Neural Network

UW CSE 473

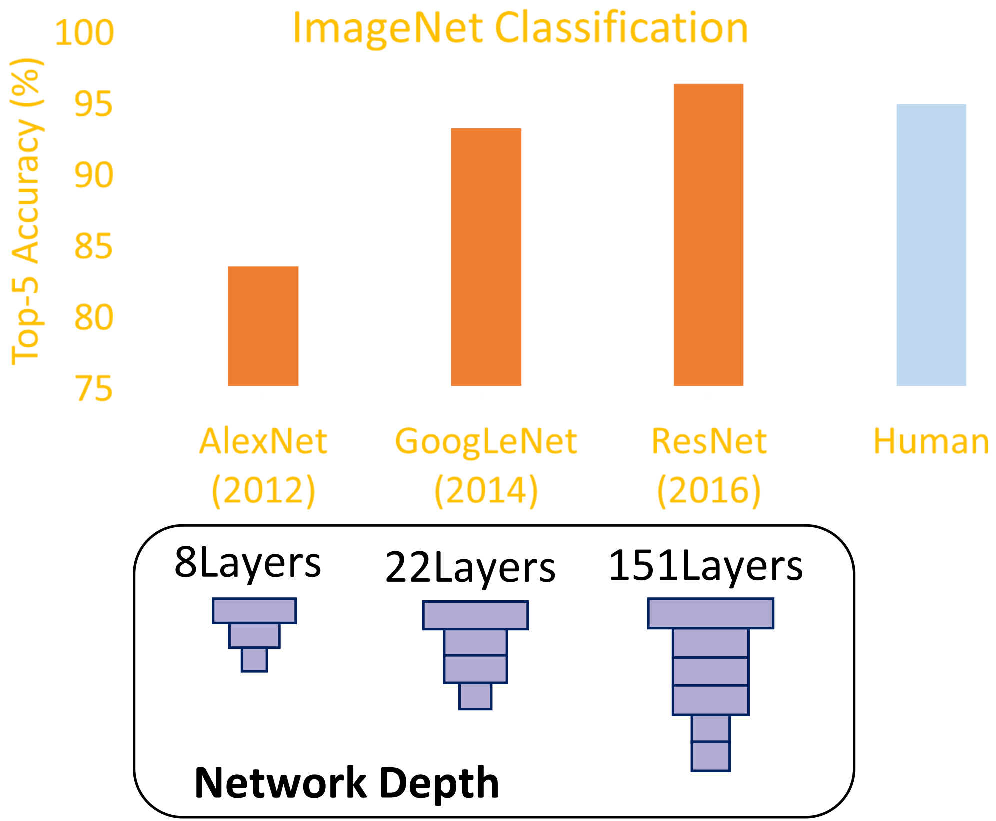
Feb/24/2023

Kechun Liu

Summary

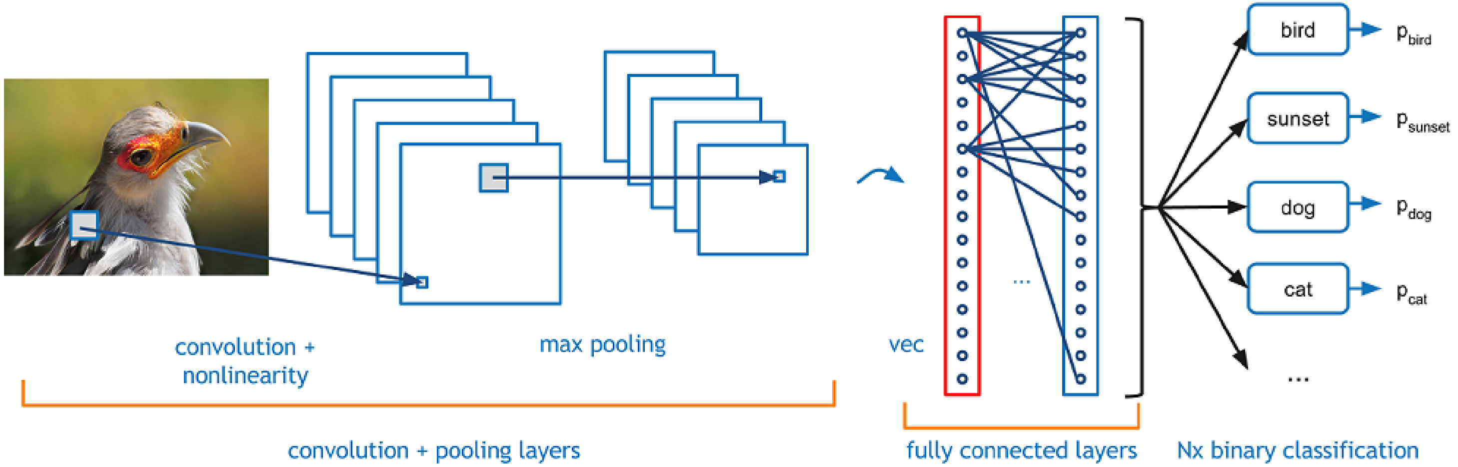
- CNN
  - Image Classification
  - Semantic Segmentation
  - Detection
- RNN
- Transformer
- RL
- PyTorch

