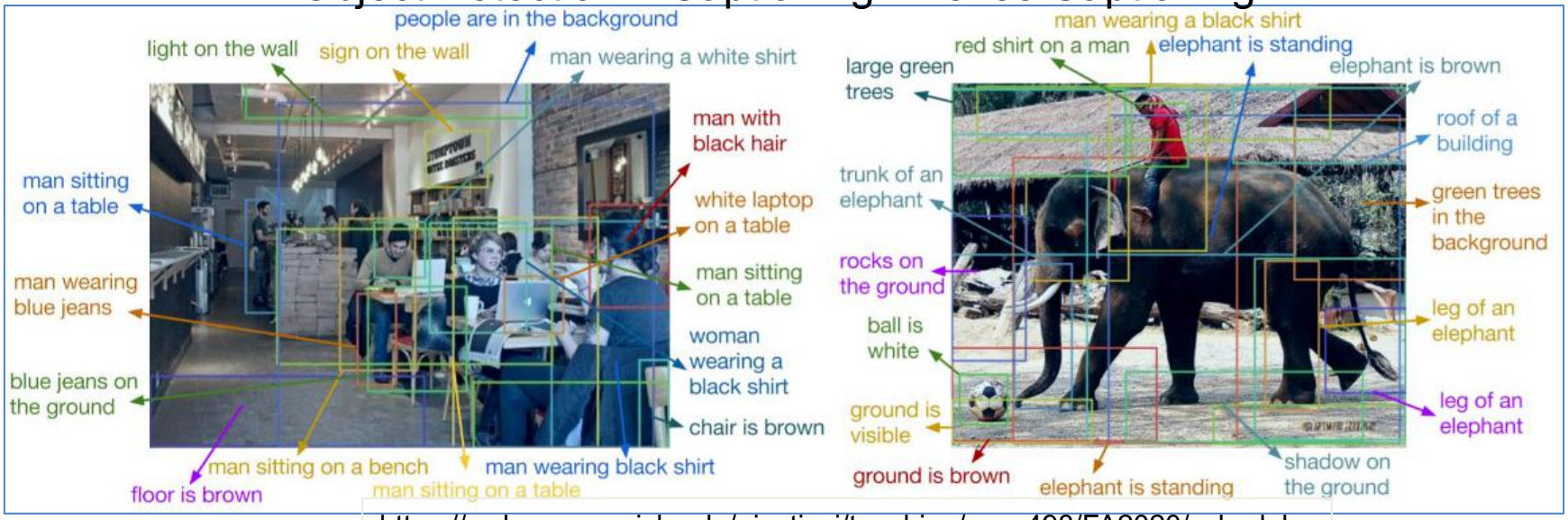
Within each grid cell:

- Regress from each of the B base boxes to a final box with 5 numbers: (dx, dy, dh, dw, confidence)
- Predict scores for each of C classes (including background as a class)
- Looks a lot like RPN, but category-specific!

Output:
7 x 7 x (5 * B + C)

Redmon et al, "You Only Look Once: Unified, Real-Time Object Detection", CVPR 2016
Liu et al, "SSD: Single-Shot MultiBox Detector", ECCV 2016
Lin et al, "Focal Loss for Dense Object Detection", ICCV 2017

Object Detection + Captioning= Dense Captioning



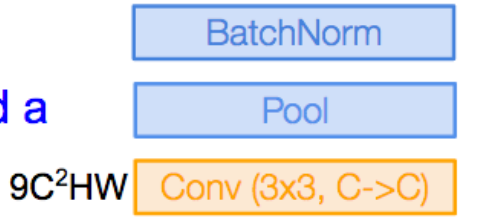
https://web.eecs.umich.edu/~justincj/teaching/eecs498/FA2020/schedule.

https://www.jics.utk.edu/recsem-reu/recsem19

MobileNets

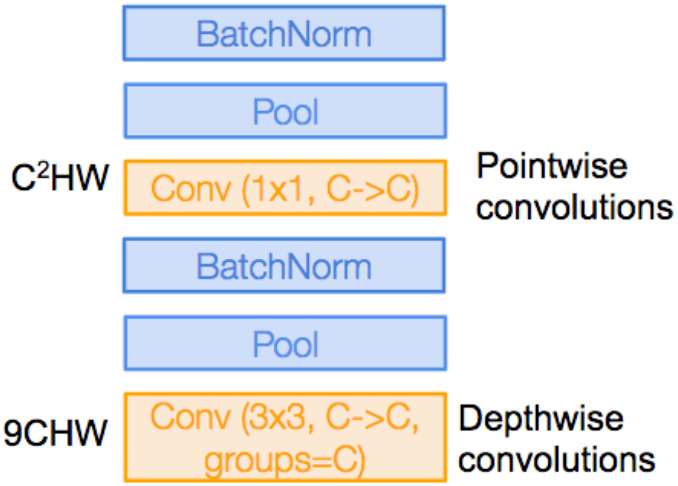
MobileNets: Efficient Convolutional Neural Networks for Mobile Applications *[Howard et al. 2017]*

- Depthwise separable convolutions replace standard convolutions by factorizing them into a depthwise convolution and a 1x1 convolution
- Much more efficient, with little loss in accuracy
- Follow-up MobileNetV2 work in 2018 (Sandler et al.)
- ShuffleNet: Zhang et al, CVPR 2018



Stanford network

Total compute:$9C^2HW$



MobileNets

Total compute:$9CHW + C^2HW$

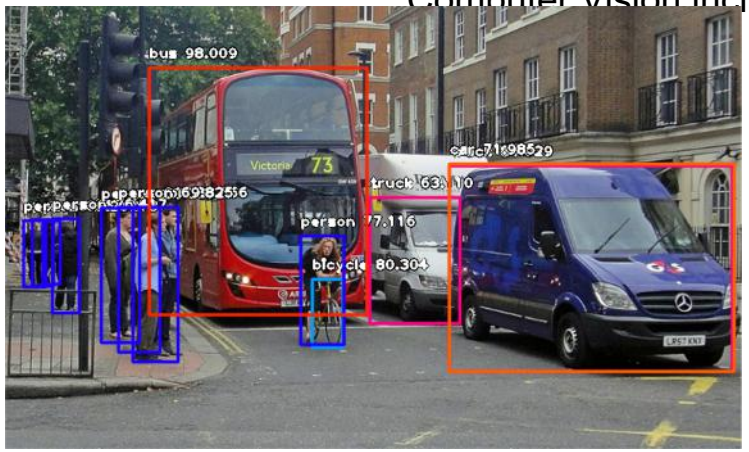
https://www.tensorflow.org/tutorials/images/segmentation

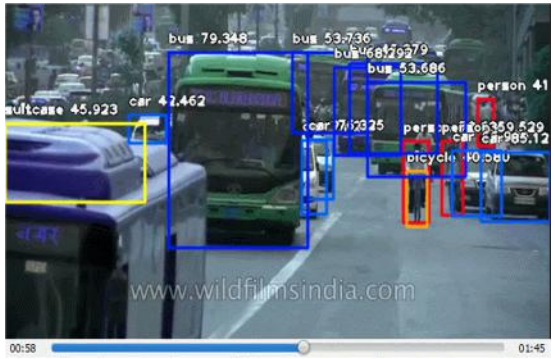
The model being used here is a modified U-Net. A U-Net consists of an encoder (downsampler) and decoder (upsampler). In-order to learn robust features, and reduce the number of trainable parameters, a pretrained model can be used as the encoder. Thus, the encoder for this task will be a pretrained MobileNetV2 model, whose intermediate outputs will be used, and the decoder will be the upsample block already implemented in TensorFlow Examples in the Pix2pix tutorial.

ImageAI

ImageAI supports a list of state-of-the-art Machine Learning algorithms for image prediction, custom image prediction, object detection, video detection, video object tracking and image predictions trainings. **ImageAI** currently supports image prediction and training using 4 different Machine Learning algorithms trained on the ImageNet-1000 dataset. **ImageAI** also supports object detection, video detection and object tracking using RetinaNet, YOLOv3 and TinyYOLOv3 trained on COCO dataset. Finally, **ImageAI** allows you to train custom models for performing detection and recognition of new objects.

Eventually, **ImageAI** will provide support for a wider and more specialized aspects of Computer Vision including and not limited to image recognition in special environments






```
from imageai.Prediction.Custom import ModelTraining
from imageai.Prediction import ImagePrediction
import os

model_trainer = ModelTraining()
model_trainer.setModelTypeAsResNet()

model_trainer.setDataDirectory("image_labels")

model_trainer.trainModel(num_objects=3,
num_experiments=5, enhance_data=False, batch_size=20,
show_network_summary=True)
```

train.tar: https://drive.google.com/uc?id=1xQ0vzz43s2M7ViFksds5dWezErXVI5m_
test.tar: https://drive.google.com/uc?id=1ivTvYwZT5wlDp7fAwxAwM4NOt04LgbJx
model_train.py: https://drive.google.com/uc?id=1UU7hZvcoFGoJL-gSNh-VqQEkxHmlFUel
Extract all files and put the train and test folder within a folder called image_labels

https://github.com/OlafenwaMoses/ImageAI/

Medical Image Processing with Deep Learning

Students: Yijie Jin (City University of Hong Kong), Yifan Zheng (Chinese University of Hong Kong)
Mentors: Dr. Kwai Wong, Dr. Raymond Chan

Introduction

Why do we use deep learning in medical imaging?
Deep learning has been a tremendous success in image processing and has many applications such as image reconstruction, object detection etc. As the performance of deep neural network is reaching or even surpassing human performance, it brings possibilities to apply it to medical imaging area.

Why do we study mammograms?
Breast cancer is the most common disease for women worldwide. The mortality of breast cancer largely depends on whether the lesion could be detected at an early stage. Mammography is the process of using low-energy X-rays to examine human breast for diagnosis and screening. We need well-trained radiologists to examine the CT images traditionally, which is costly and time-consuming. Adopting computer aided detection system can accelerate the diagnosing process as well as enhancing the diagnosis accuracy.

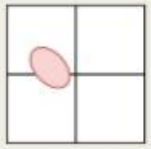
Research Objectives

- Classify mammograms into three categories
- Automatically detect tumor without prior information of the presence of a cancerous lesion

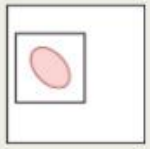
Materials and Methods

Data
We trained our neural network on mini-MIAS database of mammograms. The database contains 322 CT images (1024×1024) in total with labels. Each image is labelled as normal (healthy), benign or malignant and the label contains the coordinates of the tumor center and the radius of the tumor.

Data Augmentation
As the database has only 322 images, which is far from enough for training a neural network, we applied some data augmentation techniques before feeding the data into the neural network. The original images are cut into small image patches (128×128 and 256×256), rotated by certain degrees and flipped vertically.



More subpictures
Do not keep edges
Break tumor



Fewer subpictures
Keep edges
Do not break tumor

Cutting images centered at tumor (as in right figure) obtains the best results by now. (shown in the Performance part)

Model

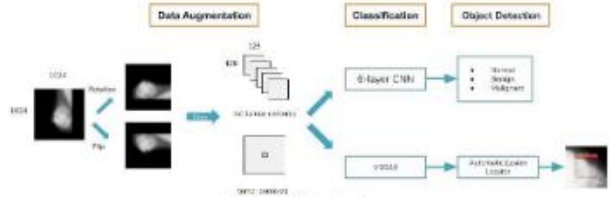


Figure1. Flowchart

Classification
We used a convolutional neural network for classification. The neural network consists of 4 convolutional layers, 1 pooling layer and 1 fully connected layer. Instead of using the usual convolutional layer, we replaced two of them with depthwise convolutional layers which could reduce the number of parameters needed to train.

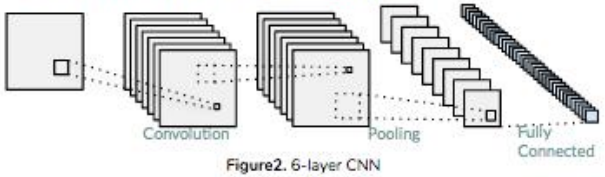


Figure2. 6-layer CNN

Automatic Lesion Detection
We use SSD (single Shot Multibox Detector) which is a popular algorithm for object detection to detect the tumor in abnormal images. We modified the model and our preprocessed images and labels are fed into the algorithm. **Figure 3** is the abstract and **Figure 4** is the whole architecture.

It is based on the VGG16 with several layers appended for automatic object detection. VGG16 is a convolutional neural network for classification which has been shown to have a remarkable performance in classification.

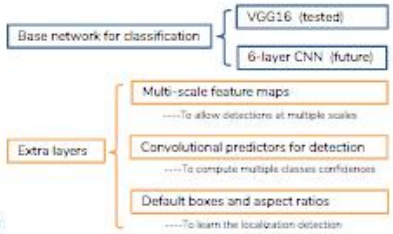


Figure3. The abstract of SSD

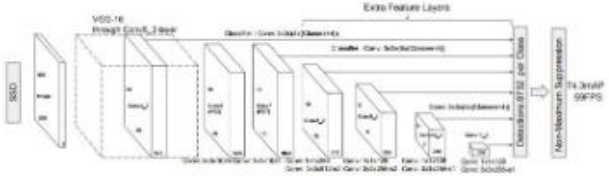
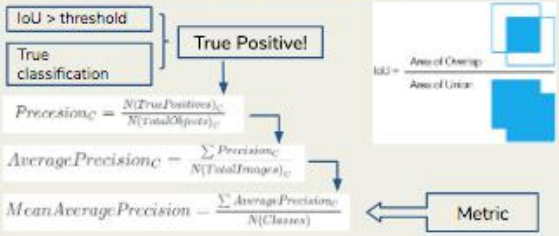


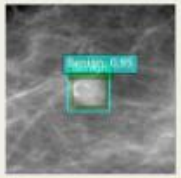
Figure4. The architecture of SSD

Performance



Test set			All sets (Train+Validation+Test)		
Benign	AP	0.8	Benign	AP	0.879
Malignant	AP	0.806	Malignant	AP	0.907
	mAP	0.803		mAP	0.902

In original SSD paper, the author performed experiment on PASCAL VOC2007 and VOC2012 databases, and the highest mAP is also around 0.8. However, there is a great gap on database sizes of their experiments(20000) and ours(700). Therefore, it is expected that with more mammogram data, better results will be achieved.



Future Work

- Use some other datasets to test our model
- Combine our classification model with SSD to decrease the computing time

Acknowledgements

This project was sponsored by the National Science Foundation through Research Experience for Undergraduates (REU) award, with additional support from the Joint Institute of Computational Sciences at University of Tennessee Knoxville. This project used allocations from the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by the National Science Foundation. In addition, the computing work was also performed on technical workstations donated by the BP High Performance Computing Team.

References

Ferlay, J., Héry, C., Autier, P. & Sankaranarayanan, R. (2010). Global burden of breast cancer. Breast cancer epidemiology, 1-19, Springer.
Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C. Y., & Berg, A. C. (2016, October). Ssd: Single shot multibox detector. European conference on computer vision (pp. 21-37). Springer, Cham.
Ting, F. F., Tan, Y. J., & Sim, K. S. (2019). Convolutional neural network improvement for breast cancer classification. Expert Systems with Applications, 120, 103-115.

Autonomous Vehicle Research Project

Students: Brendan Flood (University of Dallas)
Patrick Lau (City University of Hong Kong)
Julian Halloy (Maryville High School)
Mentors: A. Ayala (UTK), K. Wong (UTK)

Construct the Car
- Assemble the robot car
- Connect with Arduino Uno board

Setup Jetson Nano
- Install system packages and prerequisites
- Configure the development environment

Jetson Nano Car
- Connect Jetson Nano with Arduino Uno
- Send command from Nano by PySerial

Collect images
- Connect camera module to the Jetson Nano
- Drive the car and capture images for training data

Label the data
- Create folder for each object to predict
- Put the images in its respective folder

Model training
- Train by ImageAI with ResNet model
- Generate .h5 file and .json file

Translate to MAGMADNN
- Write, train, and test simple nn models in MAGMADNN

Introduction

As technology advances, image recognition software becomes more capable than ever before, to the point that a computer with such a capability can potentially be used to replace a human driver in a car. The purpose of this experiment is to design a miniature version of a self-driving car which uses a convolutional neural network to recognize images of and navigate around a closed environment.

Equipment Setup

Hardware

Software

Neural Network

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
		3×3 max pool, stride 2				
conv2_x	56×56	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

3D Model Design

Since we need to put additional components on the car, we did 3D printing designs to combine the individual parts into one piece.

Testing

Limitation

- Memory of Jetson Nano
- Data Size
- Power Supply

Data Collection

To train the network, video footage from the camera was broken down into about 10 images per second, which were sorted into network output classes based on what was in the image and how the car should react. Some images included directive signs, while others just contained the natural features of the empty hallway. An estimated **6,000** images were used to train and test the network.

Future Work

1. Collect more data for model training
2. Fine-tune the data set and thoroughly train for increased accuracy
3. Develop and test more intricate MagmaDNN networks
4. Devise new ways to store, organize, and label data

Acknowledgements

- National Science Foundation
- The Joint Institute Computational Sciences
- Extreme Science and Engineering Discovery Environment (XSEDE)

Medical Image Processing with Deep Learning

Students: Yijie Jin (City University of Hong Kong), Yifan Zheng (Chinese University of Hong Kong)
Mentors: Dr. Kwai Wong, Dr. Raymond Chan

Introduction

Why do we use deep learning in medical imaging?
Deep learning has been a tremendous success in image processing and has many applications such as image reconstruction, object detection etc. As the performance of deep neural network is reaching or even surpassing human performance, it brings possibilities to apply it to medical imaging area.

Why do we study mammograms?
Breast cancer is the most common disease for women worldwide. The mortality of breast cancer largely depends on whether the lesion could be detected at an early stage. Mammography is the process of using low-energy X-rays to examine human breast for diagnosis and screening. We need well-trained radiologists to examine the CT images traditionally, which is costly and time-consuming. Adopting computer aided detection system can accelerate the diagnosing process as well as enhancing the diagnosis accuracy.

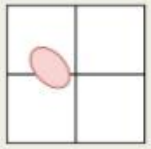
Research Objectives

- Classify mammograms into three categories
- Automatically detect tumor without prior information of the presence of a cancerous lesion

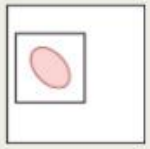
Materials and Methods

Data
We trained our neural network on mini-MIAS database of mammograms. The database contains 322 CT images (1024×1024) in total with labels. Each image is labelled as normal (healthy), benign or malignant and the label contains the coordinates of the tumor center and the radius of the tumor.

Data Augmentation
As the database has only 322 images, which is far from enough for training a neural network, we applied some data augmentation techniques before feeding the data into the neural network. The original images are cut into small image patches (128×128 and 256×256), rotated by certain degrees and flipped vertically.



More subpictures
Do not keep edges
Break tumor



Fewer subpictures
Keep edges
Do not break tumor

Cutting images centered at tumor (as in right figure) obtains the best results by now. (shown in the Performance part)

Model

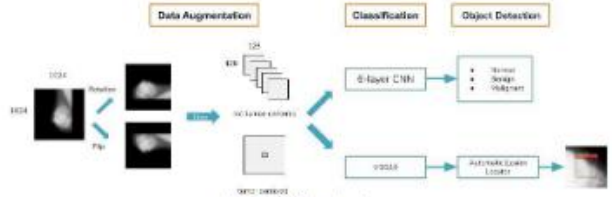


Figure1. Flowchart

Classification
We used a convolutional neural network for classification. The neural network consists of 4 convolutional layers, 1 pooling layer and 1 fully connected layer. Instead of using the usual convolutional layer, we replaced two of them with depthwise convolutional layers which could reduce the number of parameters needed to train.

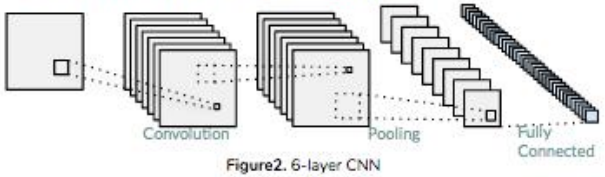


Figure2. 6-layer CNN

Automatic Lesion Detection
We use SSD (single Shot Multibox Detector) which is a popular algorithm for object detection to detect the tumor in abnormal images. We modified the model and our preprocessed images and labels are fed into the algorithm. **Figure 3** is the abstract and **Figure 4** is the whole architecture.

It is based on the VGG16 with several layers appended for automatic object detection. VGG16 is a convolutional neural network for classification which has been shown to have a remarkable performance in classification.