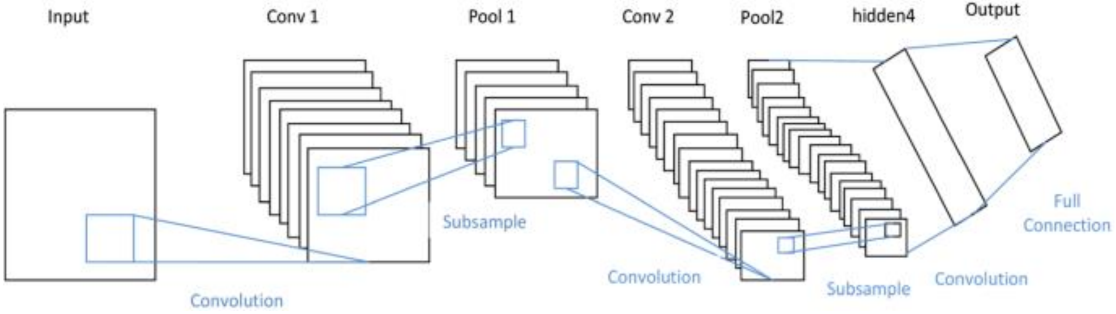
Introduction



IMAGENET

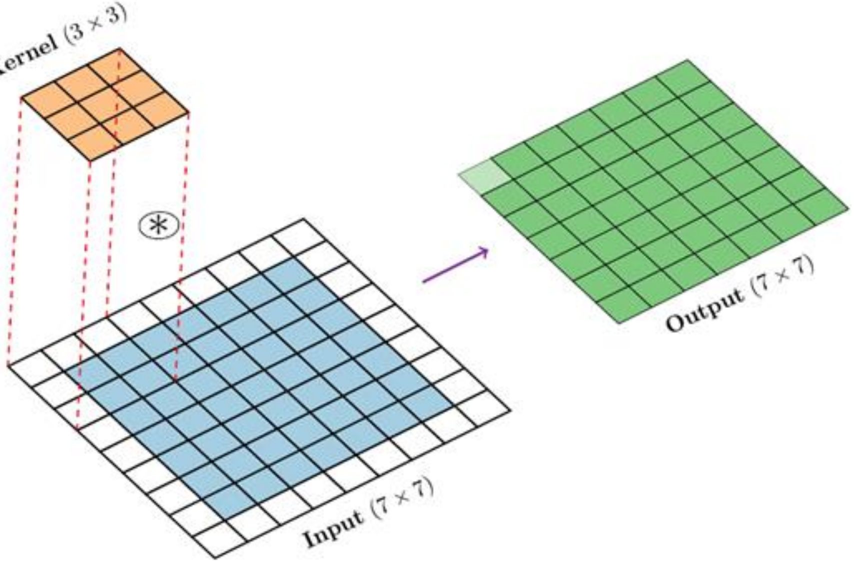
CNN

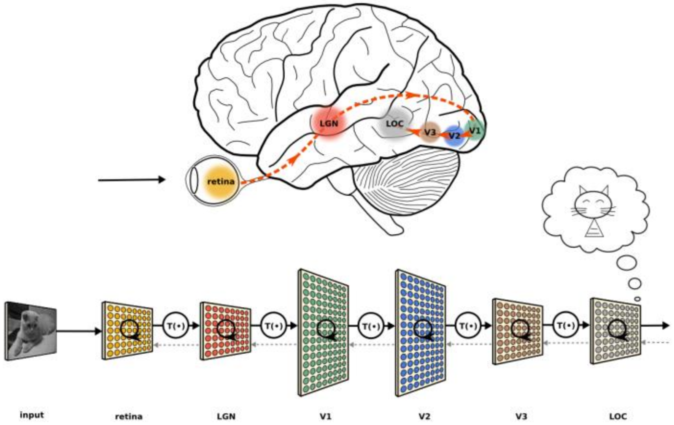

convolution + nonlinearity
max pooling
vec
...
bird
sunset
dog
cat
...
p_{bird}
p_{sunset}
p_{dog}
p_{cat}
convolution + pooling layers
fully connected layers
Nx binary classification


Input
Conv 1
Pool 1
Conv 2
Pool2
hidden4
Output
Convolution
Subsample
Convolution
Subsample
Convolution
Full Connection

Convolution Operation

Nowadays, we learn kernels from the data.





Learning

$$\begin{bmatrix} O_{11} & O_{12} \\ O_{21} & O_{22} \end{bmatrix} = \text{Convolution}\left(\begin{bmatrix} X_{11} & X_{12} & X_{13} \\ X_{21} & X_{22} & X_{23} \\ X_{31} & X_{32} & X_{33} \end{bmatrix}, \begin{bmatrix} F_{11} & F_{12} \\ F_{21} & F_{22} \end{bmatrix} \right)$$

$$O_{11} = F_{11}X_{11} + F_{12}X_{12} + F_{21}X_{21} + F_{22}X_{22}$$

$$O_{12} = F_{11}X_{12} + F_{12}X_{13} + F_{21}X_{22} + F_{22}X_{23}$$

$$O_{21} = F_{11}X_{21} + F_{12}X_{22} + F_{21}X_{31} + F_{22}X_{32}$$

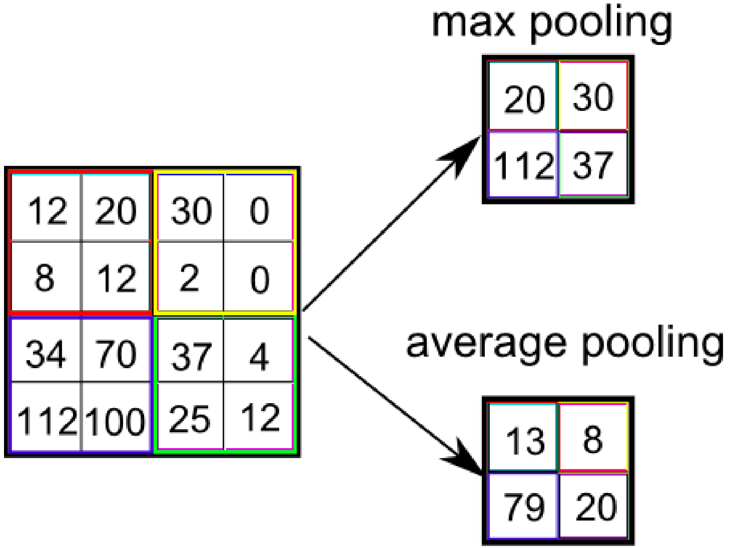
$$O_{22} = F_{11}X_{22} + F_{12}X_{23} + F_{21}X_{32} + F_{22}X_{33}$$

- Details:
- https://www.slideshare.net/EdwinEfranJimnezLepe/example-feedforward-backpropagation
- https://medium.com/@2017csm1006/forward-and-backpropagation-in-convolutional-neural-network-4dfa96d7b37e

Pooling

e.g. kernel size = 2, stride = 2 for both width and height.

The kernel size for pooling can be an even number.



CNN Structures
Image Classification

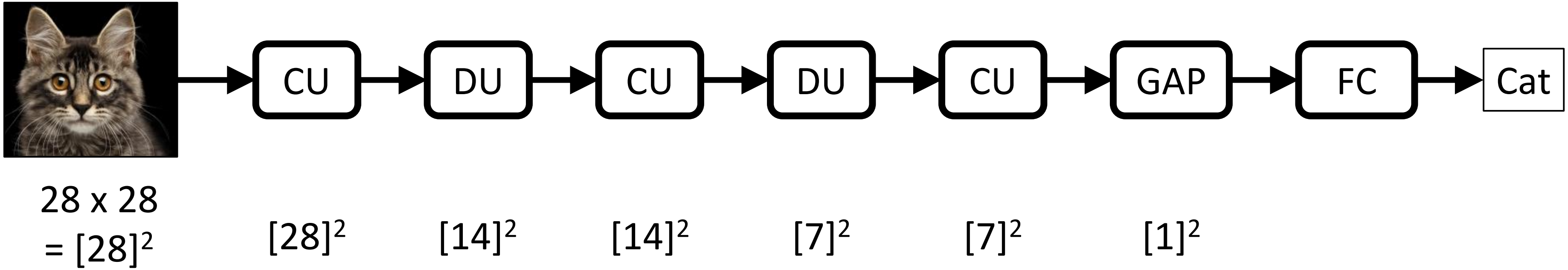
Image Classification

CU — Convolutional Unit

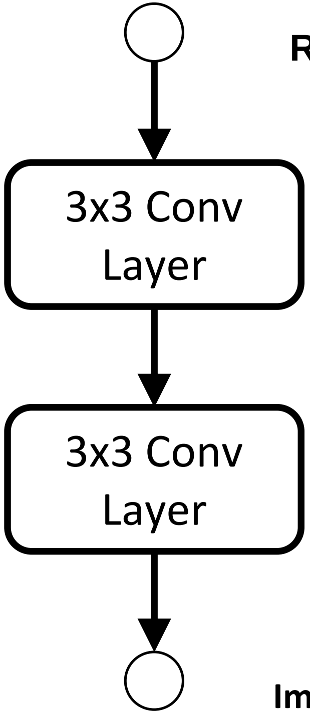
FC — Fully-connected Or Linear Layer

DU — Down-sampling Unit

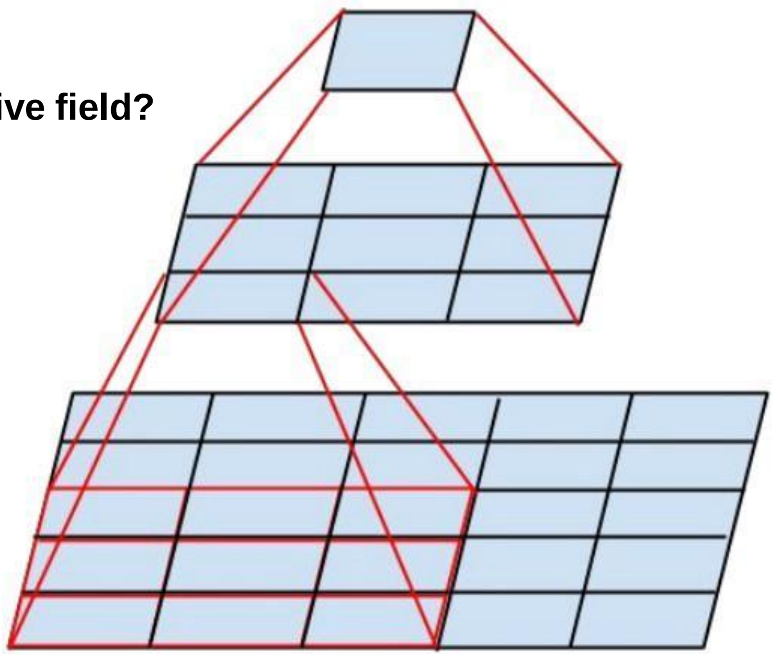
GAP — Global Avg. Pooling



Convolutional Unit (CU) - VGG



Receptive field?



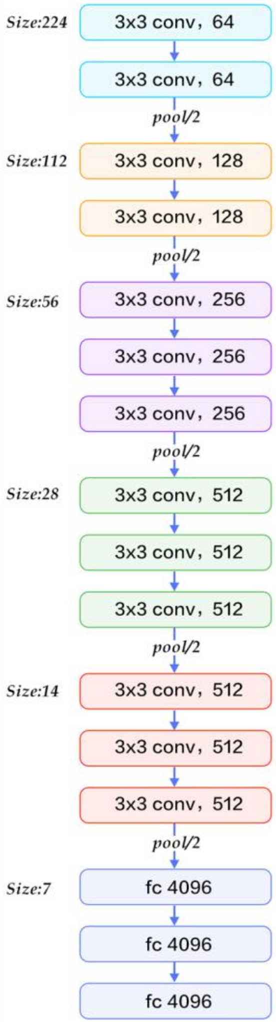
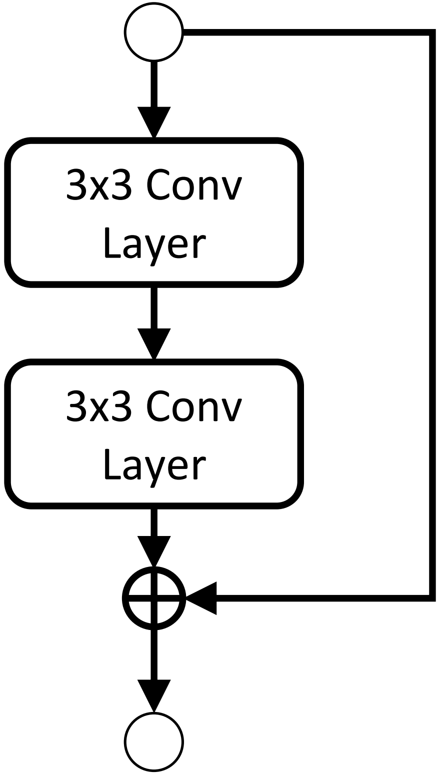


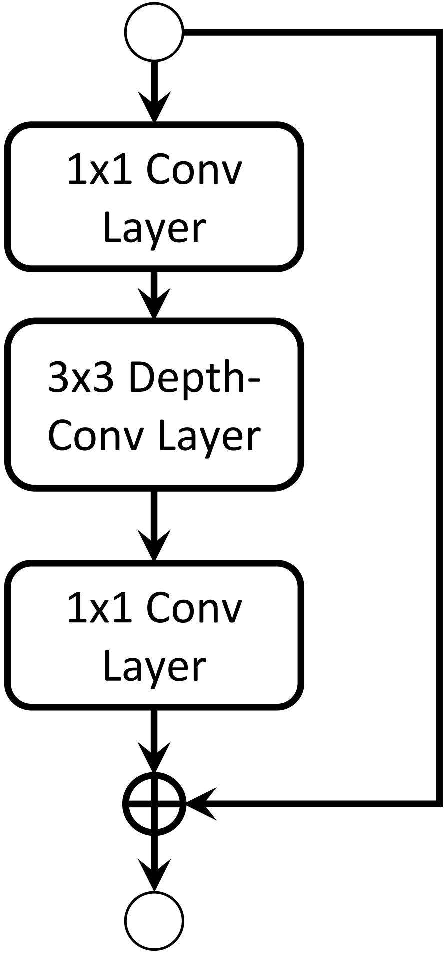
Image Source (Inception): Szegedy, Christian, et al. "Rethinking the inception architecture for computer vision." *CVPR*. 2016.