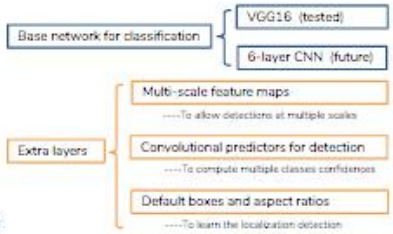


Figure3. The abstract of SSD

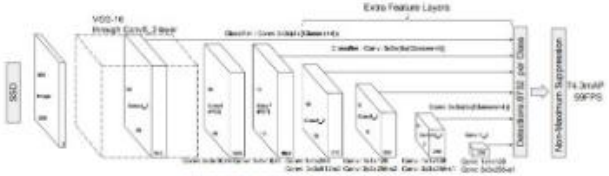
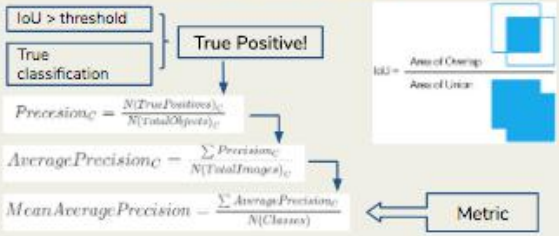


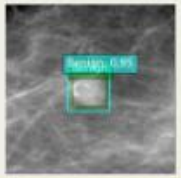
Figure4. The architecture of SSD

Performance



Test set			All sets (Train+Validation+Test)		
Benign	AP	0.8	Benign	AP	0.879
Malignant	AP	0.806	Malignant	AP	0.907
	mAP	0.803		mAP	0.902

In original SSD paper, the author performed experiment on PASCAL VOC2007 and VOC2012 databases, and the highest mAP is also around 0.8. However, there is a great gap on database sizes of their experiments(20000) and ours(700). Therefore, it is expected that with more mammogram data, better results will be achieved.



Future Work

- Use some other datasets to test our model
- Combine our classification model with SSD to decrease the computing time

Acknowledgements

This project was sponsored by the National Science Foundation through Research Experience for Undergraduates (REU) award, with additional support from the Joint Institute of Computational Sciences at University of Tennessee Knoxville. This project used allocations from the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by the National Science Foundation. In addition, the computing work was also performed on technical workstations donated by the BP High Performance Computing Team.

References

Ferlay, J., Héry, C., Autier, P. & Sankaranarayanan, R. (2010). Global burden of breast cancer. Breast cancer epidemiology, 1-19, Springer.
Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C. Y., & Berg, A. C. (2016, October). Ssd: Single shot multibox detector. European conference on computer vision (pp. 21-37). Springer, Cham.
Ting, F. F., Tan, Y. J., & Sim, K. S. (2019). Convolutional neural network improvement for breast cancer classification. Expert Systems with Applications, 120, 103-115.

Reconstruction of High Resolution Images Using Deep Transfer Learning

Tianxiang Gao - City University of Hong Kong
Zhipeng Zhu - Chinese University of Hong Kong
Mentors:
Dr. R. Archibald (ORNL),
Dr. K. Wong (UTK)

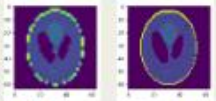
Introduction

Background:

Most current image super-resolution methods focus on single image processing with 2D CNN. Compared to 2-D CNN, 3-D CNN enables us to extract spatial and temporal correlations between consecutive frame of a video, therefore is more suitable for video image super-resolution.

Main Objective:

Train a neural network to do super-resolution on computer-generated video data, such as climate data, in the field of physics.



Data:

- Shepp-logan Phantom
  Model of a human head in the development and testing of image reconstruction algorithms.
- Image from Real Cases:
  Single images, independent to each other.
- Video from Real Cases
  Videos with main object slight changing and moving between frames, used for sequential neural network testing.
- Climate Data
  Computer-generated video based on Shallow Water Equations, which are usually used for describing the flow under a pressure surface.

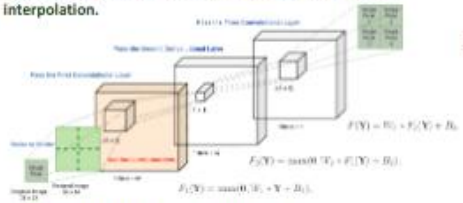
Models and Algorithms

Single Image Super-resolution (2-D CNN)

- **Basic SRCNN Model:**

Feature Extraction -> Non-linear Mapping -> Reconstruction

Preprocessing: Resize the low-resolution image to the size of the high-resolution image using bicubic interpolation.



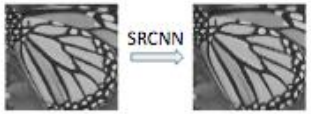
- **SRCNN Model with Transfer Learing**

Feature Extraction -> Non-linear Mapping -> Reconstruction

Freeze the Feature Extraction layer;

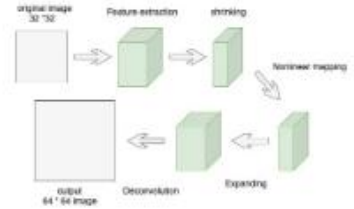
Use pre-trained parameters as initializers for the later two layers;

Fine-tune the network using our own dataset.



Single Image Super-resolution (2-D CNN)

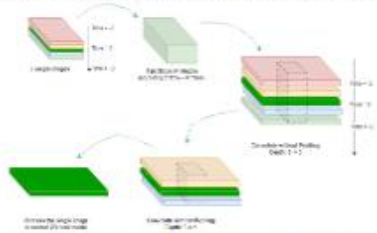
- **FSRCNN model:**
  Contrary to SRCNN model, we do not resize the input before putting them into the neural network. Instead, we put a deconvolution layer as the last layer of the network to upsample the images.
- **Since the size of input is smaller, the computational cost of FCRCNN is much smaller than that of SRCNN model.**



Sequential Image Reconstruction (3-D CNN)

- **Basic 3D SRnet Model:**

Time-domain Information Extraction -> Downgrade -> Output



Use information from pre-frames and post-frames to compensate the lost information for one frame.

Layer (type)	Output Shape	Param #
conv3d_1 (Conv3D)	(None, 5, 64, 64, 32)	896
conv3d_2 (Conv3D)	(None, 5, 64, 64, 32)	27680
conv3d_3 (Conv3D)	(None, 5, 64, 64, 32)	27680
conv3d_4 (Conv3D)	(None, 3, 62, 62, 32)	27680
conv3d_5 (Conv3D)	(None, 1, 60, 60, 32)	27680
reshape_1 (Reshape)	(None, 60, 60, 32)	0
conv2d_1 (Conv2D)	(None, 60, 60, 4)	1156

Sequential Image Reconstruction after Pre-Processing using Single Image Super-Relosution Model

- **Separate 2-D and 3-D Networks:**

SRCNN Network -> Interim Results -> 3D SRnet Network

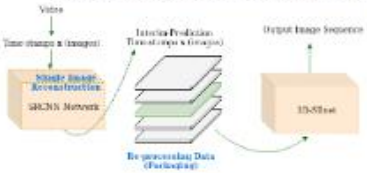
Advantage: Can set different hyper-parameter for two models;
Easy to train and implement

Disadvantage: Missing backpropogation between two networks

Do super-resolution twice:

Use SRCNN network to super-resolute the video frame-by-frame and save the interim results as the input of 3-D model.

Use 3D SRnet network to do sequential-image reconstruction, leveraging the spatial correlations between consecutive frames.



Current Problem:

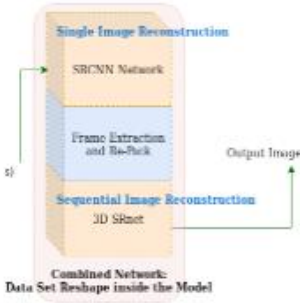
Interim results are packed for 3D processing. After putting the processed data into the second network, performance is facing a sudden drop. The combination of two neural network does not make an '1+1>2' effect as we expected.

- **Two-In-One Network:**

SRCNN Layers
+
Lambda Layer
(data repacking)
+
3D SRnet Layers

Regard SRCNN network and 3D SRnet as two layers;

Add a Lambda Layer as the bridge of SRCNN and 3D SRnet. Pack consecutive frames as a small batch, resize the output tensor from SRCNN layer, and pass the modified tensor to 3D SRnet.



Challenge:

Inside the lambda layer, we need to use input batch size as index of looping. But the number is stored to be dynamic, means we cannot get the exact number to run the code.

Performance and Analysis

	Basic SRCNN (Image from Real Cases)	SRCNN with Transfer Learning (Image from Real Cases)	Basic 3D SRnet (Video from Real Cases)
PSNR	Raw Data: 33.2460 After Processing: 33.9450	Raw Data: 33.2460 After Processing: 34.5681	Raw Data: 22.1305 After Processing: 33.1454
SSIM	Raw Data: 0.91219 After Processing: 0.92870	Raw Data: 0.91219 After Processing: 0.93395	Raw Data: 0.73280 After Processing: 0.95365
MAE	Raw Data: 0.01673 After Processing: 0.01605	Raw Data: 0.01673 After Processing: 0.01488	Raw Data: 10.58593 After Processing: 5.05191
MSE	Raw Data: 0.00093 After Processing: 0.00078	Raw Data: 0.00093 After Processing: 0.00068	Raw Data: 396.006513 After Processing: 107.77480

	SRCNN network and 3D SRnet (Climate Data)
PSNR	Raw Data: 11.9369 Interim Result: 24.2009 After Reloading: 5.4658 After Processing: 16.9059
SSIM	Raw Data: 0.2889 Interim Result: 0.8601 After Reloading: 0.0752 After Processing: 0.3452
MSE	Raw Data: 5060.8718 Interim Result: 247.2685 After Reloading:36429.17 After Processing:1325.938

- Compression methods (e.g, JPEG) and Resizing impose different loss on pictures;
- Performance of the neural network also depends on the quality of raw data;
- Hyper-parameter tuning can improve performance of neural network, but the improvement is not significant;
- Deepen the network (adding more layers) may not improve performance for particular cases;
- Sudden increase of interim result reloading may be caused by data re-packing.

PSNR: Peak Singnal-to-noise Ratio
SSIM: Structural Similarity Index
MAE: Mean Absolute Error
MSE: Mean Square Error

Future Work

Due to limit of computational capacity and lack of appropriate data, we have not done enough training on our neural network. We plan to collect more data and use them to fine tune our neural network.

Also, since we have many choices for the 2-D model and 3-D model, we believe that the structure of our neural network can also be improved.

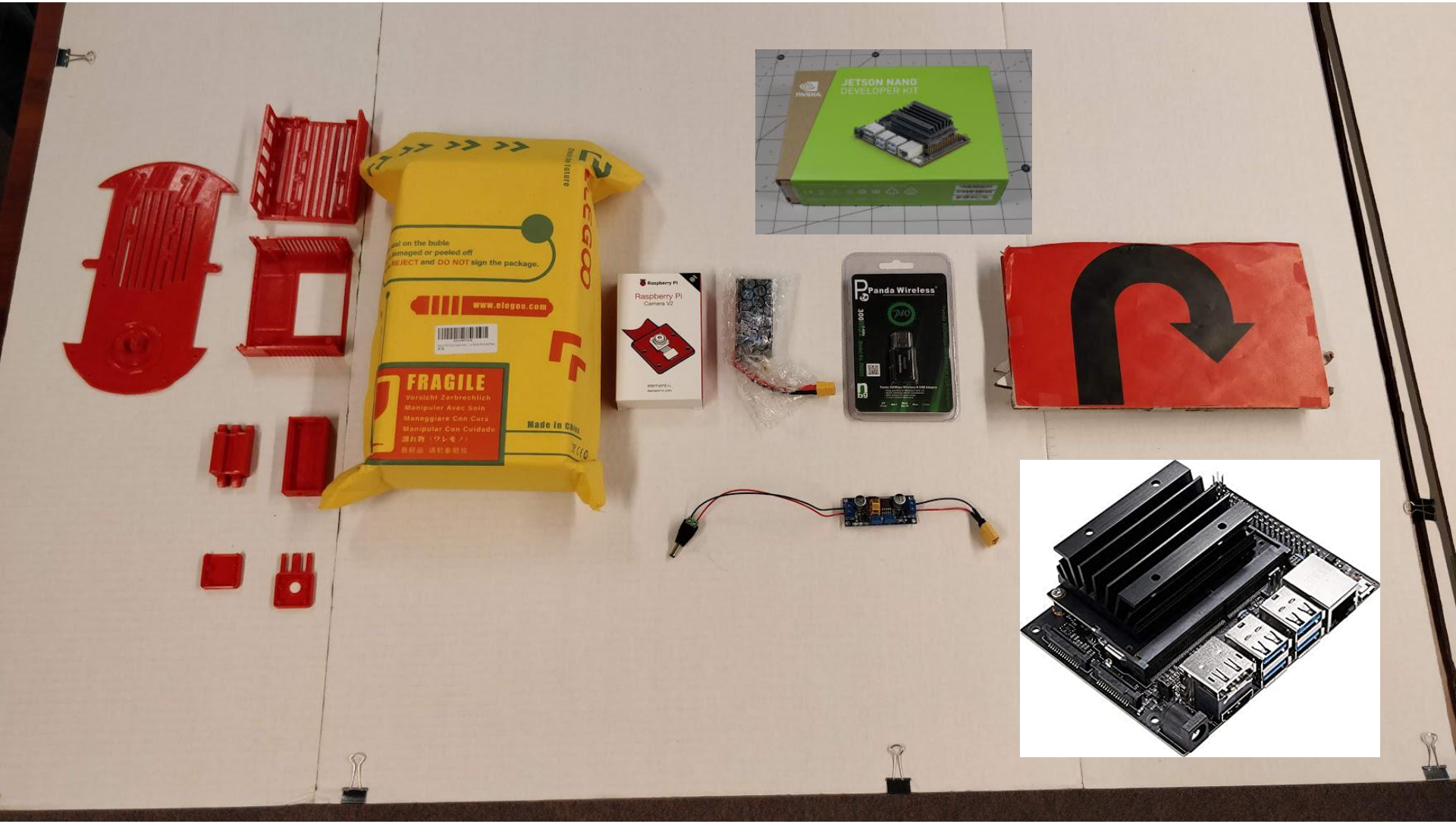
With more data, we can change the structure of our neural network, and compare their overall performance on a large dataset to get an optimized neural network.

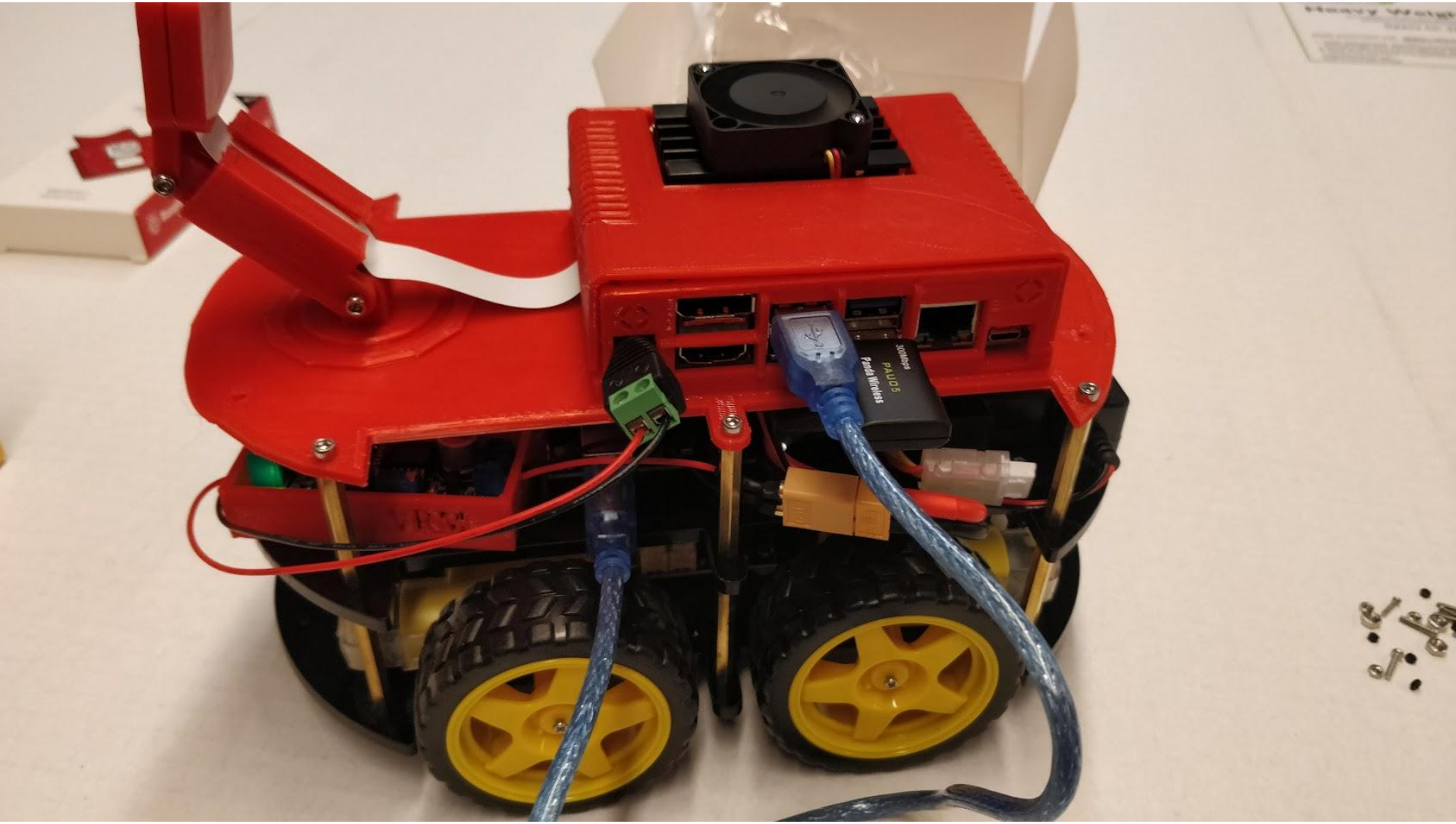
Reference and Acknowledgement

- Dong C, Loy CC, He K, Tang X. Image super-resolution using deep convolutional networks. IEEE transactions on pattern analysis and machine intelligence. 2015 Jun 1;38(2):295-307.
- Dong C, Loy CC, Tang X. Accelerating the super-resolution convolutional neural network. InEuropean conference on computer vision 2016 Oct 8 (pp. 391-407). Springer, Cham.
- Kim SY, Lim J, Na T, Kim M. 3DSRnet: Video Super-resolution using 3D Convolutional Neural Networks. arXiv preprint arXiv:1812.09079. 2018 Dec 21.

This project was sponsored by the National Science Foundation through Research Experience for Undergraduates (REU) award, with additional support from the Joint Institute of Computational Sciences at University of Tennessee Knoxville. This project used allocations from the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by the National Science Foundation. In addition, the computing work was also performed on technical workstations donated by the HP High Performance Computing Team.

Building the RC Car




300Mbps
PAU05
Panda Wireless

Autonomous Vehicle

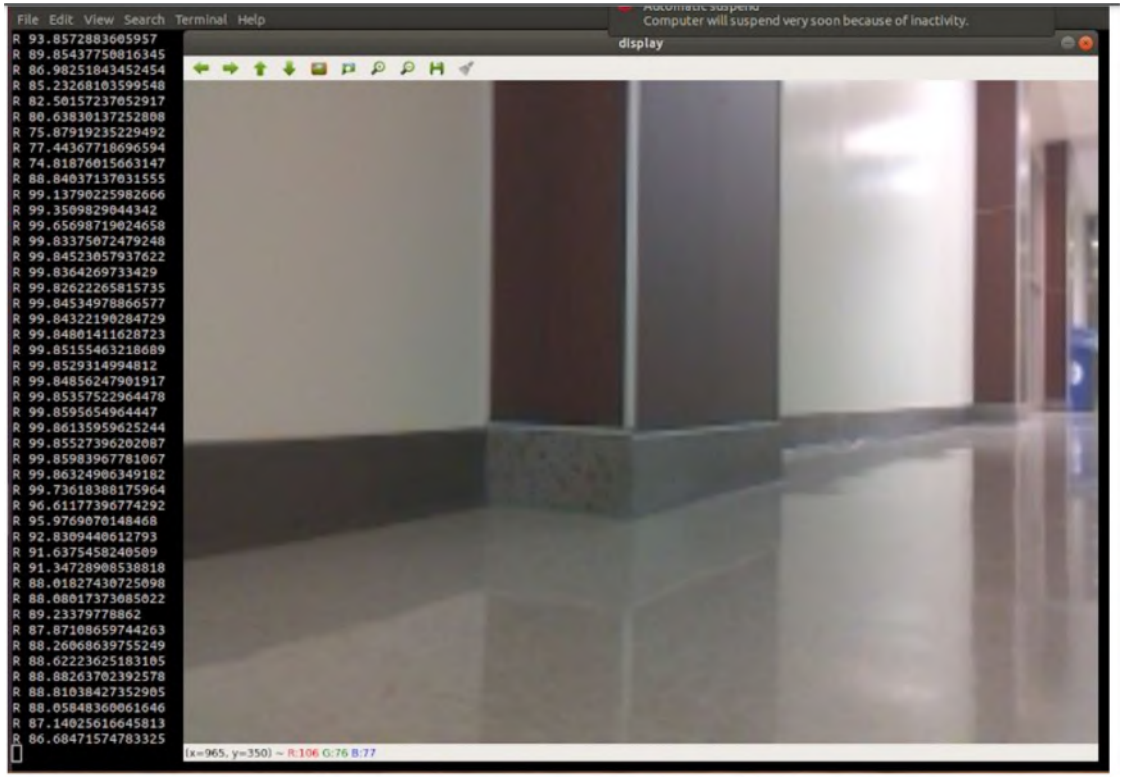
Inference on the Jetson Nano

The .h5 and .json file are stored on the Jetson Nano
The file location is input in the python code:

```
execution_path = os.getcwd()
prediction.setModelPath(os.path.join(execution_path, "7-30-test/model.h5") )
prediction.setJsonPath( os.path.join(execution_path, "7-30-test/model_class.json") )
```

Then the Jetson Nano writes an image to image.jpg, and makes a prediction based on that image

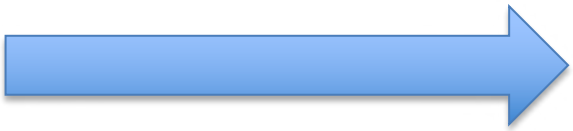
This happens multiple times per second in a loop to allow the car to drive



Running the Autonomous Vehicle

Autonomous Vehicle Inference

model.h5 model_class.json



autonomous-car.py

add location of .h5 and .json files

$ python3 autonomous-car.py

model.h5

Software Setup

Download the Jetson Nano Developer Kit SD Card Image, and note where it was saved on the computer. https://developer.nvidia.com/jetson-nano-sd-card-image

Use Etcher to write the Jetson Nano Developer Kit SD Card Image to your microSD card

Download, install, and launch Etcher. https://www.balena.io/etcher

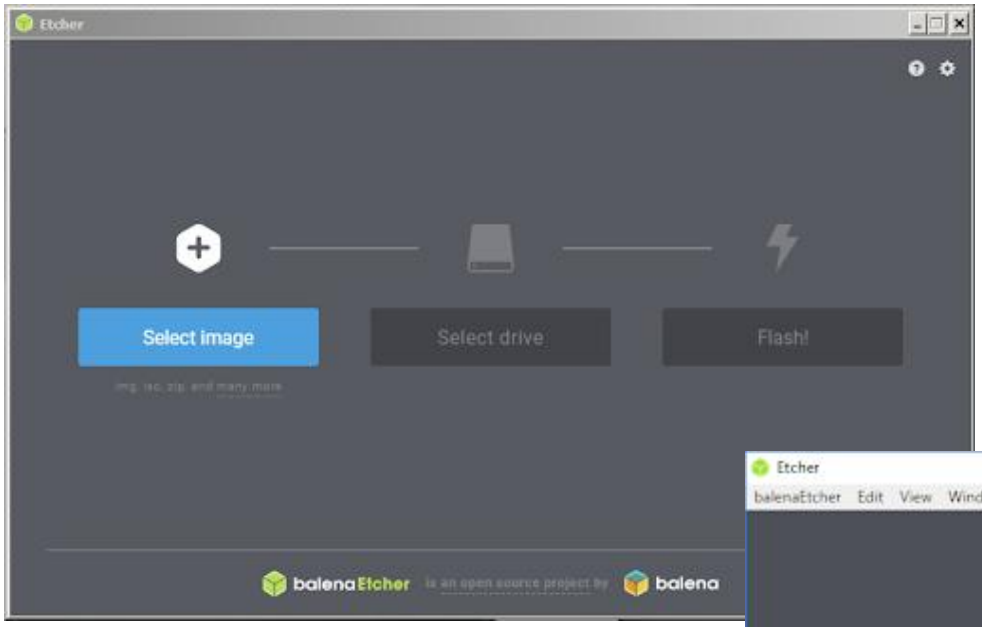
Click “Select image” and choose the zipped image file downloaded earlier.
Insert your microSD card if not already inserted.
Click Cancel (per this explanation) if Windows prompts you with a dialog like this:

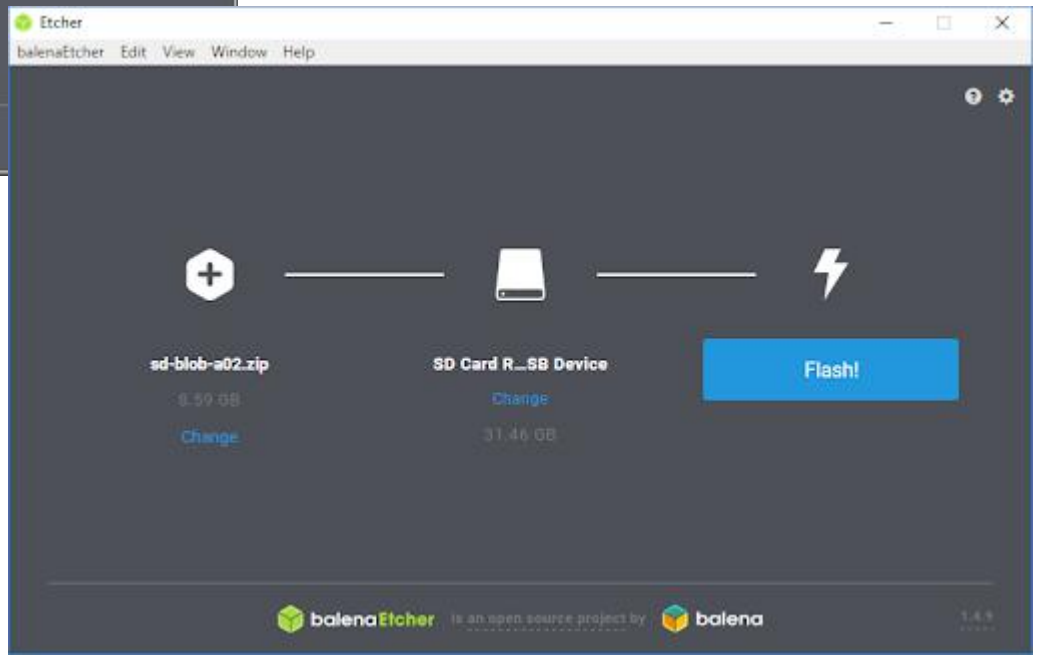
Click “Select drive” and choose the correct device.

Click “Flash!” It will take Etcher about 10 minutes to write and validate the image if your microSD card is connected via USB3.

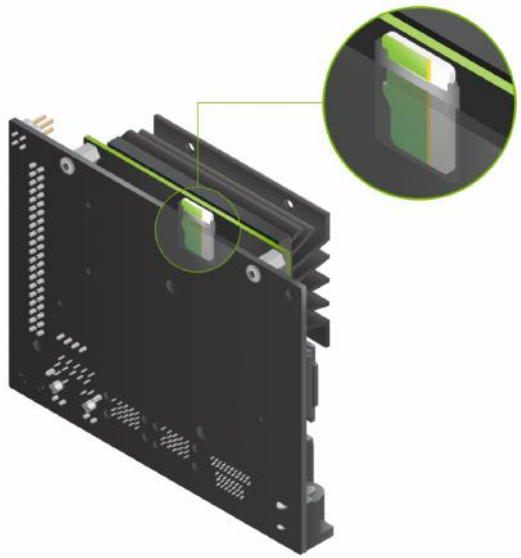
Source: https://developer.nvidia.com/embedded/learn/get-started-jetson-nano-devkit

Software Setup





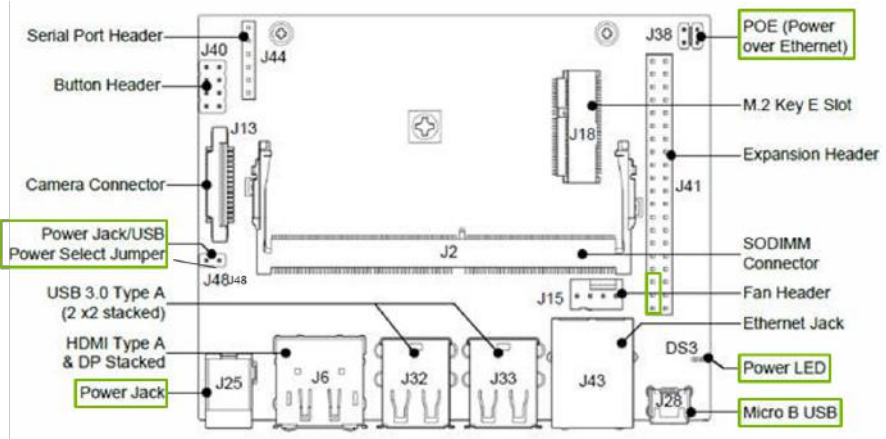
Software Setup

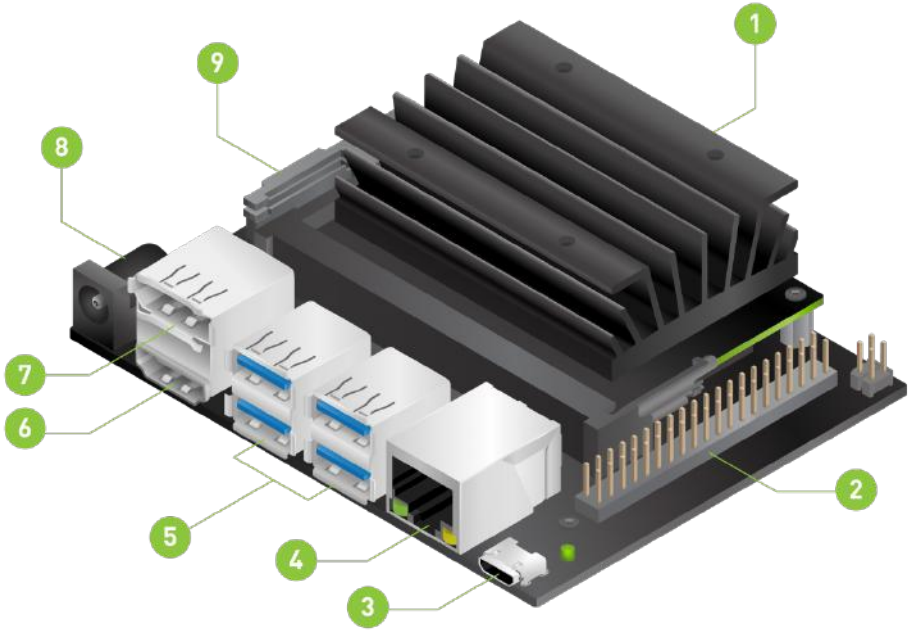


Put Micro-SD card in Jetson Nano
Connect Jetson Nano to peripherals

- Monitor
- Keyboard
- Mouse
- Power – Bridge J48

Follow prompts on screen to setup Linux





Software Setup

Jetson Nano only has 4GB of memory
A **swap file** is needed
- If sudo permissions are ever denied:
Run:

```
$ sudo -s
```

This will give the user sudo (root) permissions while retaining the current user file location (directory). Enter user's password and continue.

Then swap back to the user with:

```
$ sudo login <username>
```

Enter the user's password and continue

Software Setup

- Run:
    $ sudo fallocate -l 8G /mnt/8GB.swap
    $ sudo mkswap /mnt/8GB.swap
    $ chmod 0600 /mnt/8GB.swap
    $ sudo swapon /mnt/8GB.swap

j. Once the above is working, add the following line into /etc/fstab:
    $ vim /etc/fstab

- Press “i” key to enter insert mode.
- To the end of the file add:
    “/mnt/8GB.swap none swap sw 0 0”
- Save and Exit (in vi/vim press the “Esc” key to get out of insert mode then type “:wq” and press “Enter”)
- The process for writing to a file through the vim text editor will not change.

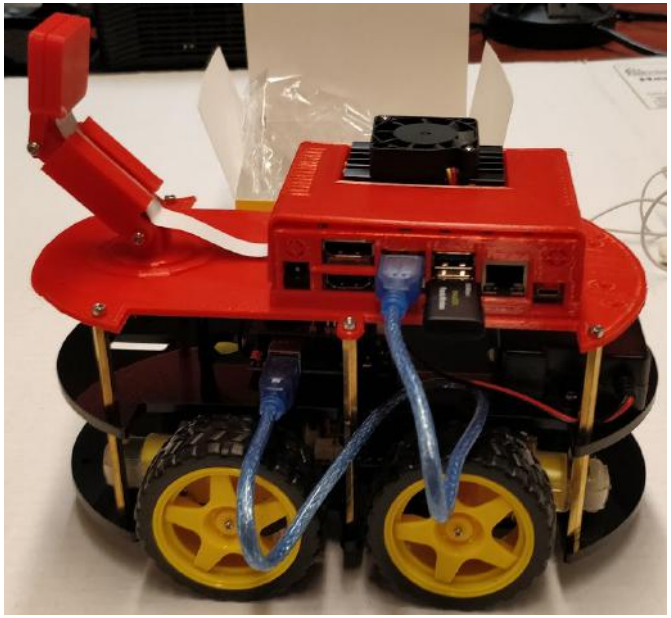
k. Reboot the system.
    $ sudo reboot

l. Make sure the swap space gets mounted automatically after reboot:
    $ sudo mkswap /mnt/8GB.swap

- A message about the swap being mounted should appear.

Software Setup

Arduino Setup
Install Arduino IDE on a separate computer from
https://www.arduino.cc/en/Main/Software
Download arduino-code.ino from Lapenna Material/References/Autonomous Vehicle/
Plug the arduino into the computer via the blue USB cable
Open the arduino-code.ino file with the Arduino IDE
Upload the code to the Arduino Uno
Plug the Arduino back into the Autonomous Vehicle



Software Setup

Jetson Nano and Arduino Setup

*Needs to be done each time you restart the Nano or plug in the arduino

$ ls –l /dev/ttyACM*

Using the correct reference for the arduino

$ sudo chmod a+rw /dev/ttyACM0

Download communicate-with-arduino.py from /Lapenna Material/References/Autonomous Vehicle/ OR git clone https://bitbucket.org/CFDL/opendnnwheel.git

If the arduino is not on port ACM0, you will need to modify the top of this code

```
with serial.Serial('/dev/ttyACM0', 9600, timeout=10) as ser:
    while True:
        …
```

Run the code:

$ python3 communicate-with-arduino.py

Software Setup

communicate-with-arduino.py
Awaits the user to input character commands

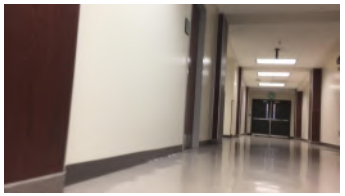
Command	Action
W	Forward
S	Backward
A	Left
D	Right
Q	Stop
X	Slow Down
Z	Speed Up
T	Return (Turn 180°)
R	Left 90°
Y	Right 90°

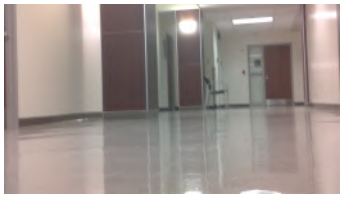
*Remember to turn on the battery on the car!

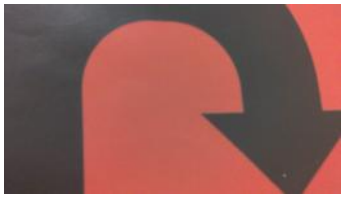
For example, if you wish to go forward, you type ‘w’ or ‘W’ and hit enter.
If wheels turn, you now have a working RC Vehicle!

Theory

Training
communicate-with-arduino.py
collect-img.py







```
es  Terminal                    Tue 10:04 ●
                                Automatic suspend
File  Edit  View  Search  Te     Computer will suspend very soon because of inactivity.
import numpy as np
import cv2
import os

def gstreamer_pipeline (capture_width=1920, capture_height=1080, display_width=1920, display_heig
ht=1080, framerate=15, flip_method=2) :
    return ('nvarguscamerasrc ! '
    'video/x-raw(memory:NVMM), '
    'width=(int)%d, height=(int)%d, '
    'format=(string)NV12, framerate=(fraction)%d/1 ! '
    'nvvidconv flip-method=%d ! '
    'video/x-raw, width=(int)%d, height=(int)%d, format=(string)BGRx ! '
    'videoconvert ! '
    'video/x-raw, format=(string)BGR ! appsink'  % (capture_width,capture_height,framerate,flip_m
ethod,display_width,display_height))


cap = cv2.VideoCapture(gstreamer_pipeline(flip_method=2), cv2.CAP_GSTREAMER)


num = 0
while os.path.exists('images{}'.format(num)):
    num += 1


# Create file path for images
dirName = 'images{}'.format(num)
os.mkdir(dirName)
print("Directory " , dirName ,  " Created ")

img_num = 0

while(cap.isOpened()):
    ret, frame = cap.read()
    if ret==True:
        cv2.imwrite('%s/image%d.jpg' % (dirName,img_num),frame)
        print('frame captured%d' % (img_num))
        img_num += 1
        # Specifies display size
        #display = cv2.resize(frame,(640,480))
        #cv2.imshow('display',display)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            print('Pressed Q')
            break
    else:
        break
-- INSERT --                                                    28,1          Top
```

```
File  Edit  View  Search  Termi
captured image465.jpg
captured image466.jpg
captured image467.jpg
captured image468.jpg
captured image469.jpg
captured image470.jpg
captured image471.jpg
captured image472.jpg
captured image473.jpg
captured image474.jpg
captured image475.jpg
captured image476.jpg
captured image477.jpg
captured image478.jpg
captured image479.jpg
captured image480.jpg
```

Theory

Training

scp <nano username>@<nano IP address>:/path/to/images/*.jpg <destination>

Images

Sort Images to classes
Compress to .tar

Turn on public link sharing
Identify code in link
“drive.google.com/file/d/<code>/view”

Images

Download using gdown <code>

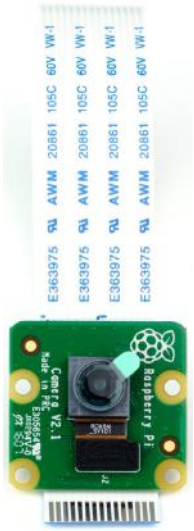
Google Drive

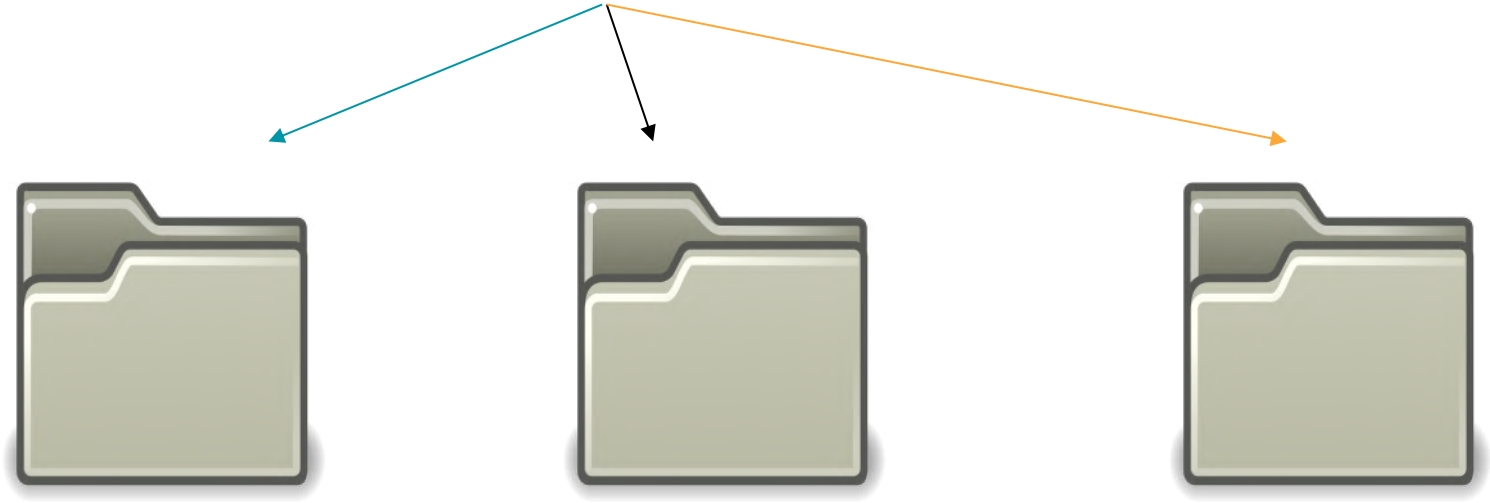
Complete Training

Autonomous Vehicle

Input / Output

- ✓ Raspberry Pi Camera - Input: 1920(px)x1080(px)x3(RGB) = 6,220,800
- ✓ Capture many images using the car. (Getting as close as possible to the environment that you will run the neural network in)
- ✓ image0.jpg, image1.jpg, … , image(n).jpg
- ✓ Create directories for each class





W
(forward)

L
(correct left)

R
(correct right)

Autonomous Vehicle

W
(forward)

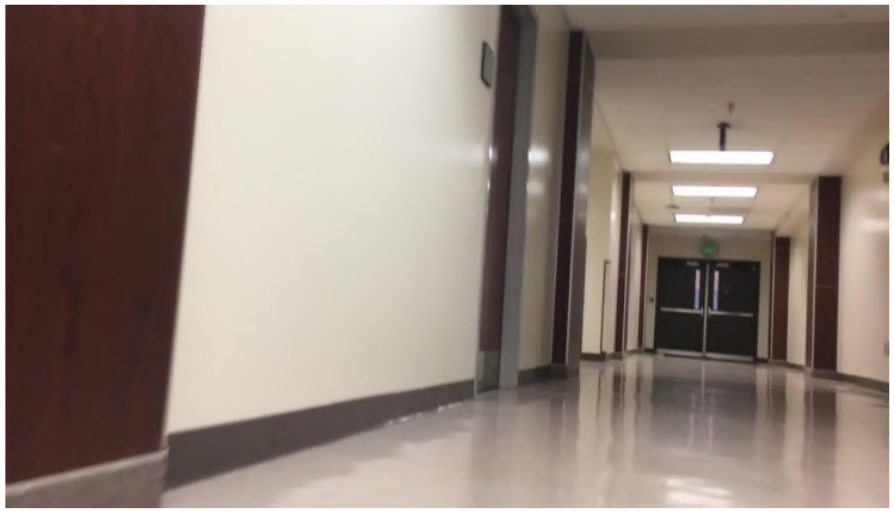
L
(correct left)