Basic Block in ResNet



ResNet: He, Kaiming, et al. "Deep residual learning for image recognition." CVPR. 2016.

- Residual Connection
- Element-wise addition of input and output
- Improves gradient flow and accuracy
- In ResNet-18 and ResNet-34
- Still computationally expensive
  - Hard to train very deep networks (> 100 layers)



Bottleneck in ResNet

- Used in ResNet-50, ResNet-101, ResNet-152, etc...
- Computationally Efficient

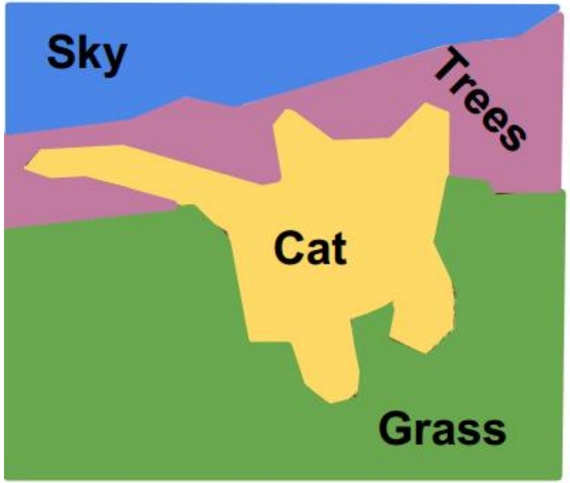
Influence:

- Bottleneck unit with Depth-wise convs
  - MobileNetv2
  - ShuffleNetv2

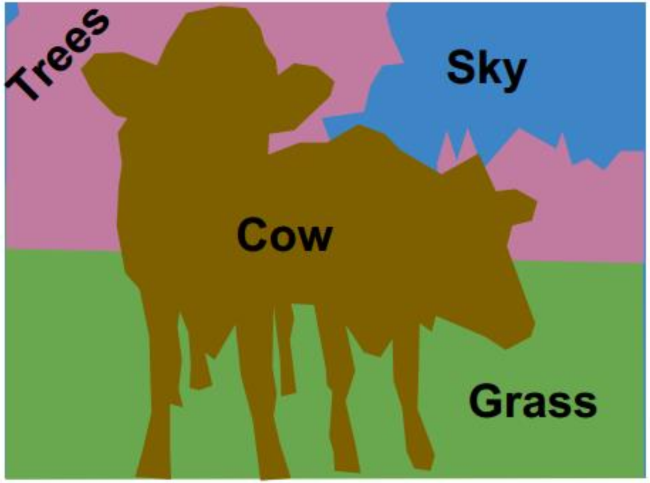
- **MobileNetv2:** Sandler, Mark, et al. "Mobilenetv2: Inverted residuals and linear bottlenecks." CVPR, 2018.
- **ShuffleNetv2:** Ma, Ningning, et al. "Shufflenet v2: Practical guidelines for efficient cnn architecture design." ECCV, 2018.

CNN Structures

Semantic Segmentation

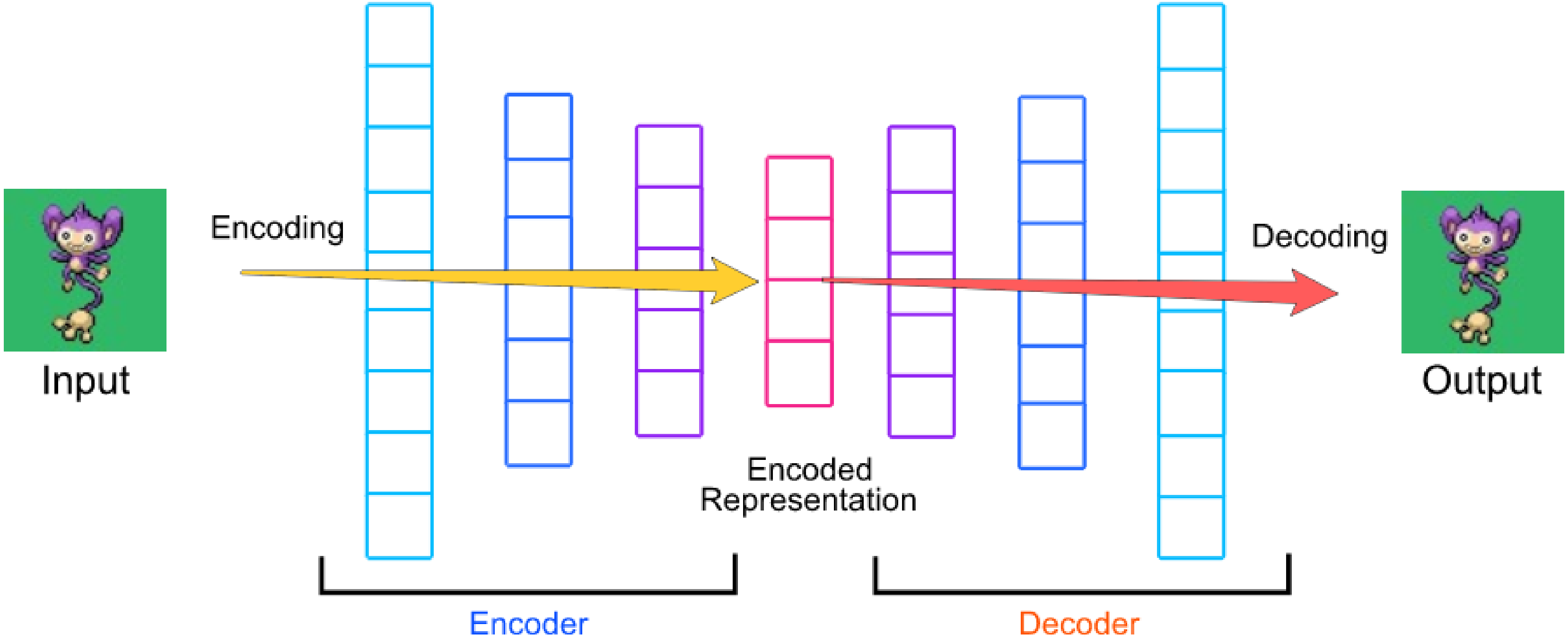


Sky
Trees
Cat
Grass

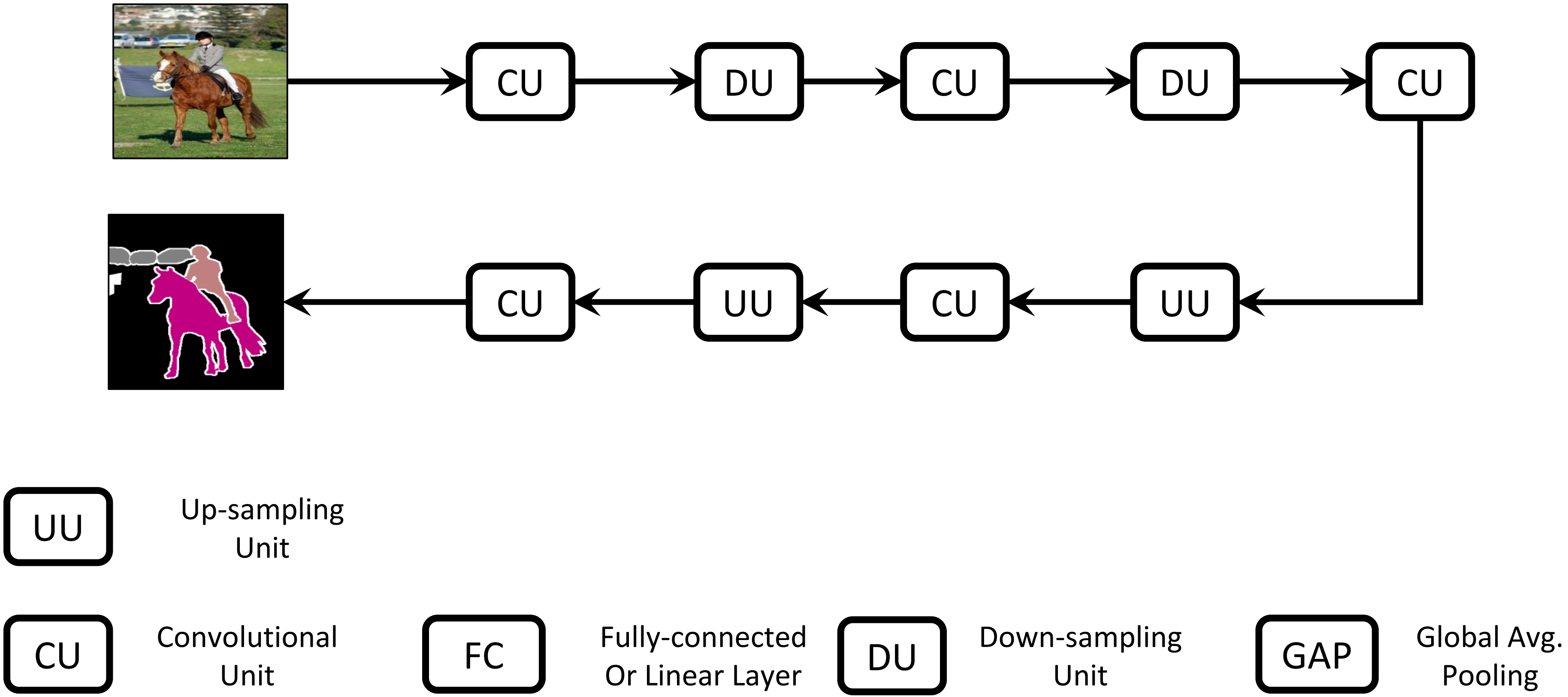


Trees
Sky
Cow
Grass

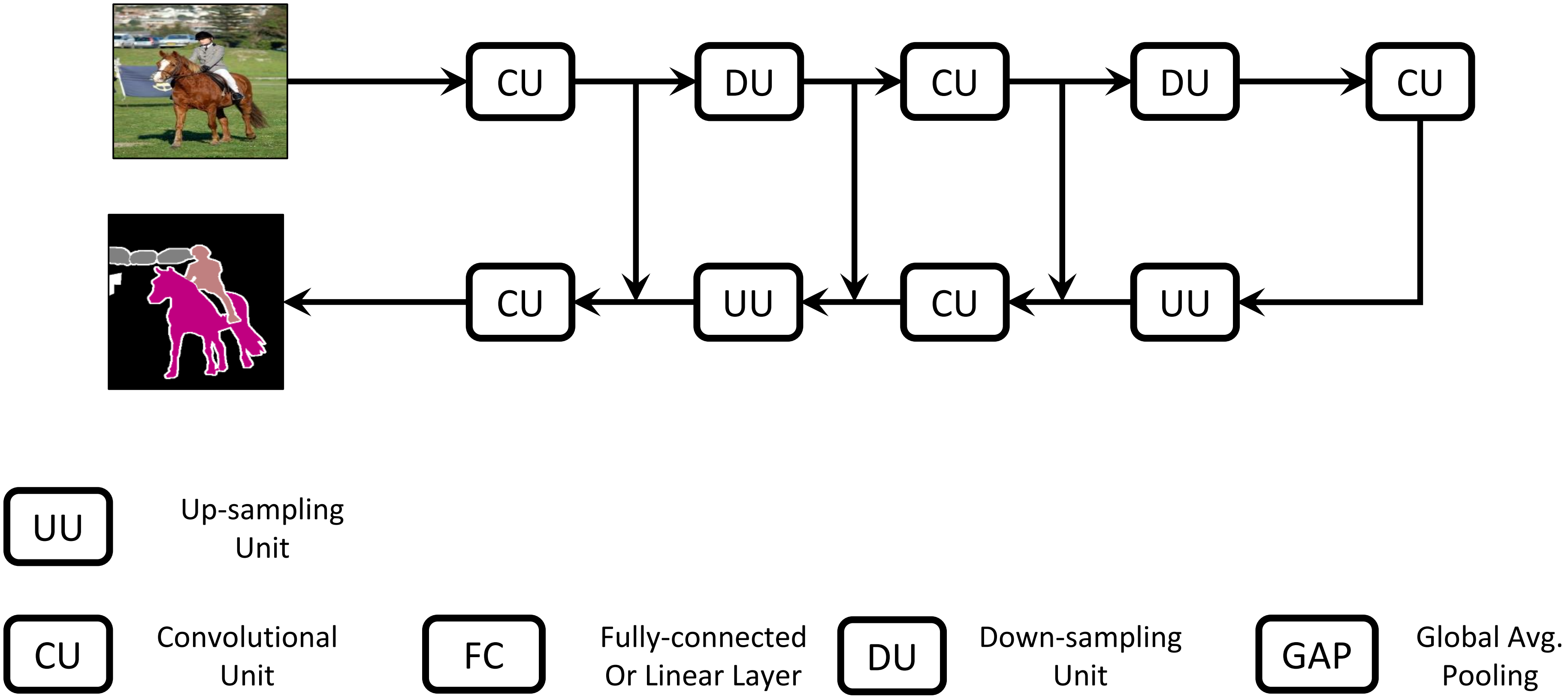
Encoder-Decoder



Encoder-Decoder in Semantic Segmentation