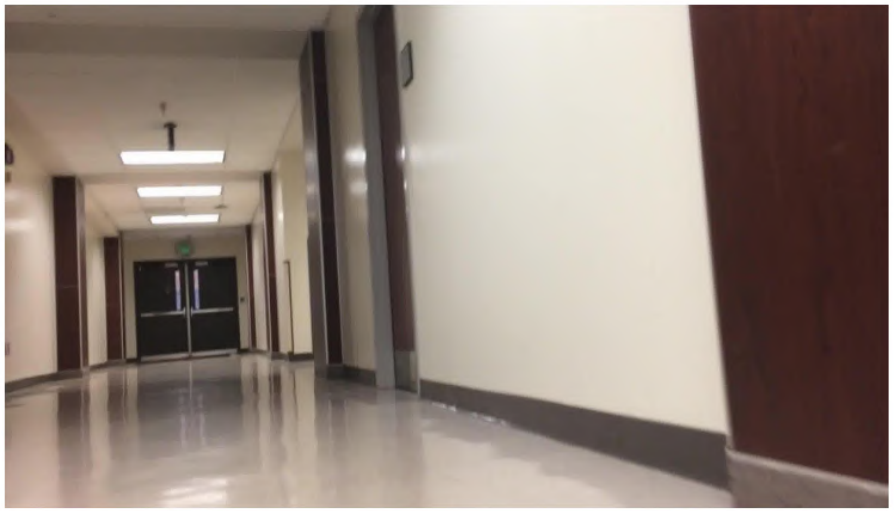
R
(correct right)

Sort images into their respective folders (time consuming)

- W – images in which the car needs to continue going straight
- L – images in which the car needs to turn left slightly to avoid the right wall
- R – images in which the car needs to turn right slightly to avoid the left wall

Data Augmentation : flip, rotate images

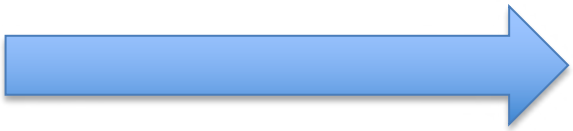




Running the Autonomous Vehicle

Autonomous Vehicle Inference

model.h5 model_class.json



autonomous-car.py

add location of .h5 and .json files

$ python3 autonomous-car.py

model.h5

Autonomous Vehicle

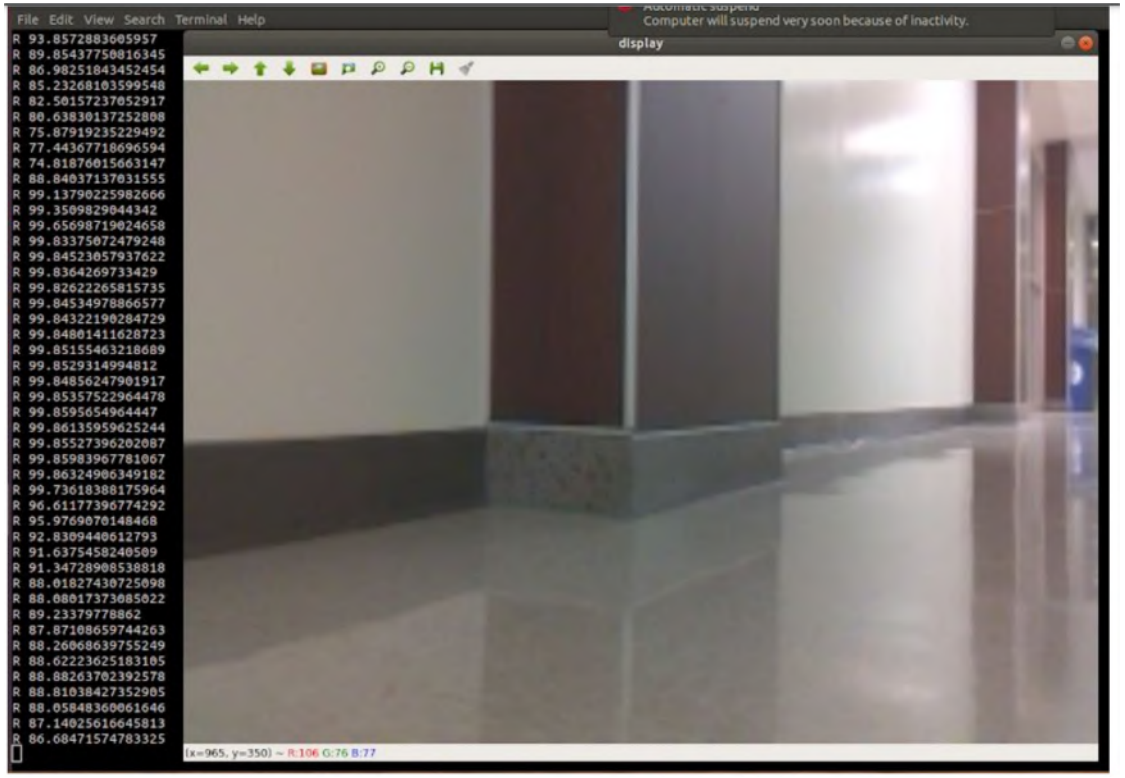
Inference on the Jetson Nano

The .h5 and .json file are stored on the Jetson Nano
The file location is input in the python code:

```
execution_path = os.getcwd()
prediction.setModelPath(os.path.join(execution_path, "7-30-test/model.h5") )
prediction.setJsonPath( os.path.join(execution_path, "7-30-test/model_class.json") )
```

Then the Jetson Nano writes an image to image.jpg, and makes a prediction based on that image

This happens multiple times per second in a loop to allow the car to drive

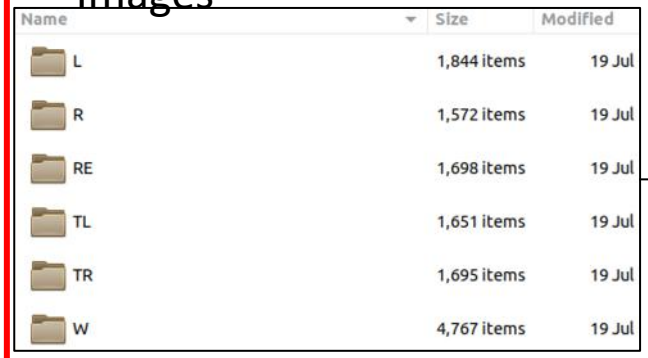


TensorRT

Training

PyTorch

Images



Train with ResNet18

train.py

Creates model tar file

model_best.pth.tar

onnx_export.py

resnet18.onnx

labels.txt - text file with class names

Inference

TensorRT

imagenet-camera.py

Prediction

L
R
RE
TL
TR
W

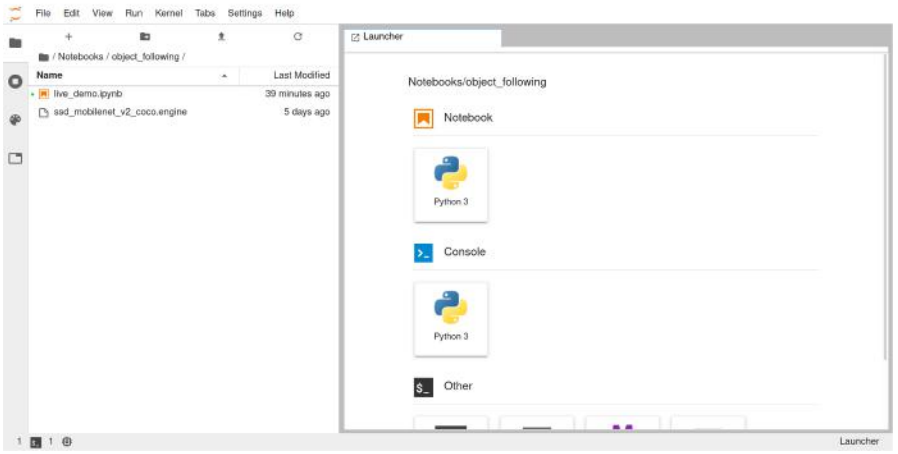
Python files are from jetson-inference github: https://github.com/dusty-nv/jetson-inference

Download Image

The Jetbot has its own SD card image that comes with all you need to run the Jetbot preinstalled. We are using the image from the jetbot.org website at: https://jetbot.org/master/software_setup/sd_card.html. We used the image listed under old releases for JetPack version 4.3 and Jetbot version 0.4.0 in order to perform object detection. You can flash this image using Balena Etcher. Simply select the image and the target sd card and flash.

You will have to connect the Jetson Nano to a monitor, keyboard and mouse in order to connect to your wifi network, but after then you will not need to. Select the wifi network to connect to and in system settings go into the network settings and set the network to be available to all users. This makes it so the Jetson Nano will connect to the network without you having to log in.

You can connect to the Jetbot by entering the ip address in the url of a computer on the same network like so: http://<ip address>:8888/. This will open Jupyter Lab on the Jetbot, the default password is jetbot.



Jetson Inference

We need to install jetson inference as it does not come with the Jetbot image.

```
$ sudo apt-get update
$ sudo apt-get install git cmake libpython3-dev python3-numpy
$ git clone --recursive https://github.com/dusty-nv/jetson-inference
$ cd jetson-inference
$ mkdir build
$ cd build
$ cmake -DENABLE_NVMM=OFF ../
$ make -j$(nproc)
$ sudo make install
$ sudo ldconfig
```

We will use jetson inference to perform object detection.

```
$ cd aarch64/bin
$ detectnet --flip-method=rotate-180
```

The first time detectnet runs it will take a while for TensorRT to build the model but afterwards it should start quickly

The “--flip-method=rotate-180” argument is optional and depends on the orientation of your camera.

https://developer.nvidia.com/blog/realtime-object-detection-in-10-lines-of-python-on-jetson-nano/
https://youtu.be/bcM5AQSAzUY
https://github.com/dusty-nv/jetson-inference/blob/master/docs/detectnet-console-2.md

Object Detection

```
import jetson.inference
import jetson.utils
import sys

net = jetson.inference.detectNet("ssd-mobilenet-v2", threshold=0.5)
camera = jetson.utils.videoSource("csi://0", argv=sys.argv)
display = jetson.utils.glDisplay()

while display.IsOpen():
    img = camera.Capture()
    detections = net.Detect(img)
    display.RenderOnce(img)
    display.SetTitle("Object Detection | Network {:.0f} FPS".format(net.GetNetworkFPS()))
```

You must have jetson inference installed. To run this code, put it in a python file and then “python3 <filename>.py”. You can flip the image with argument “--input-flip=rotate-180”.

https://developer.nvidia.com/blog/realtime-object-detection-in-10-lines-of-python-on-jetson-nano/
https://youtu.be/bcM5AQSAzUY
https://github.com/dusty-nv/jetson-inference/blob/master/docs/detectnet-console-2.md

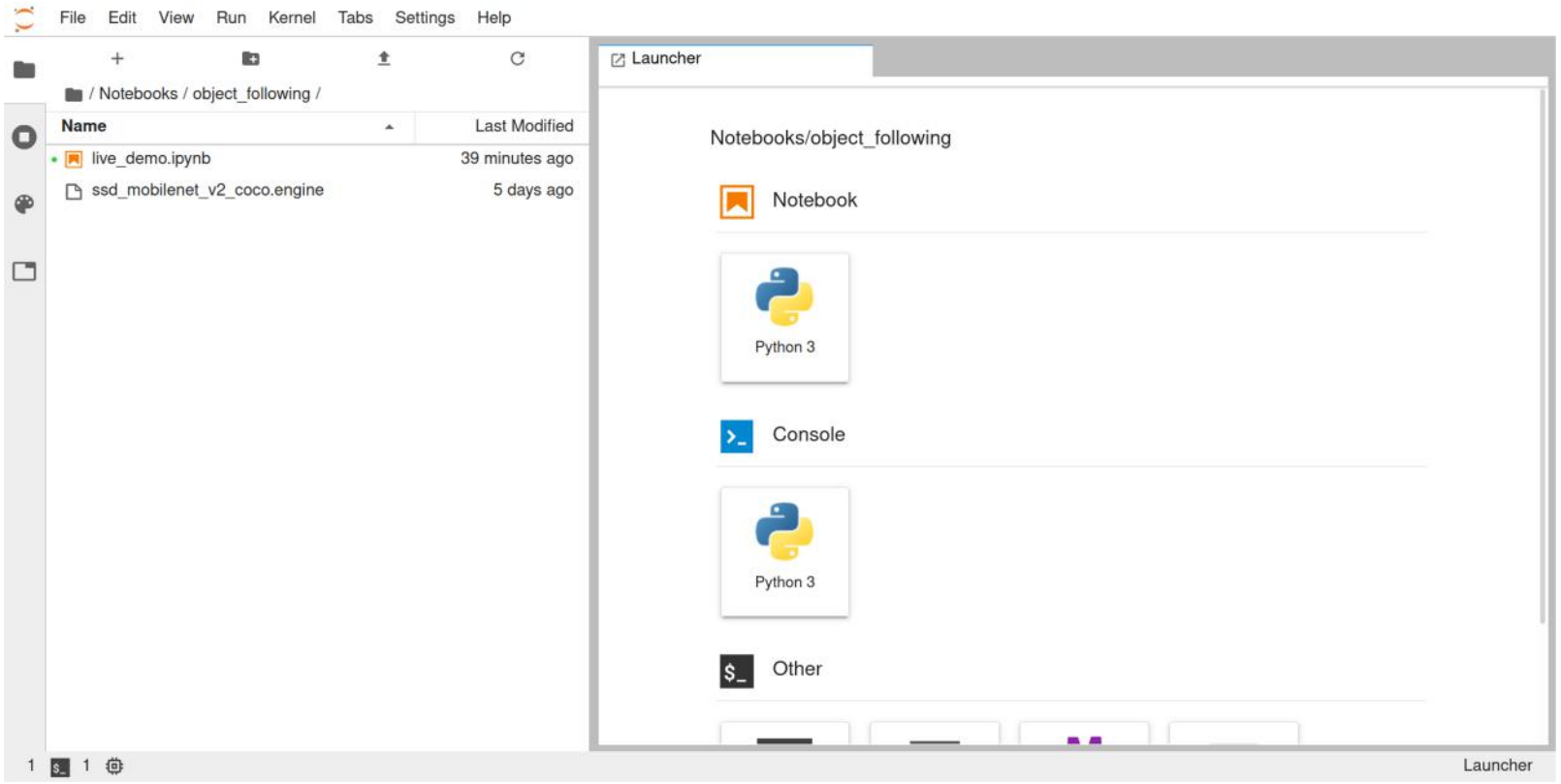
Object Detection

```
import jetson.inference
import jetson.utils
import sys

net = jetson.inference.detectNet("ssd-mobilenet-v2", threshold=0.5)
camera = jetson.utils.videoSource("csi://0", argv=sys.argv)
display = jetson.utils.glDisplay()

while display.IsOpen():
    img = camera.Capture()
    detections = net.Detect(img)
    display.RenderOnce(img)
    display.SetTitle("Object Detection | Network {:.0f} FPS".format(net.GetNetworkFPS()))
```

jetson.inference.detectNet is a class for running object detection on the Jetson Nano. We initialize it with SSD Mobilenet v2 trained on the Common Objects in Context (COCO) dataset, which has 91 classes. Then we use jetson.utils to initialize the camera and display. We create a loop where we capture an image, perform inference, and render the results on the display. This happens until the display is closed (esc key). There is a list of the classes here: https://github.com/amikelive/coco-labels/blob/master/coco-labels-paper.txt

Object Following

In order to run the object_following notebook, you need to get the model from the jetbot.org website here: https://jetbot.org/master/examples/object_following.html. Download the model for Jetbot version 0.4. Then you can scp this model to the jetbot with: `scp path/to/ssd_mobilenet_v2_coco.engine jetbot@<ip address>:~/Notebooks/object_following/`
Then you can connect to the jetbot with Jupyter Lab at http://<ip address>:8888/ (password; jetbot). Navigate to ~/Notebooks/object_following and follow the Jupyter notebook.



Object Detection

```
import jetson.inference
import jetson.utils
import sys

net = jetson.inference.detectNet("ssd-mobilenet-v2", threshold=0.5)
camera = jetson.utils.videoSource("csi://0", argv=sys.argv)
display = jetson.utils.glDisplay()

while display.IsOpen():
    img = camera.Capture()
    detections = net.Detect(img)
    display.RenderOnce(img)
    display.SetTitle("Object Detection | Network {:.0f} FPS".format(net.GetNetworkFPS()))
```

You must have jetson inference installed. To run this code, put it in a python file and then “python3 <filename>.py”. You can flip the image with argument “--input-flip=rotate-180”.

https://developer.nvidia.com/blog/realtime-object-detection-in-10-lines-of-python-on-jetson-nano/
https://youtu.be/bcM5AQSAzUY
https://github.com/dusty-nv/jetson-inference/blob/master/docs/detectnet-console-2.md

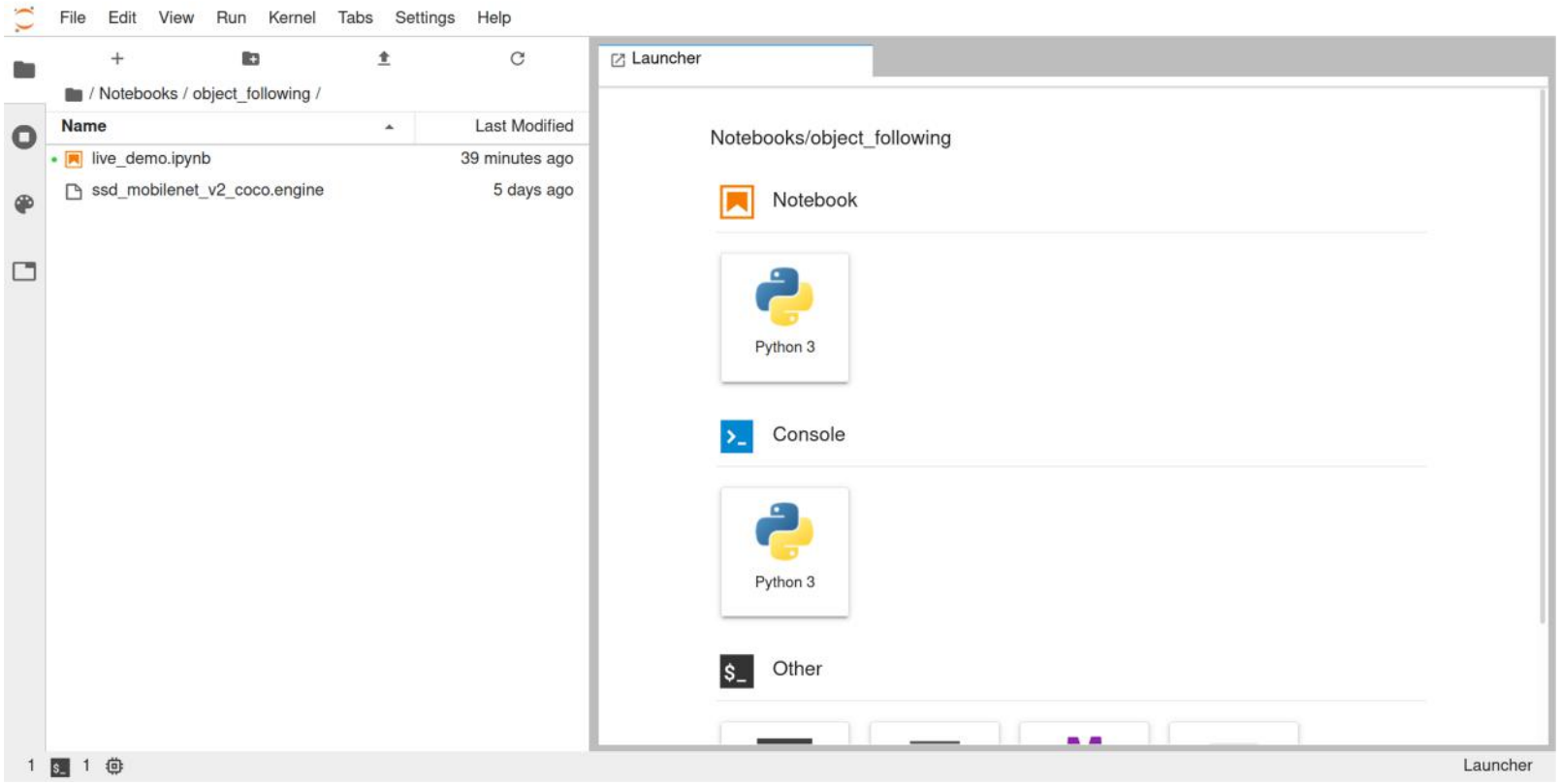
Object Detection

```
import jetson.inference
import jetson.utils
import sys

net = jetson.inference.detectNet("ssd-mobilenet-v2", threshold=0.5)
camera = jetson.utils.videoSource("csi://0", argv=sys.argv)
display = jetson.utils.glDisplay()

while display.IsOpen():
    img = camera.Capture()
    detections = net.Detect(img)
    display.RenderOnce(img)
    display.SetTitle("Object Detection | Network {:.0f} FPS".format(net.GetNetworkFPS()))
```

jetson.inference.detectNet is a class for running object detection on the Jetson Nano. We initialize it with SSD Mobilenet v2 trained on the Common Objects in Context (COCO) dataset, which has 91 classes. Then we use jetson.utils to initialize the camera and display. We create a loop where we capture an image, perform inference, and render the results on the display. This happens until the display is closed (esc key). There is a list of the classes here: https://github.com/amikelive/coco-labels/blob/master/coco-labels-paper.txt

Object Following

In order to run the object_following notebook, you need to get the model from the jetbot.org website here: https://jetbot.org/master/examples/object_following.html. Download the model for Jetbot version 0.4. Then you can scp this model to the jetbot with: `scp path/to/ssd_mobilenet_v2_coco.engine jetbot@<ip address>:~/Notebooks/object_following/`
Then you can connect to the jetbot with Jupyter Lab at http://<ip address>:8888/ (password; jetbot). Navigate to ~/Notebooks/object_following and follow the Jupyter notebook.