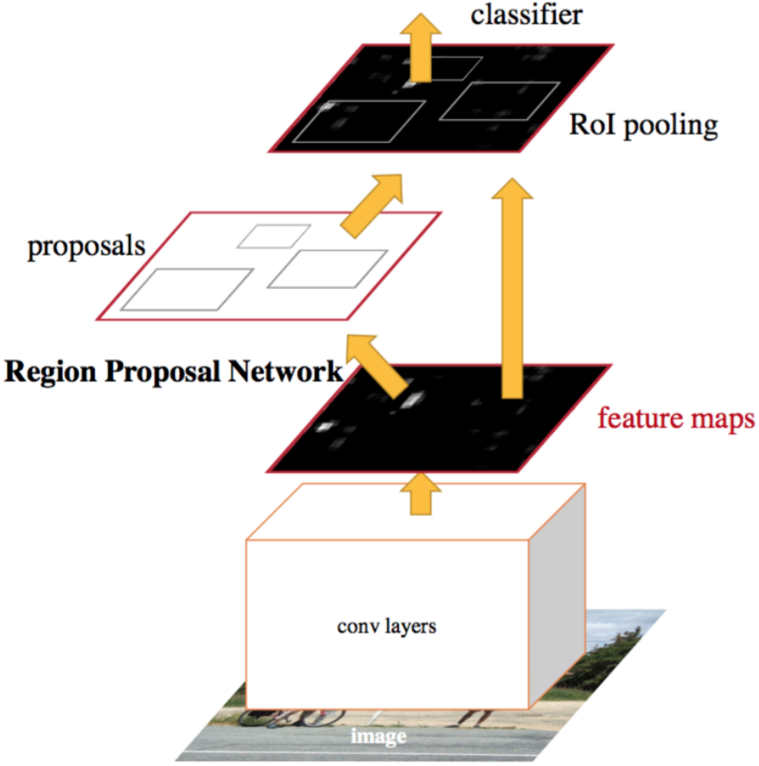


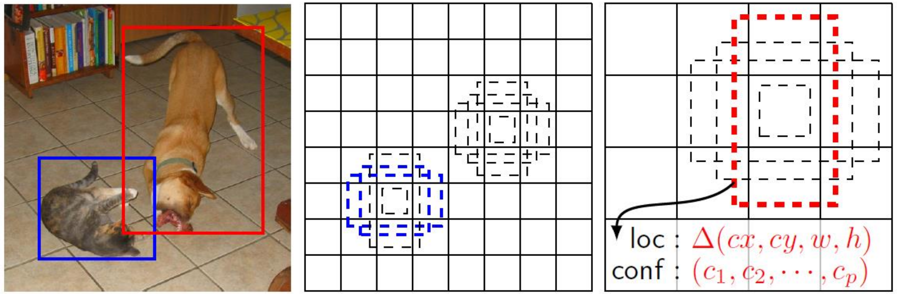
U-Net


CU
DU
CU
DU
CU
CU
UU
CU
UU
UU
Up-sampling Unit
CU
Convolutional Unit
FC
Fully-connected Or Linear Layer
DU
Down-sampling Unit
GAP
Global Avg. Pooling

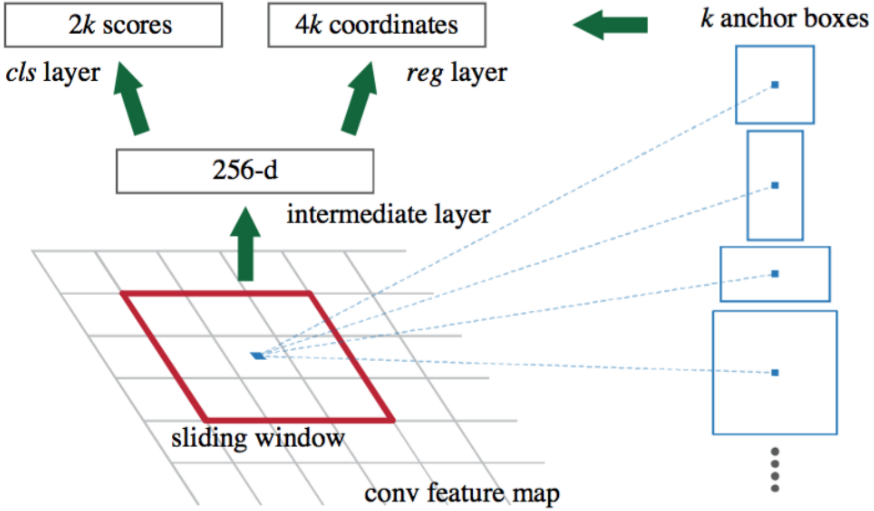
CNN Structures
Detection

Faster RCNN





(a) Image with GT boxes (b) 8×8 feature map (c) 4×4 feature map

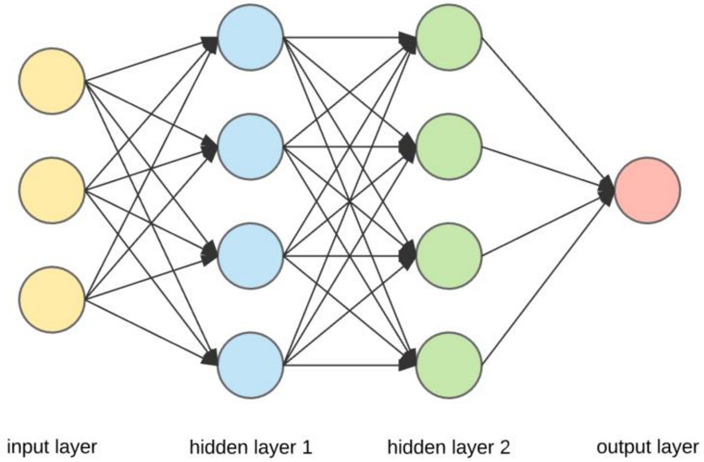


RNN Structures

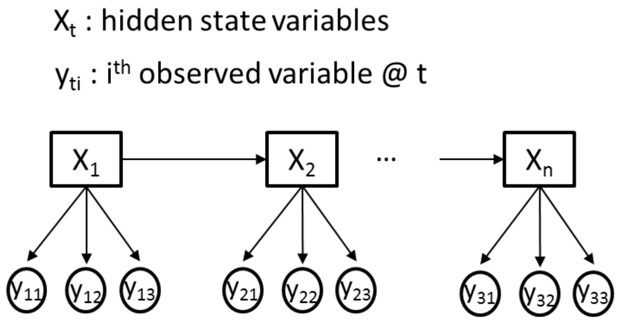
Challenges for time-series signals

- Different signal length
- Online inference for new timepoint

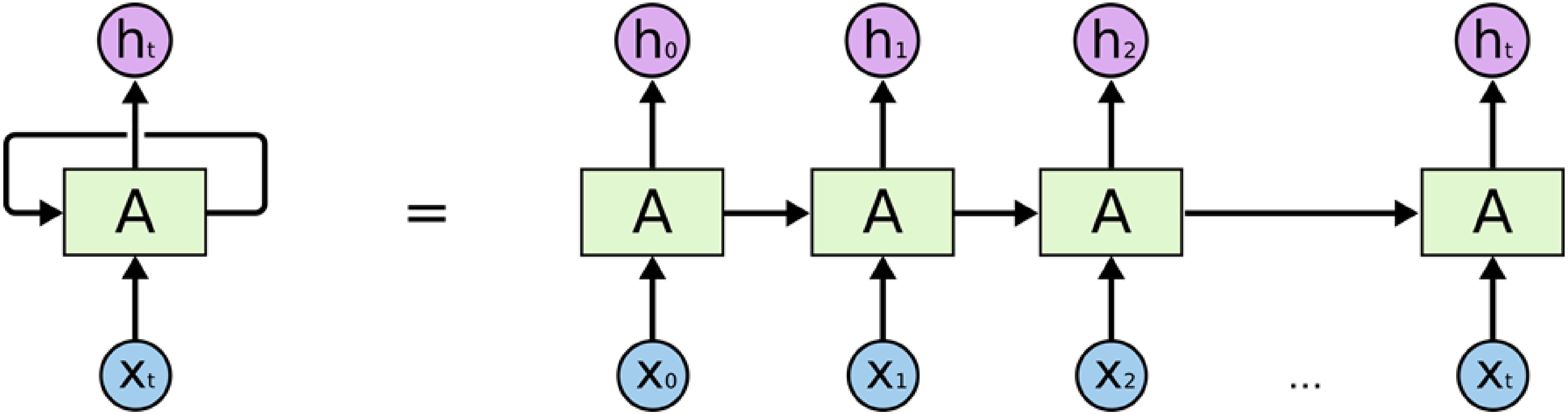
(Vanilla) Neural Network
1940s - 1980s



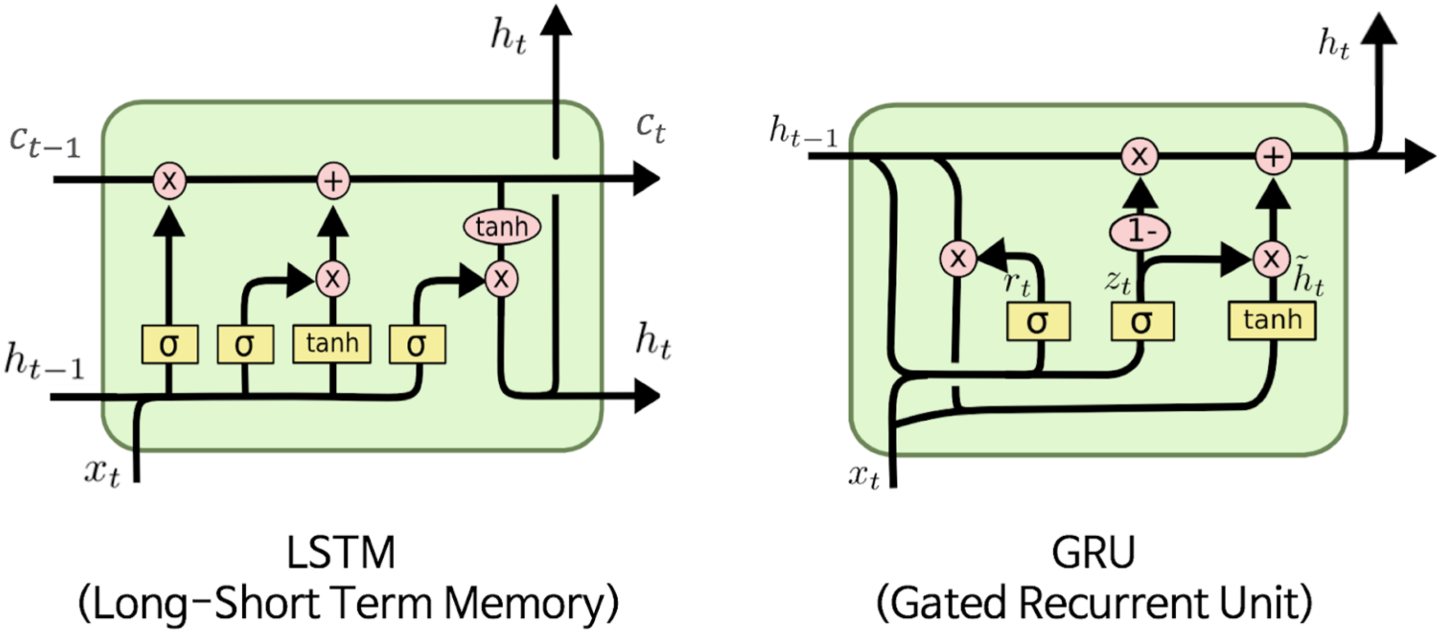
Hidden Markov Model
Andrew Viterbi, 1967
Lawrence Rabiner, 1989