Topics of Unit 11

- ✓ **UNIT 11 : DNN Computing**
  - **DNN Principles: Mathematics**
  - **CNN Model: ResNet**
  - **DNN Computing: Essential Practices**
  - **DNN Usage: I/O, Transfer Learning**
  - **CNN Applications: Unet, Object Detection, Autonomous Vehicle**
  - **Advanced Topics: GAN, Reinforcement Learning**

Content of Unit 11

- ✓ **Generative Adversarial Networks (GANs)**
- ✓ **Reinforcement Learning**

GANs

Discriminative vs Generative Models

Discriminative Model: Learn a probability distribution p(y|x) → Assign labels to data; Feature learning (supervised)

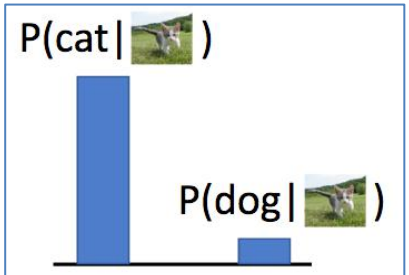
Generative Model: Learn a probability distribution p(x) → Detect outliers; Feature learning (unsupervised); Sample to **generate** new data

Conditional Generative Model: Learn p(x|y) → Assign labels, while rejecting outliers!; Generate new data conditioned on input labels

Density Function
p(x) assigns a positive number to each possible x; higher numbers mean x is more likely

Data: x

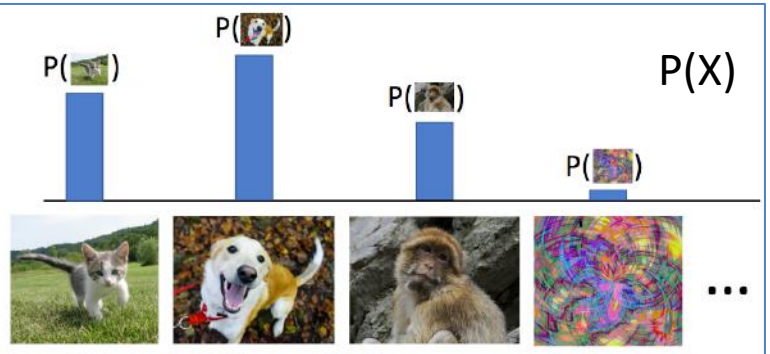
P(y|x)



Density functions are **normalized**:

$$\int_X p(x)dx = 1$$

Different values of x **compete** for density



Generative model: All possible images compete with each other for probability mass

Recall **Bayes' Rule:**

$$P(x \mid y) = \frac{P(y \mid x)}{P(y)} P(x)$$

Conditional Generative Model

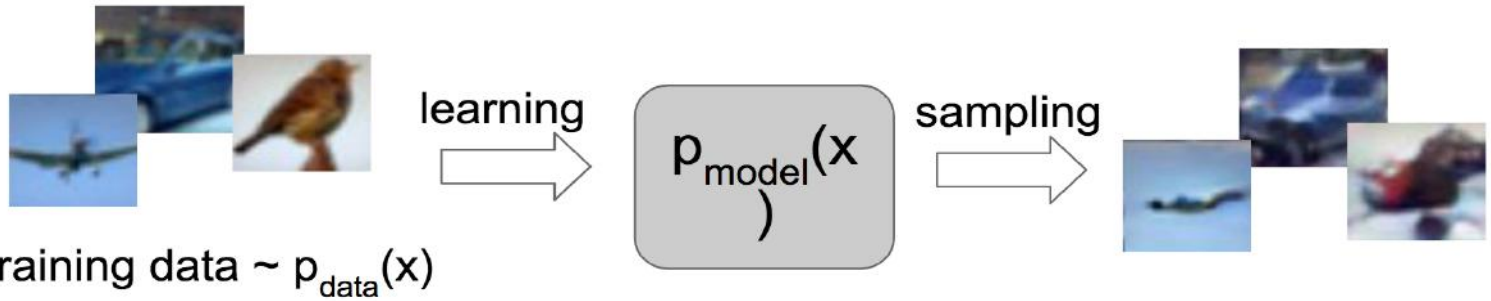
Discriminative Model

Prior over labels

(Unconditional) Generative Model

Generative Network

Given training data, generate new samples from same distribution



Objectives:
1. Learn $P_{model}(x)$ that approximates $P_{data}(x)$
2. Sampling new x from $P_{model}(x)$

- ✓ Generative model : Take input samples to Learn a model distribution and get a density estimation
- ✓ To find outlier and uncover important feature
- ✓ Learn the latent model with smaller dimension

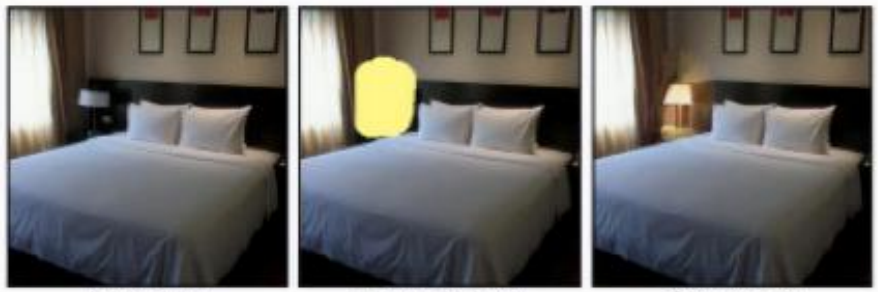
Formulate as density estimation problems:

- ✓ Explicit density estimation: explicitly define and solve for pmodel(x)
- ✓ Implicit density estimation: learn model that can sample from pmodel(x) without explicitly defining it.

- ✓ Realistic samples for artwork, super-resolution, colorization, etc.
- ✓ Learn useful features for downstream tasks such as classification.
- ✓ Getting insights from high-dimensional data (physics, medical imaging, etc.)
- ✓ Modeling physical world for simulation and planning (robotics and reinforcement learning applications)-Many more ...

Generative Adversarial Nets - GANs

- ✓ In a seminal 2014 research paper simply titled "Generative Adversarial Nets," Goodfellow and colleagues describe the first working implementation of a generative model based on adversarial networks.
- ✓ Generative Adversarial Nets (GANs) consist of two parts: generators and discriminators. The generator model produces synthetic examples (e.g., images) from random noise sampled using a distribution, which along with real examples from a training data set are fed to the discriminator, which attempts to distinguish between the two. Both the generator and discriminator improve in their respective abilities until the discriminator is unable to tell the real examples from the synthesized examples with better than the 50% accuracy expected of chance.
- ✓ GANs could produce media synthesis, like deepfakes, which is media that takes a person in existing media and replaces them with someone else's likeness.
- ✓ StyleGAN, a model Nvidia developed, has generated high-resolution head shots of fictional people by learning attributes like facial pose, freckles, and hair. StyleGAN 2 makes improvements with respect to both architecture and training methods, redefining the state of the art in terms of perceived quality.
- ✓ GANs train in an unsupervised fashion, meaning that they infer the patterns within data sets without reference to known, labeled, or annotated outcomes. Interestingly, the discriminator's work informs that of the generator — every time the discriminator correctly identifies a synthesized work, it tells the generator how to tweak its output so that it might be more realistic in the future.

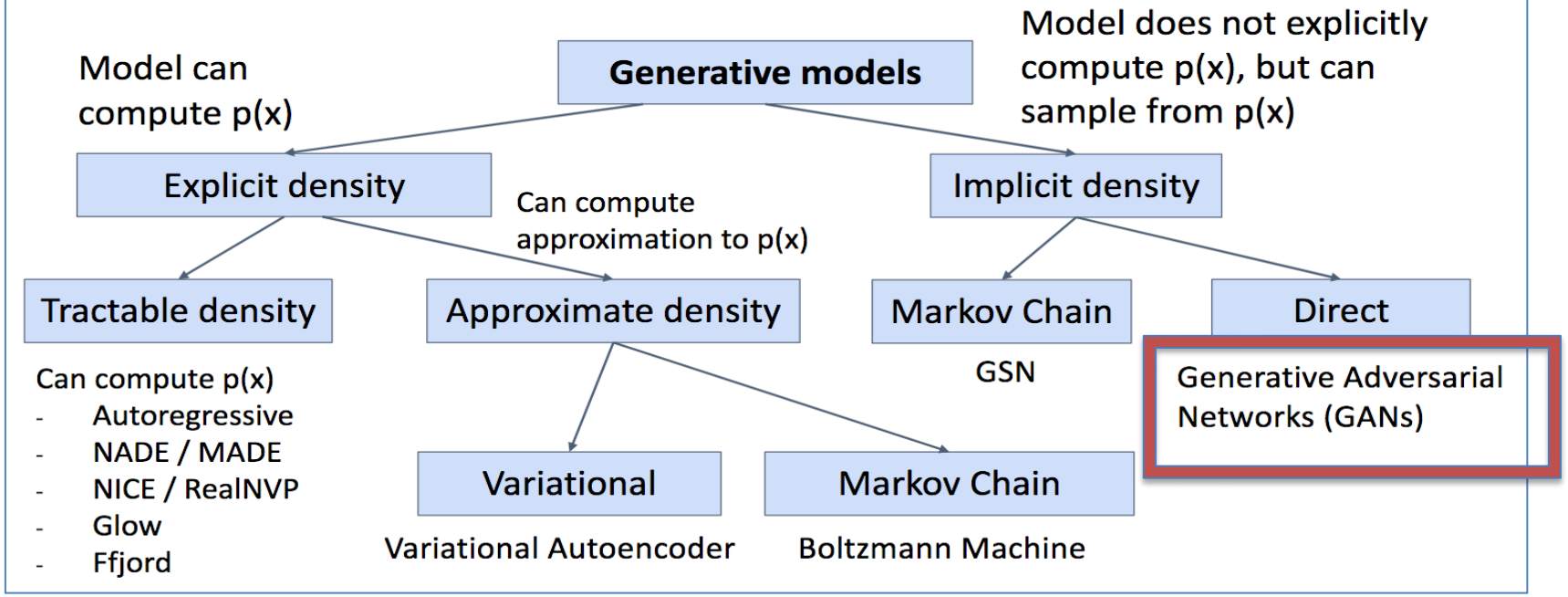


Overview

- ✓ In practice, GANs suffer from a number of shortcomings owing to their architecture. The simultaneous training of generator and discriminator models is inherently unstable. Sometimes the parameters oscillate or destabilize, which isn't surprising given that after every parameter update, the nature of the optimization problem being solved changes.
- ✓ The generator and discriminator also run the risk of overpowering each other. If the generator becomes too accurate, it'll exploit weaknesses in the discriminator that lead to undesirable results, whereas if the discriminator becomes too accurate, it'll impede the generator's progress toward convergence.
- ✓ A lack of training data also threatens to impede GANs' progress in the semantic realm, which in this context refers to the relationships among objects. Today's best GANs struggle to reconcile the difference between palming and holding an object, for example — a differentiation most humans make in seconds.
- ✓ In June 2019, Microsoft researchers detailed ObjGAN, a novel GAN that could understand captions, sketch layouts, and refine the details based on the wording. The coauthors of a related study proposed a system — StoryGAN — that synthesizes storyboards from paragraphs.
- ✓ GANs have been applied to the problems of super-resolution (image upsampling) and pose estimation (object transformation). Tang says one of his teams used GANs to train a model to upscale 200-by-200-pixel satellite imagery to 1,000 by 1,000 pixels, and to produce images that appear as though they were captured from alternate angles.
- ✓ Scientists at Carnegie Mellon last year demoed Recycle-GAN, a data-driven approach for transferring the content of one video or photo to another. When trained on footage of human subjects, the GAN generated clips that captured subtle expressions like dimples and lines that formed when subjects smiled and moved their mouths.

Generative Networks

Taxonomy of Generative Models



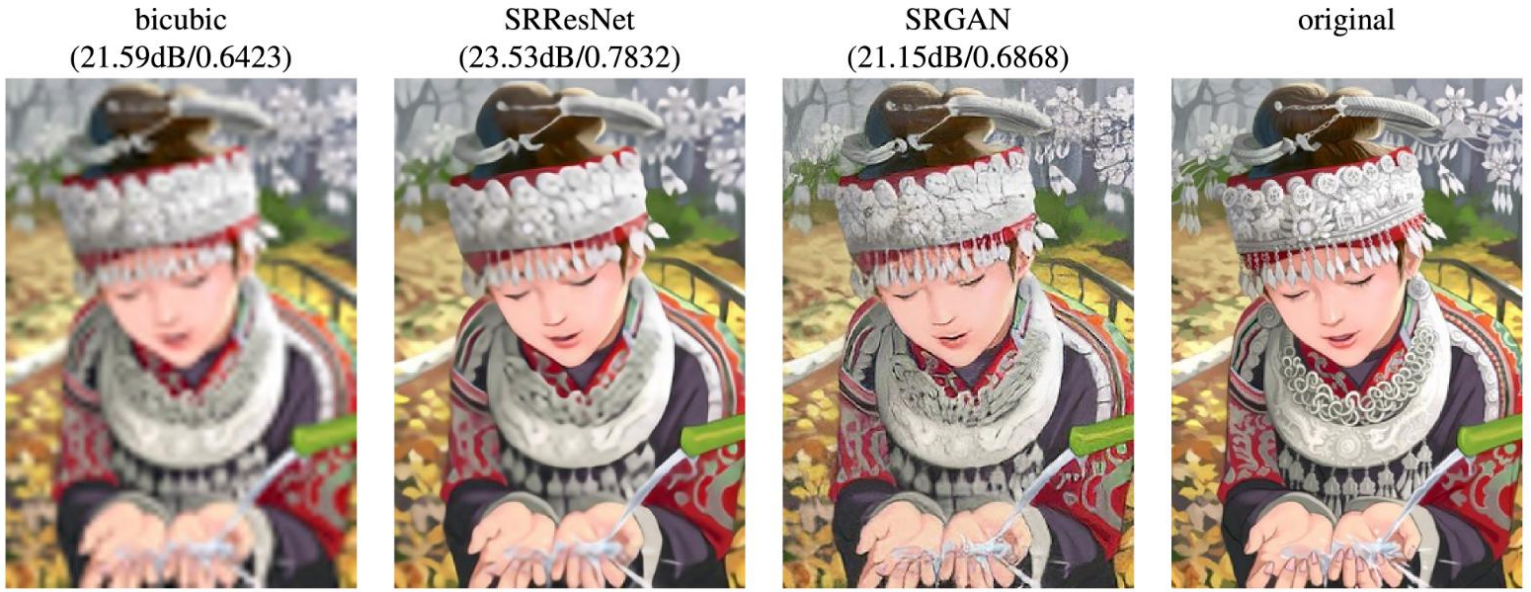
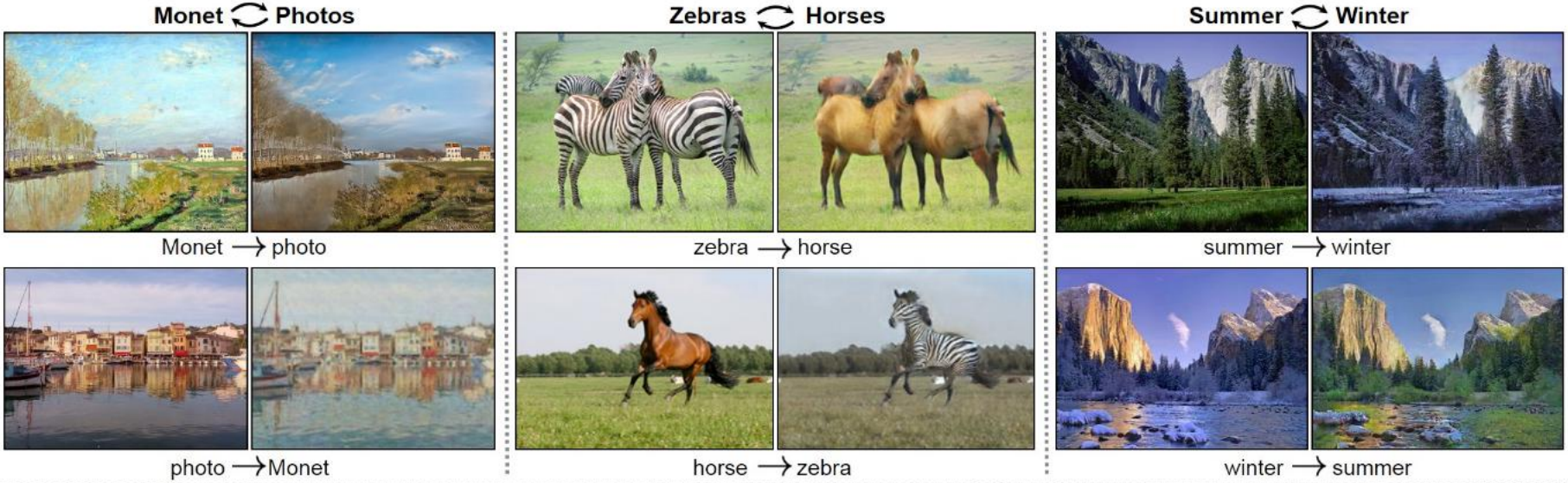
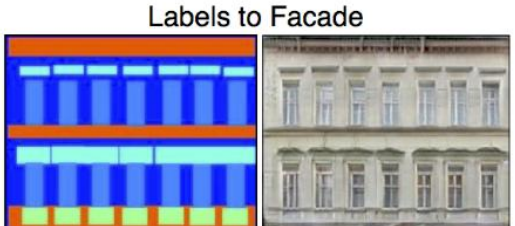
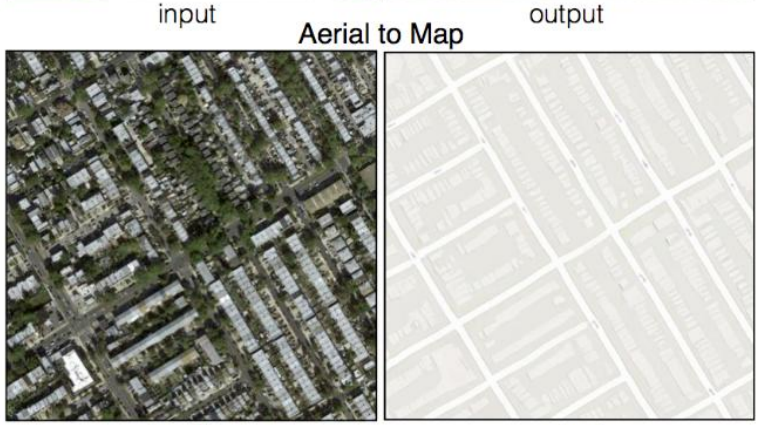


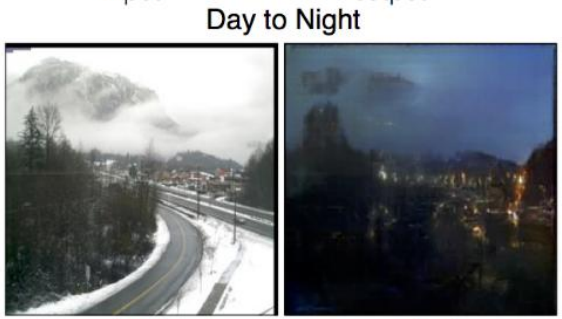
Image Super-Resolution: Low-Res to High-Res

CycleGAN & Pix2Pix

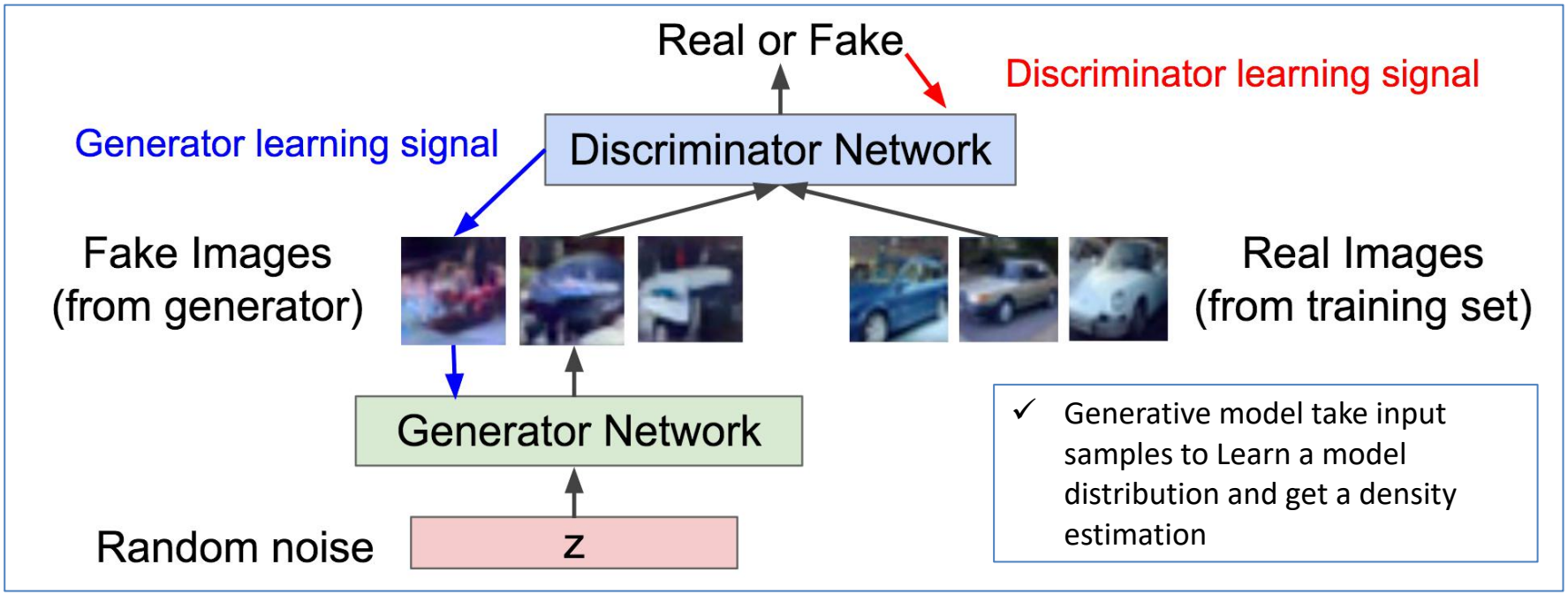








Generative Adversarial Networks



✓ Generative model take input samples to Learn a model distribution and get a density estimation

Generative Adversarial Networks: Training Objective

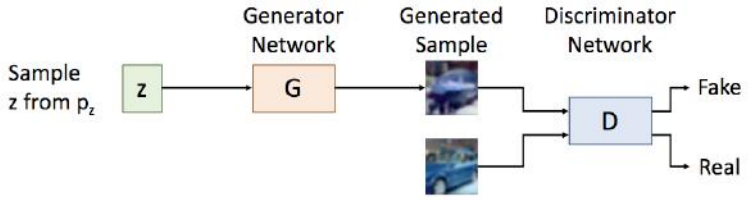
Jointly train generator G and discriminator D with a **minimax game**

Discriminator wants D(x) = 1 for real data

Discriminator wants D(x) = 0 for fake data

$$\min_G \max_D \left(E_{x \sim p_{data}} [\log D(x)] + E_{z \sim p(z)} \left[\log \left(1 - D(G(z)) \right) \right] \right)$$

Generator wants D(x) = 1 for fake data



A pair of competing networks gaming against each other

Generator network: try to fool the discriminator by generating real-looking images

Discriminator network: try to distinguish between real and fake images

512 x 384 cars

1024 x 1024 faces

INNOVATIVE COMPUTING LABORATORY

JICS Joint Institute for Computational Sciences

Smiling woman

Neutral woman

Neutral man

Samples from the model

Average Z vectors, do arithmetic

Smiling Man

GAN

http://www.offconvex.org/2020/07/06/GAN-min-max/
https://towardsdatascience.com/generative-adversarial-networks-explained-34472718707a

Putting it together: GAN training algorithm

for number of training iterations **do**

for k steps **do**

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
- Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D_{\theta_d}(x^{(i)}) + \log(1 - D_{\theta_d}(G_{\theta_g}(z^{(i)}))) \right]$$

end for

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Update the generator by ascending its stochastic gradient (improved objective):

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log(D_{\theta_d}(G_{\theta_g}(z^{(i)})))$$

end for

Some find k=1 more stable, others use k > 1, no best rule.

Followup work (e.g. Wasserstein GAN, BEGAN) alleviates this problem, better stability!

Arjovsky et al. "Wasserstein gan." arXiv preprint arXiv:1701.07875 (2017)
Berthelot, et al. "Began: Boundary equilibrium generative adversarial networks." arXiv preprint arXiv:1703.10717 (2017)

Minimax objective function: Real Data fake Data

Discriminator $\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$

a game-theoretic approach

Alternate between:

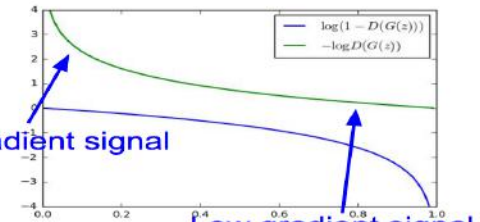
1. **Gradient ascent** on discriminator

$$\max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

2. **Instead: Gradient ascent** on generator, different objective

$$\max_{\theta_g} \mathbb{E}_{z \sim p(z)} \log(D_{\theta_d}(G_{\theta_g}(z)))$$

Instead of minimizing likelihood of discriminator being correct, now maximize likelihood of discriminator being wrong.
Same objective of fooling discriminator, but now higher gradient signal for bad samples => works much better! Standard in practice.

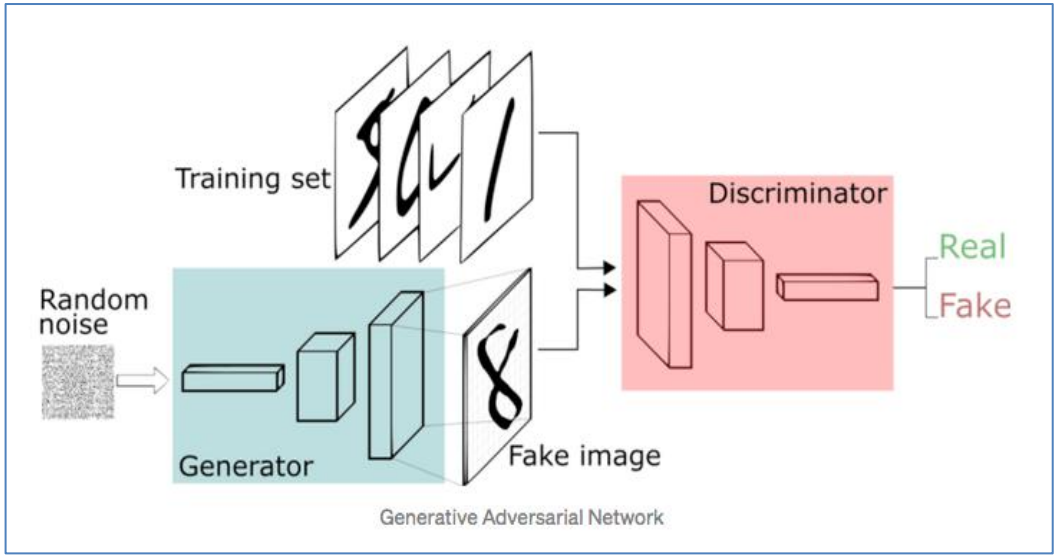


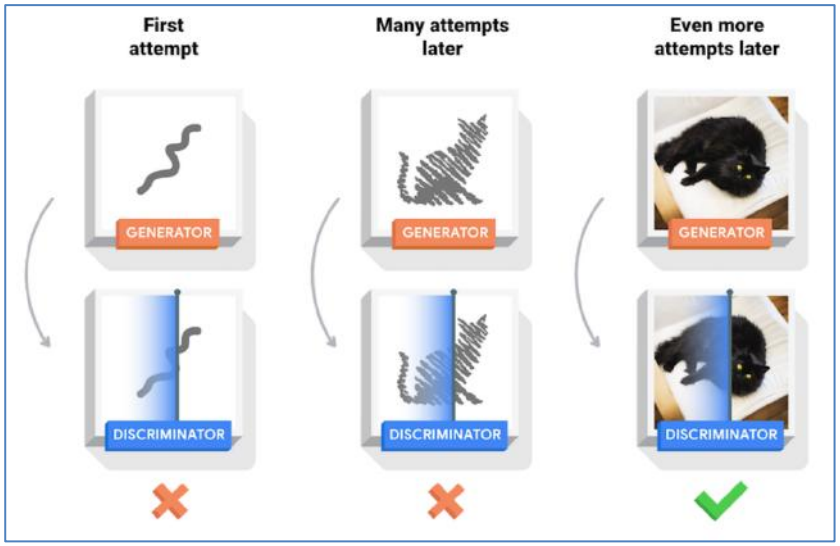
DeepDream – Inceptionism

- ✓ **DeepDream** is an artistic image-modification technique that uses the representations learned by convolutional neural networks.
- ✓ **DeepDream** is a computer vision program created by Google engineer Alexander Mordvintsev that uses a convolutional neural network to find and enhance patterns in images, creating a dream-like hallucinogenic appearance in the deliberately over-processed images.[1]
- ✓ DeepDream is an experiment that visualizes the patterns learned by a neural network. Similar to when a child watches clouds and tries to interpret random shapes, DeepDream over-interprets and enhances the patterns it sees in an image.
- ✓ It does so by forwarding an image through the network, then calculating the gradient of the image with respect to the activations of a particular layer. The image is then modified to increase these activations, enhancing the patterns seen by the network, and resulting in a dream-like image. This process was dubbed "Inceptionism"

https://www.tensorflow.org/tutorials/generative/deepdream

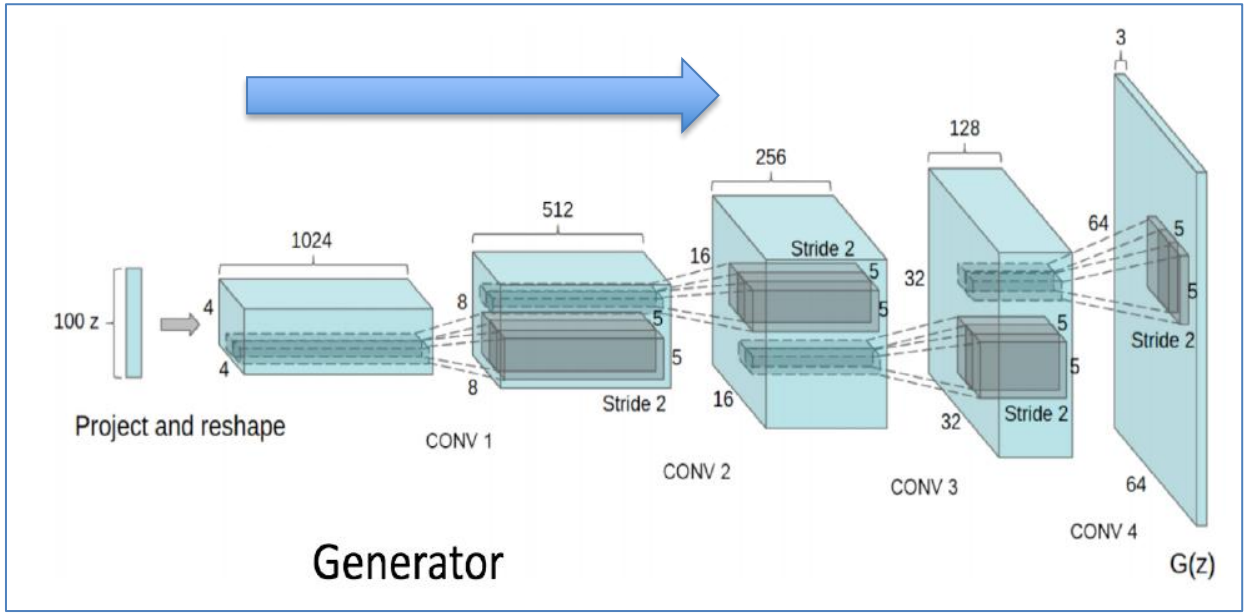
GANs

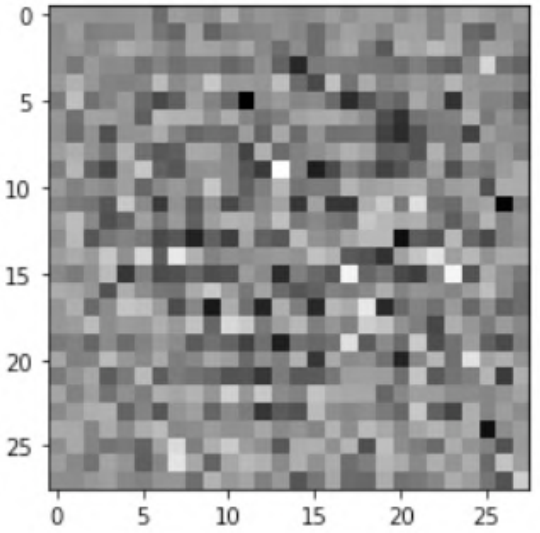


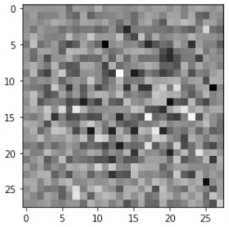


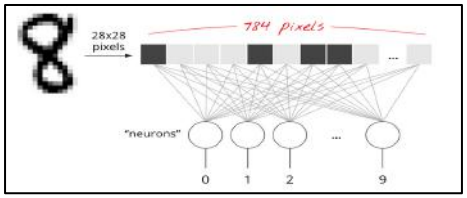
Generated samples

DCGAN Generator

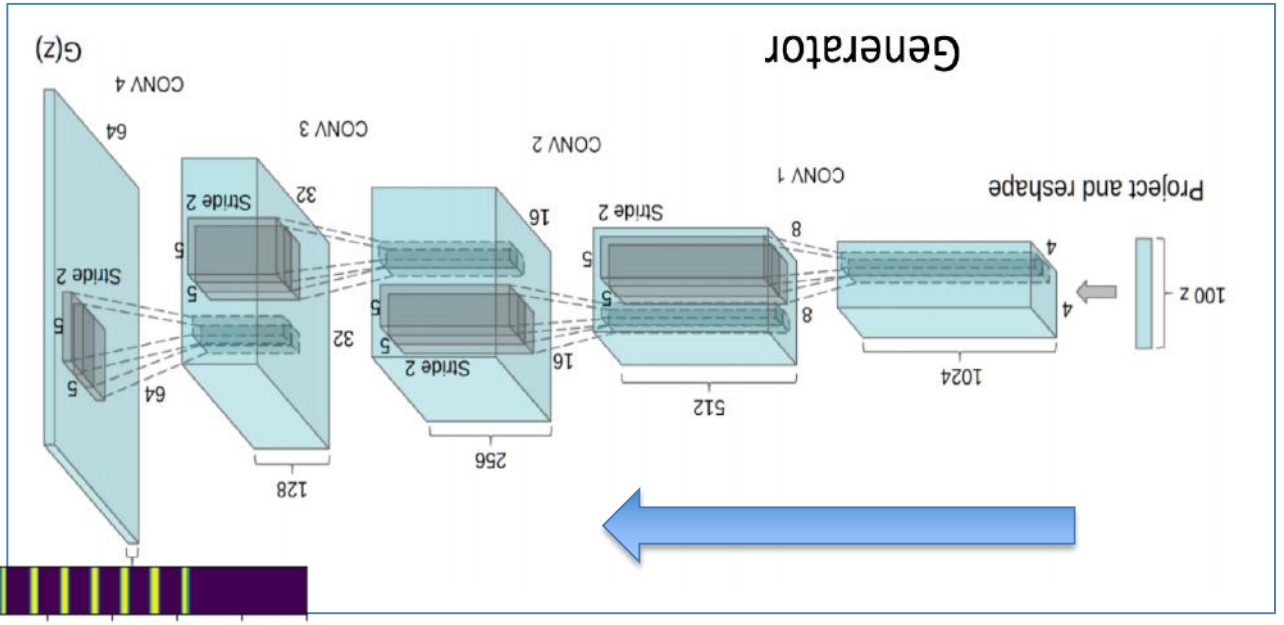


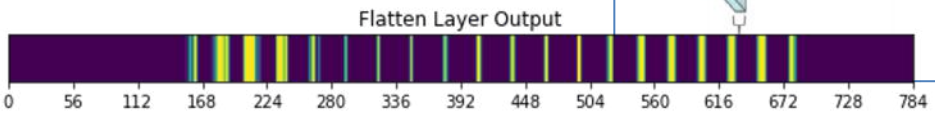






1 image = a vector of 784



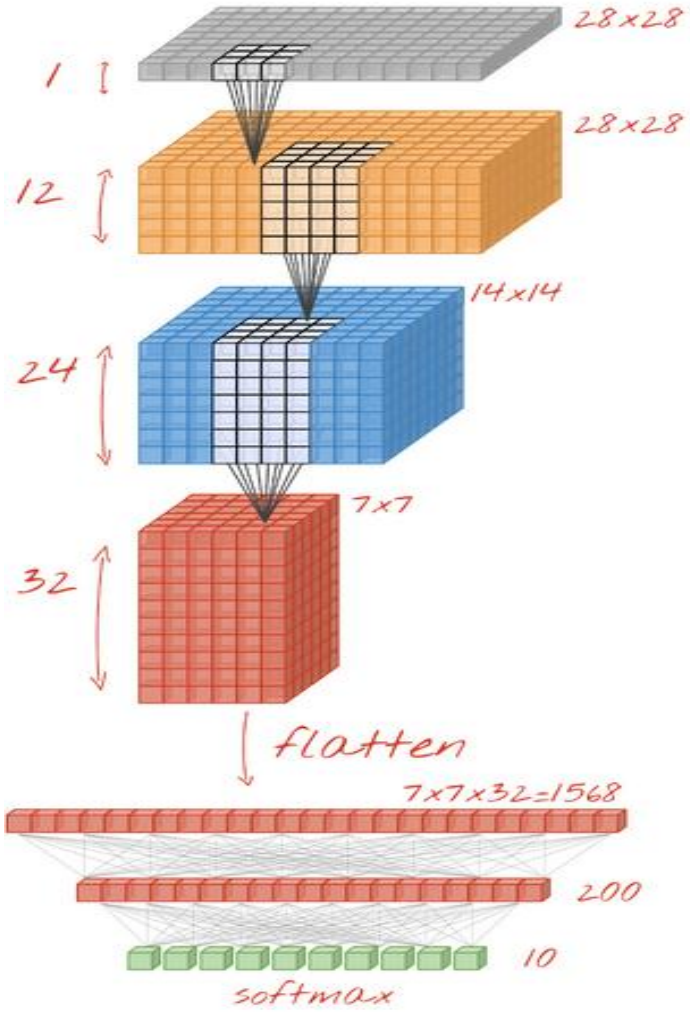


Convolution NN : MNIST 28x28 Pixels

[FW, FH, C, H] = [filter size ? X ?, input channel (Depth), output channel (no. of filter)]

```
tf.keras.layers.Reshape(input_shape=(28*28,), target_shape=(28, 28, 1))
```

```
tf.keras.layers.Conv2D(kernel_size=3, filters=12, padding='same', activation='relu')
```



Convolutional 3x3 filters=12

$W_1[3, 3, 1, 12]$

Convolutional 6x6 filters=24

$W_2[6, 6, 12, 24]$ stride 2

Convolutional 6x6 filters=32

$W_2[6, 6, 24, 32]$ stride 2

https://codelabs.developers.google.com/codelabs/cloud-tensorflow-mnist/#0

Dense layer

$W_4[1568, 200]$

Softmax dense layer

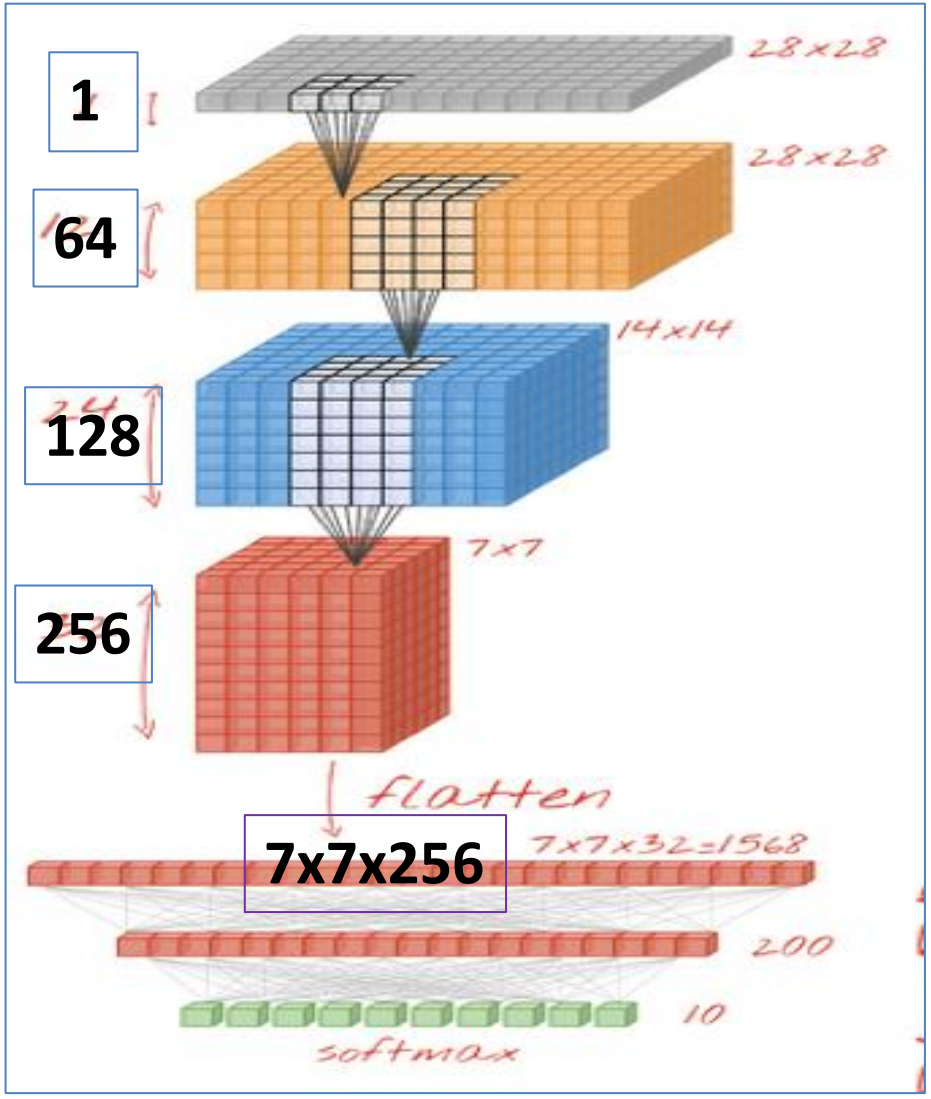
$W_5[200, 10]$

Deep Convolutional Generative Adversarial Network - DCGAN

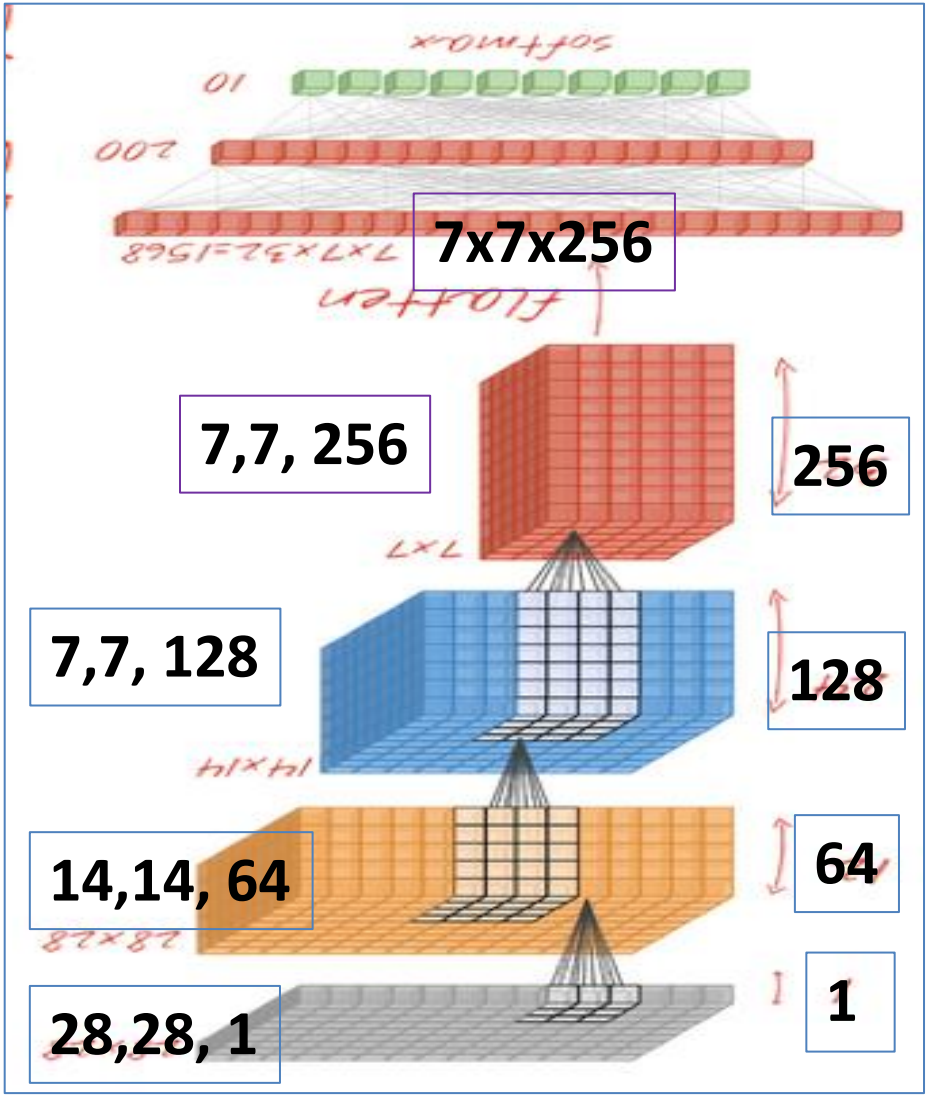
https://www.tensorflow.org/tutorials/generative/dcgan

```
def make_generator_model():
    model = tf.keras.Sequential()
    model.add(layers.Dense(7*7*256, use_bias=False, input_shape=(100,)))
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU())

    model.add(layers.Reshape((7, 7, 256)))
    assert model.output_shape == (None, 7, 7, 256) # Note: None is the batch size

    model.add(layers.Conv2DTranspose(128, (5, 5), strides=(1, 1), padding='same', use_bias=False))
    assert model.output_shape == (None, 7, 7, 128)
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU())

    model.add(layers.Conv2DTranspose(64, (5, 5), strides=(2, 2), padding='same', use_bias=False))
    assert model.output_shape == (None, 14, 14, 64)
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU())

    model.add(layers.Conv2DTranspose(1, (5, 5), strides=(2, 2), padding='same', use_bias=False, activation='tanh'))
    assert model.output_shape == (None, 28, 28, 1)

    return model
```

Generator



convolution



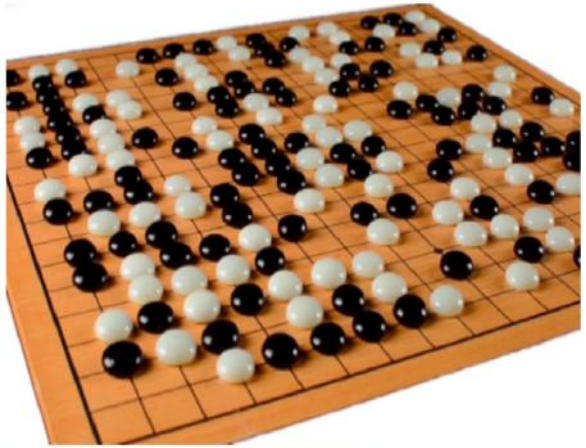
Inverse convolution

[tf.keras.layers.Conv2DTranspose]

Content of Webinar 9

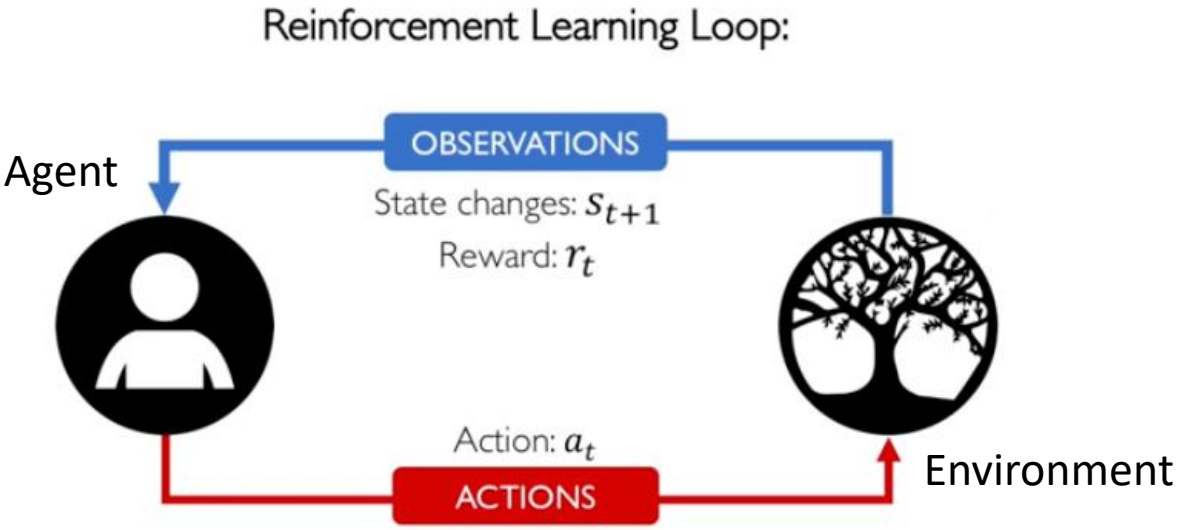
- ✓ Generative Adversarial Networks (GANs)
- ✓ Reinforcement Learning
- ✓ Hyperparameter Tuning
- ✓ openDIEL
- ✓ Parallel Computing, Data and Model Parallelism

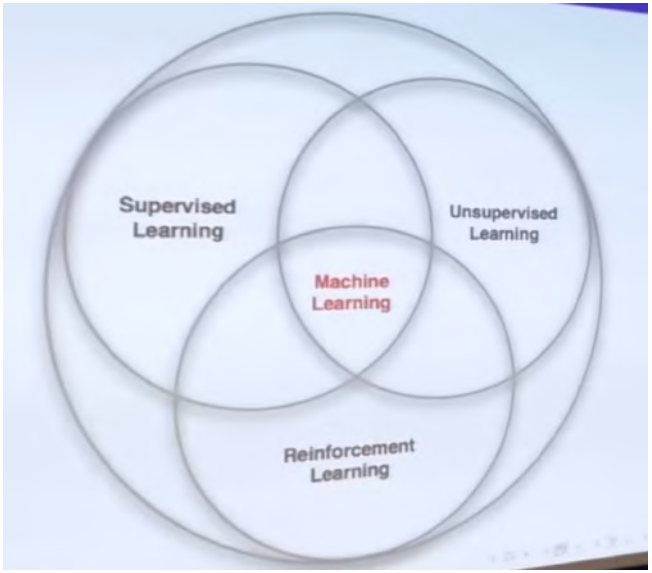
Reinforcement Learning



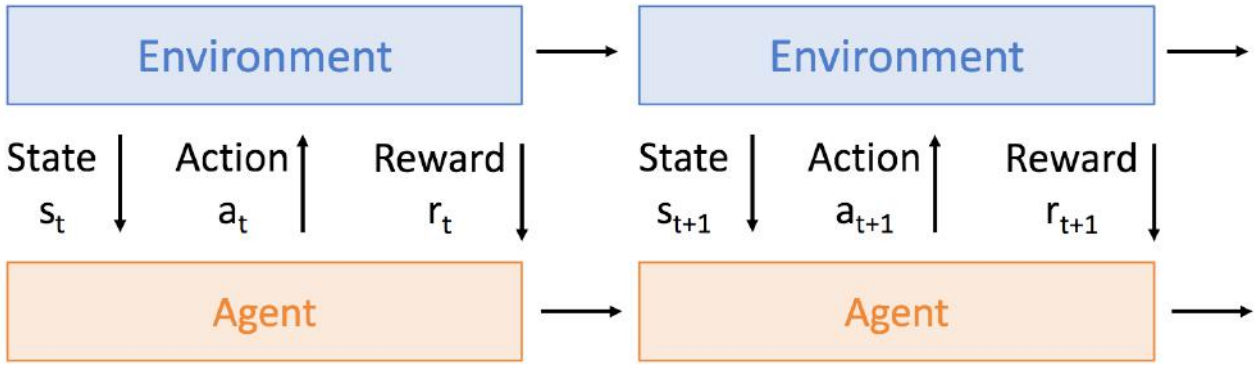
Reinforcement Learning (RL)

- ✓ Reinforcement learning (RL) is a general framework where agents learn to perform actions in an environment so as to maximize a reward. The two main components are the environment, which represents the problem to be solved, and the agent, which represents the learning algorithm.
- ✓ The agent and environment continuously interact with each other. At each time step, the agent takes an action on the environment based on its *policy*, where is the current observation from the environment, and receives a reward and the next observation from the environment. The goal is to improve the policy so as to maximize the sum of rewards (return).
- ✓ RL, known as a semi-supervised learning model in machine learning, is a technique to allow an agent to take actions and interact with an environment so as to maximize the total rewards.
- ✓ The study of RL is to construct a mathematical framework to solve the problems. For example, to find a good policy we could use valued-based methods like Q-learning to measure how good an action is in a particular state or policy-based methods to directly find out what actions to take under different states without knowing how good the actions are.





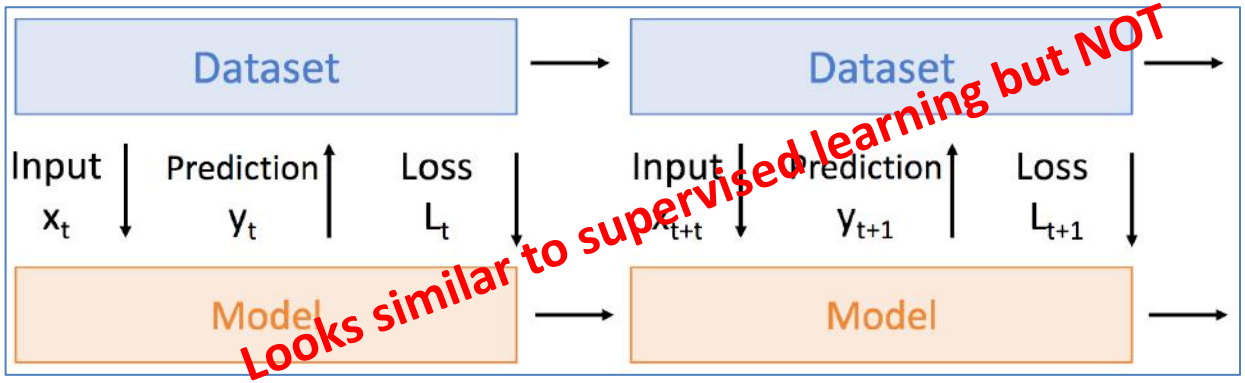
Reinforcement Learning



Many applications
Engineering, CS
Mathematics,
Economics, etc.
Expert system
Control system
Operations research

- ✓ Stochasticity: Rewards and state transitions may be random
- ✓ Nondifferentiable: Can't backprop through world
- ✓ no supervisor, no labels, trial and error based on reward signals
- ✓ Feedback is delayed (in time), not instantaneous, sequential actions,
- ✓ Dynamics nature, Agent takes action based on the signal it receives

- ✓ Objective: Complete the game with the highest score
- ✓ State: Raw pixel inputs of the game screen
- ✓ Action: Game controls e.g. Left, Right, Up, Down
- ✓ Reward: Score increase/decrease at each time step

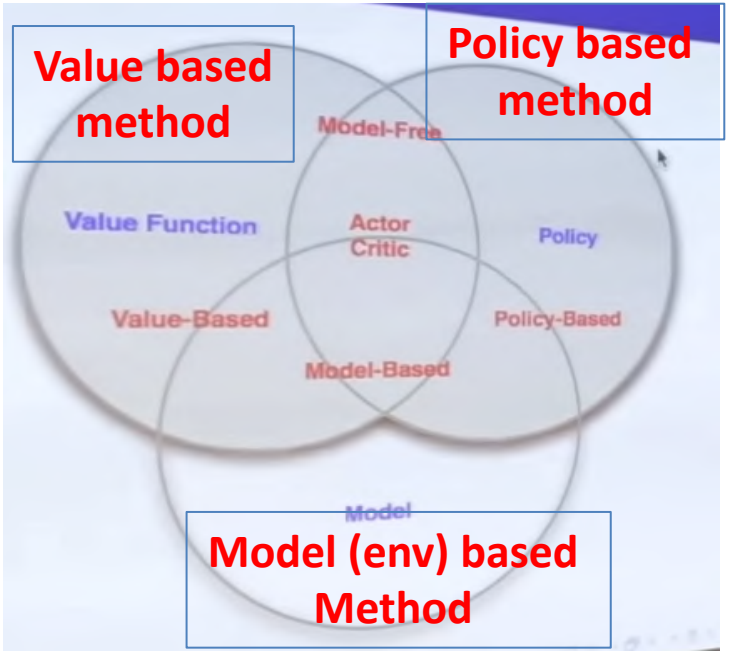


Supervised learning, epoch training, RNN

Math of RL : Markov Decision Process (MDP)

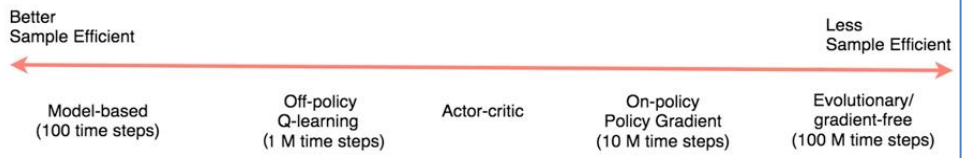
Value function is a prediction of future reward, used to evaluate goodness/badness of states and to select between actions

- ✓ A policy is the agent's behavior
- ✓ It is a map from state to action
- ✓ Deterministic policy or Stochastic policy



- ✓ A model predict what the environment will do next

3 Types of Reinforcement Learning



Model-based

- Learn the model of the world, then plan using the model
- Update model often
- Re-plan often

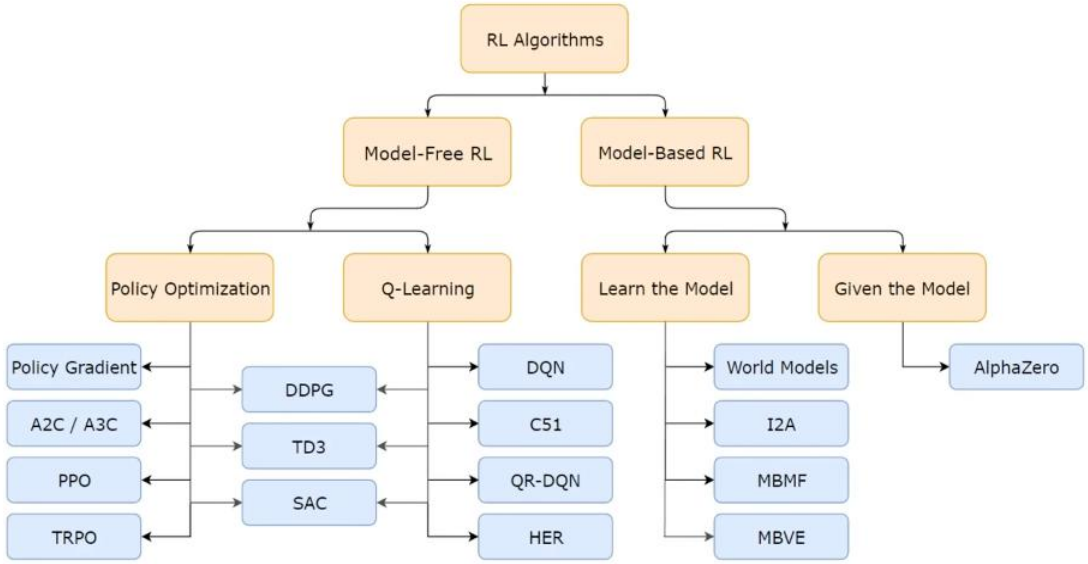
Value-based

- Learn the state or state-action value
- Act by choosing best action in state
- Exploration is a necessary add-on

Policy-based

- Learn the stochastic policy function that maps state to action
- Act by sampling policy
- Exploration is baked in

Taxonomy of RL Methods



Math of RL :Q-Function

Deep Reinforcement Learning Algorithms

Value Learning

Find $Q(s,a)$

$a = \underset{a}{\operatorname{argmax}}\, Q(s,a)$

Policy Learning

Find $\pi(s)$

Sample $a \sim \pi(s)$

Defining the Q-function

$$R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots$$

Total reward, R_t, is the discounted sum of all rewards obtained from time t

$$Q(s,a) = \mathbb{E}[R_t]$$

The Q-function captures the **expected total future reward** an agent in state, s, can receive by executing a certain action, a

The Q-value function at state s and action a, is the expected cumulative reward from taking action a in state s and then following the policy:

$$Q^{\pi}(s,a) = \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t | s_0 = s, a_0 = a, \pi\right]$$

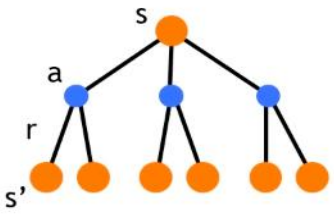
Bellman equation update

$$Q_{i+1}(s,a) = \mathbb{E}\left[r + \gamma \max_{a'} Q_i(s',a') | s,a\right]$$

Qi will converge to Q*

Use DNN to estimate Q(s,a)

Q-Learning



- State-action value function: $Q^{\pi}(s,a)$
  - Expected return when starting in s, performing a, and following π

- Q-Learning: Use **any policy** to estimate Q that maximizes future reward:
  - Q directly approximates Q* (Bellman optimality equation)
  - Independent of the policy being followed
  - Only requirement: keep updating each (s,a) pair

Learning Rate — Discount Factor

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha\left(R_{t+1} + \gamma \max_a Q_t(s_{t+1}, a) - Q_t(s_t, a_t)\right)$$

New State — Old State — Reward

Q-Learning

Q-Learning: Value Iteration

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha \left(R_{t+1} + \gamma \max_a Q_t(s_{t+1}, a) - Q_t(s_t, a_t) \right)$$

Learning Rate, Discount Factor, New State, Old State, Reward

	A1	A2	A3	A4
S1	+1	+2	-1	0
S2	+2	0	+1	-2
S3	-1	+1	0	-2
S4	-2	0	+1	+1

```
initialize Q[num_states,num_actions] arbitrarily
observe initial state s
repeat
      select and carry out an action a
      observe reward r and new state s'
      Q[s,a] = Q[s,a] + α(r + γ max_a' Q[s',a'] - Q[s,a])
      s = s'
until terminated
```

Q-Learning: Representation Matters

- In practice, Value Iteration is impractical
  - Very limited states/actions
  - Cannot generalize to unobserved states

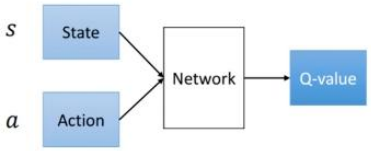
- Think about the **Breakout** game
  - State: screen pixels
    - Image size: 84×84 (resized)
    - Consecutive 4 images
    - Grayscale with 256 gray levels

$256^{84\times84\times4}$ rows in the Q-table!

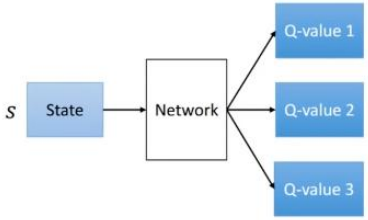
$= 10^{69,970} >> 10^{82}$ atoms in the universe

DQN: Deep Q-Learning

Use a neural network to approximate the Q-function:



$$Q(s, a; \boldsymbol{\theta}) \approx Q^*(s, a)$$



DQN and Double DQN

- Loss function (squared error):

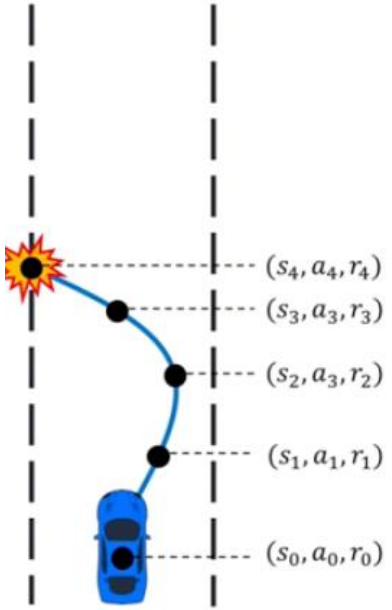
$$L = \mathbb{E}[(\underbrace{r + \gamma max_{a'} Q(s', a')}_{\text{target}} - \underbrace{Q(s, a)}_{\text{prediction}})^2]$$

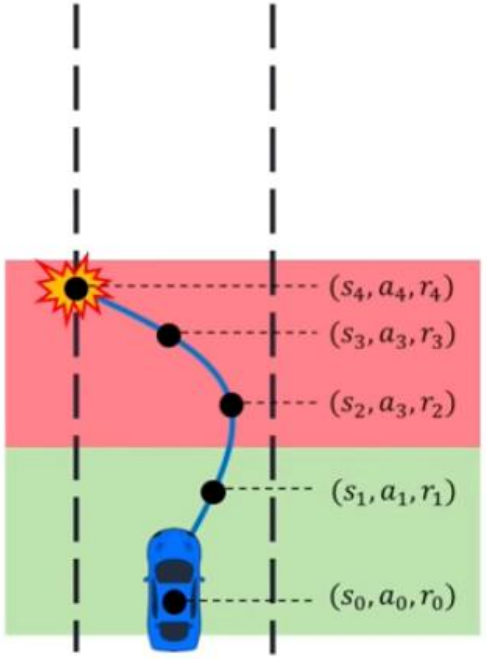
- DQN: same network for both Q
- Double DQN: separate network for each Q
  - Helps reduce bias introduced by the inaccuracies of Q network at the beginning of training

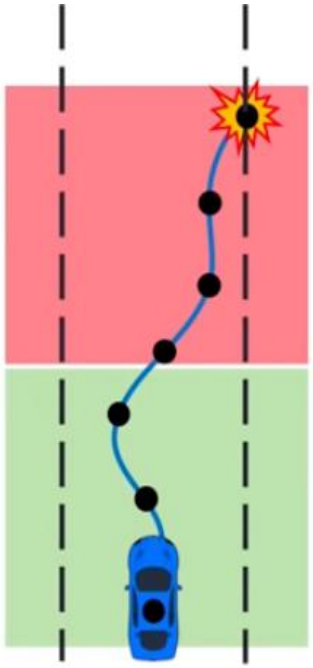
RL Example

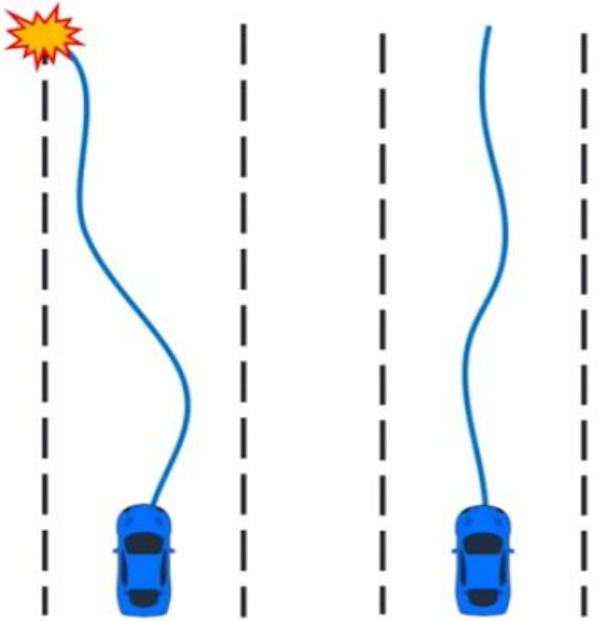
Training Algorithm

1. Initialize the agent
2. Run a policy until termination
3. Record all states, actions, rewards
4. Decrease probability of actions that resulted in low reward
5. Increase probability of actions that resulted in high reward

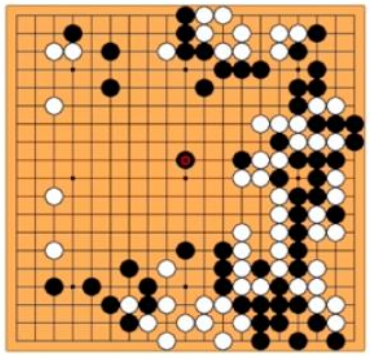






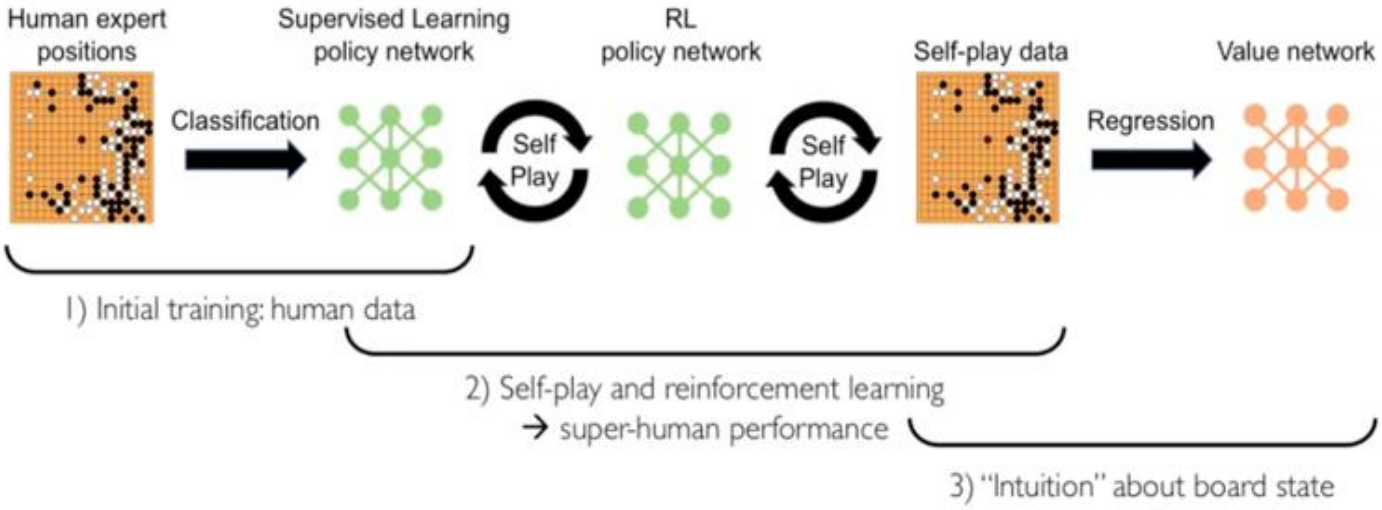


Go Game



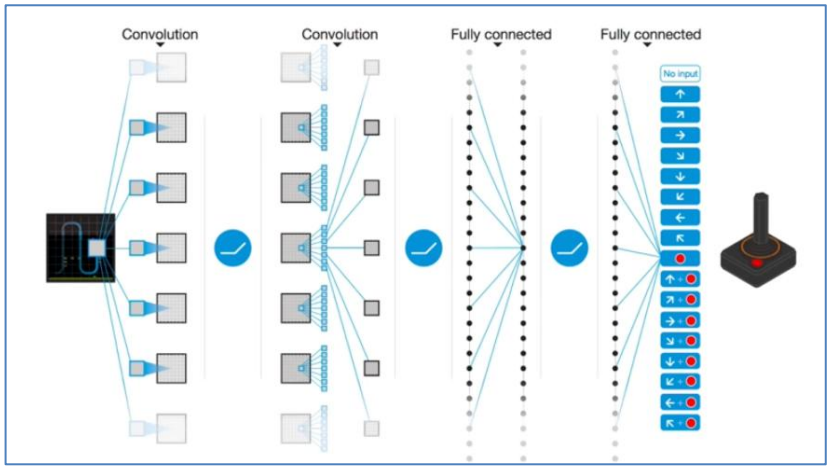
Board Size n x n	Positions 3^{n^2}	% Legal	Legal Positions
1×1	3	33.33%	1
2×2	81	70.37%	57
3×3	19,683	64.40%	12,675
4×4	43,046,721	56.49%	24,318,165
5×5	847,288,609,443	48.90%	414,295,148,741
9×9	4.434264882×10^{38}	23.44%	$1.039191487\,91\times10^{38}$
13×13	4.300233593×10^{80}	8.66%	$3.72497923077\times10^{79}$
19×19	$1.740896506\times10^{172}$	1.20%	$2.081681993\,82\times10^{170}$

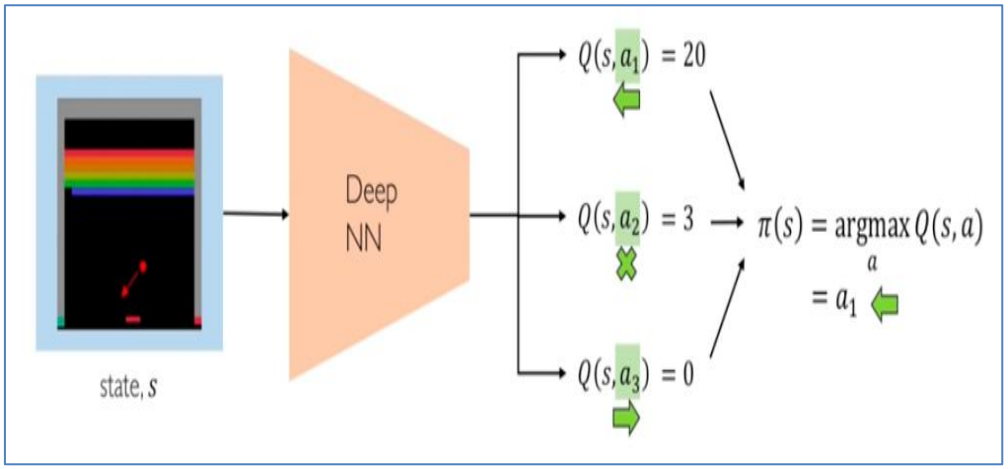
- ✓ Objective: Win the game!
- ✓ State: Position of all pieces
- ✓ Action: Where to put the next piece down
- ✓ Reward: On last turn: 1 if you won, 0 if you lost



- ✓ AlphaGo: (January 2016) -- Used imitation learning+tree search+RL-Beat 18-time world champion Lee Sedol
- ✓ AlphaGoZero (October 2017) -- Simplified version of AlphaGo-No longer using imitation learning-Beat (at the time) #1 ranked KeJie
- ✓ AlphaZero(December2018) -- Generalized to other games: Chess and Shogi

Q-Network





Q-network Architecture

$Q(s, a; \theta)$: neural network with weights θ

A single feedforward pass to compute Q-values for all actions from the current state => efficient!

FC-4 (Q-values)

FC-256

32 4x4 conv, stride 2

16 8x8 conv, stride 4

Last FC layer has 4-d output (if 4 actions), corresponding to $Q(s_t, a_1)$, $Q(s_t, a_2)$, $Q(s_t, a_3)$, $Q(s_t, a_4)$

Number of actions between 4-18 depending on Atari game

Current state s_t: 84x84x4 stack of last 4 frames
(after RGB->grayscale conversion, downsampling, and cropping)

Downsides of Q-learning

Complexity:

- Can model scenarios where the action space is discrete and small
- Cannot handle continuous action spaces

IMPORTANT: Imagine you want to predict steering wheel angle of a car!

Flexibility:

- Cannot learn stochastic policies since policy is deterministically computed from the Q function

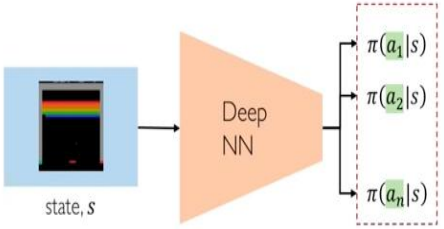
To overcome, consider a new class of RL training algorithms: Policy gradient methods

Policy Gradient (PG): Key Idea

DQN (before): Approximating Q and inferring the optimal policy,

Policy Gradient: Directly optimize the policy!

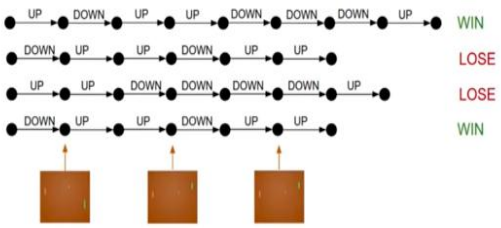
$$\pi^* = \arg\max_{\pi} \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t \mid \pi\right]$$



$$\sum_{a_i \in A} \pi(a_i|s) = 1$$

$$\pi(a|s) = P(action|state)$$

Policy Gradient (PG): Training



1. Run a policy for a while
2. Increase probability of actions that lead to high rewards
3. Decrease probability of actions that lead to low/no rewards

```
function REINFORCE
 Initialize θ
 for episode ~ π_θ
  {s_i, a_i, r_i}_{i=1}^{T-1} ← episode
  for t = 1 to T-1
    ∇ ← ∇_θ log π_θ(a_t|s_t) R_t
    θ ← θ + α∇
 return θ
```

log-likelihood of action

$$\nabla_\theta \log \pi_\theta(a_t|s_t)\, R_t$$

reward

AlphaGo Beats Top Human Player at Go (2016)

1) Initial training: human data

2) Self-play and reinforcement learning → super-human performance

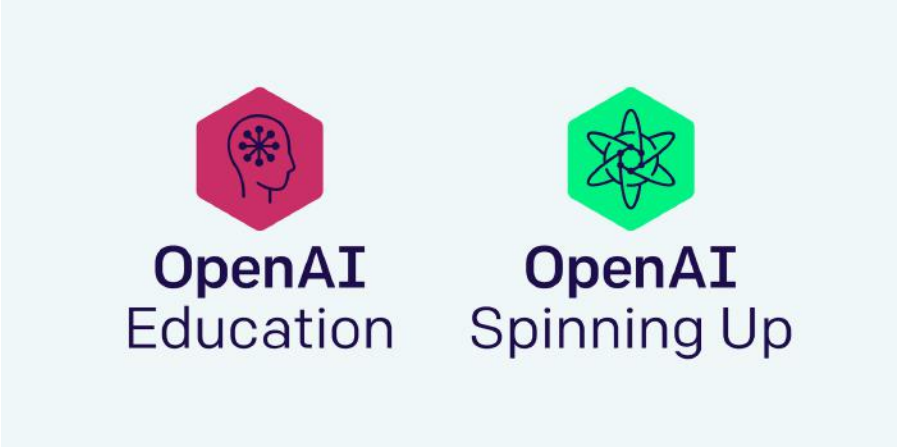
3) "Intuition" about board state

RL – Examples, openAI

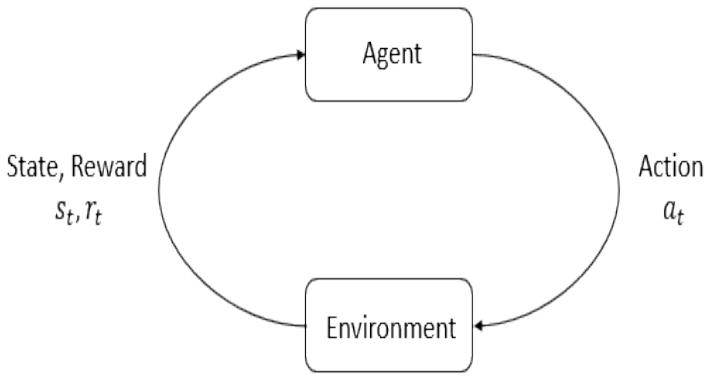
https://github.com/jeffheaton/t81_558_deep_learning -- module 12 Reinforcement Learning
https://gym.openai.com/
https://spinningup.openai.com/en/latest/

Gym is a toolkit for developing and comparing reinforcement learning algorithms. It supports teaching agents everything from walking to playing games like Pong or Pinball.

Spinning Up in Deep RL : An educational resource designed to let anyone learn to become a skilled practitioner in deep reinforcement learning. Spinning Up consists of examples of RL code, educational exercises, documentation, and tutorials.



Reinforcement Learning - Overview



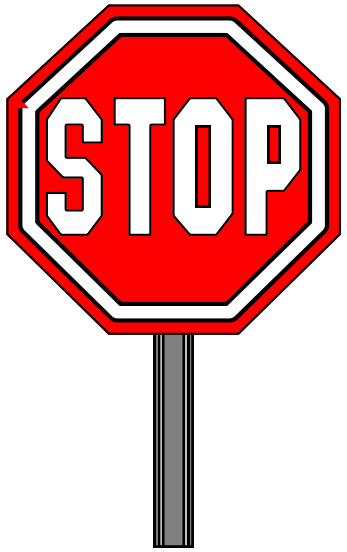
The main characters of RL are the **agent** and the **environment**. The environment is the world that the agent lives in and interacts with. At every step of interaction, the agent sees a (possibly partial) observation of the state of the world, and then decides on an action to take. The environment changes when the agent acts on it, but may also change on its own.
The agent also perceives a **reward** signal from the environment, a number that tells it how good or bad the current world state is. The goal of the agent is to maximize its cumulative reward, called **return**. Reinforcement learning methods are ways that the agent can learn behaviors to achieve its goal.

A **state S** is a complete description of the state of the world. There is no information about the world which is hidden from the state. An **observation** is a partial description of a state, which may omit information.
In deep RL, we almost always represent states and observations by a real-valued vector, matrix, or higher-order tensor. For instance, a visual observation could be represented by the RGB matrix of its pixel values; the state of a robot might be represented by its joint angles and velocities.
When the agent is able to observe the complete state of the environment, we say that the environment is **fully observed. When the agent can only see a partial observation, we say that the environment is partially observed.**

Different environments allow different kinds of actions. The set of all valid actions in a given environment is often called the **action space**. Some environments, like Atari and Go, have **discrete action spaces**, where only a finite number of moves are available to the agent. Other environments, like where the agent controls a robot in a physical world, have **continuous action spaces**. In continuous spaces, actions are real-valued vectors.
This distinction has some quite-profound consequences for methods in deep RL. Some families of algorithms can only be directly applied in one case, and would have to be substantially reworked for the other.

https://spinningup.openai.com/en/latest/spinningup/rl_intro.html

The End



- The End!