Recurrent Neural Network

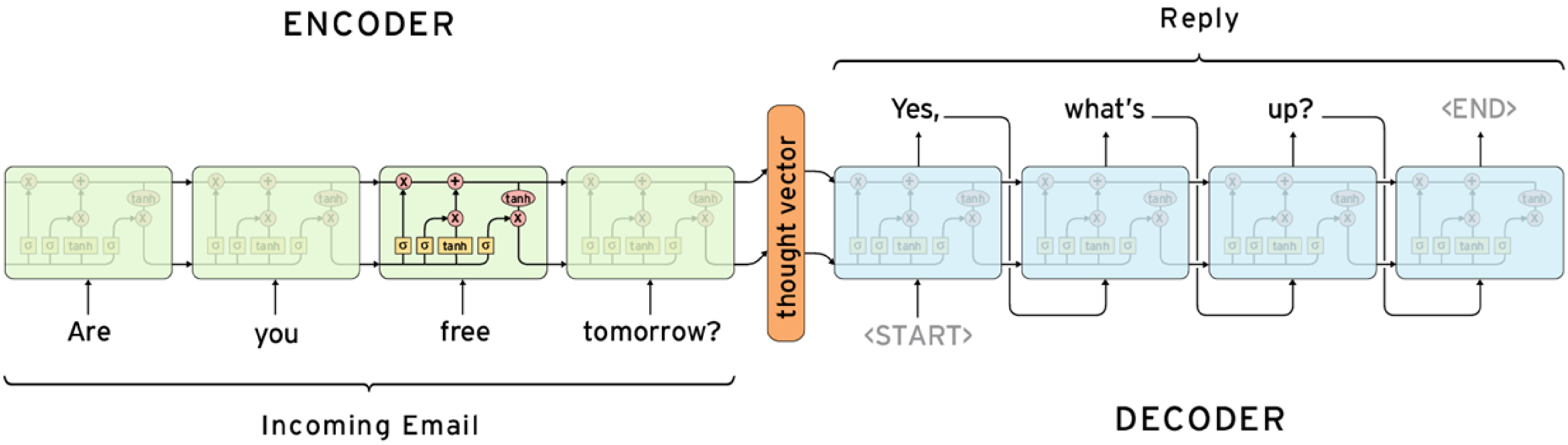


LSTM and GRU: Memory for RNNs



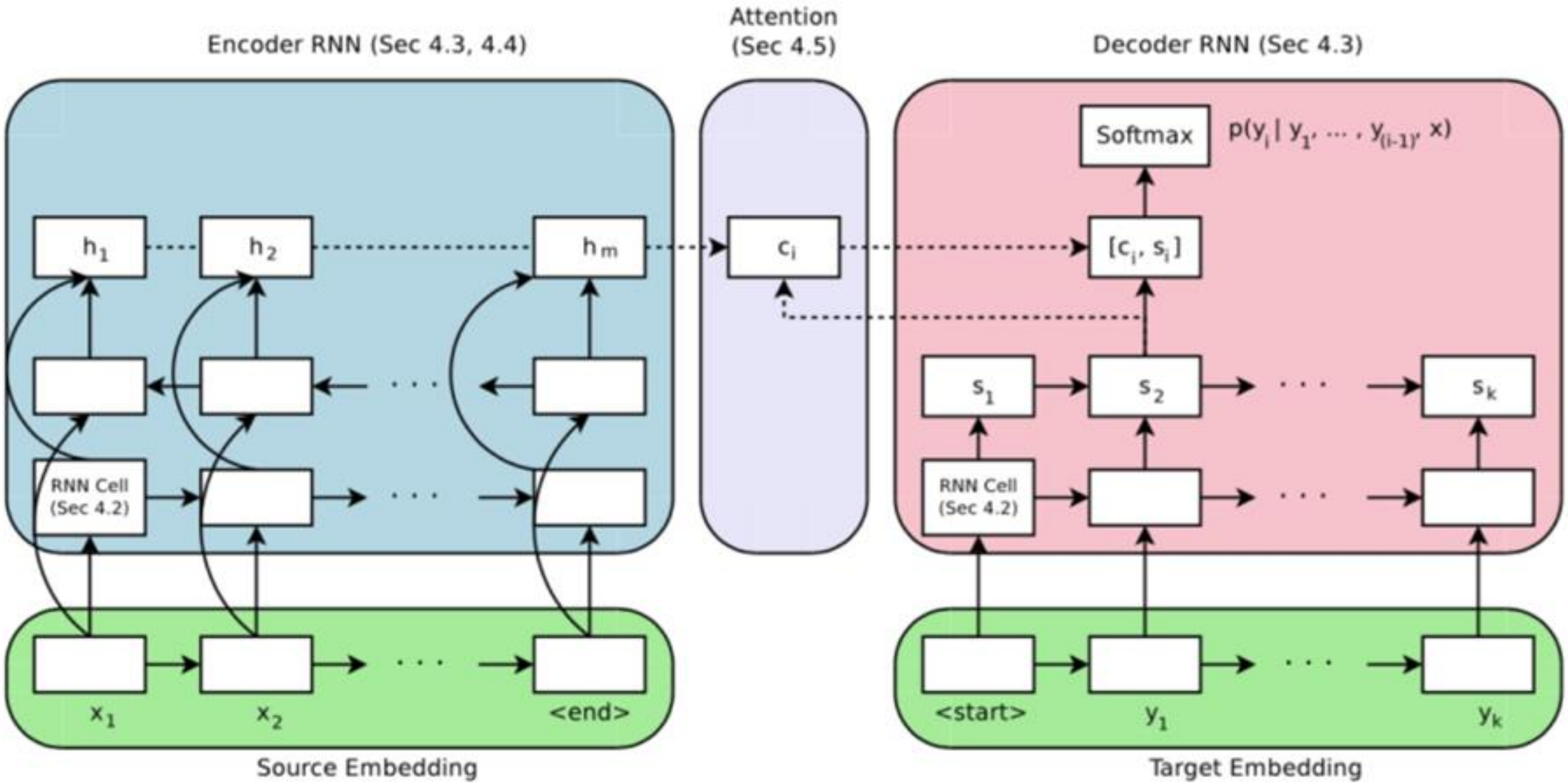
http://dprogrammer.org/rnn-lstm-gru
https://towardsdatascience.com/grus-and-lstm-s-741709a9b9b1

Seq-2-Seq



- Encoder maps a variable-length source sequence (input) to a fixed-length vector
- Decoder maps the vector representation back to a variable-length target sequence (output)
- Two RNNs are trained jointly to maximize the conditional probability of the target sequence given a source sequence

Seq-2-Seq with Attention



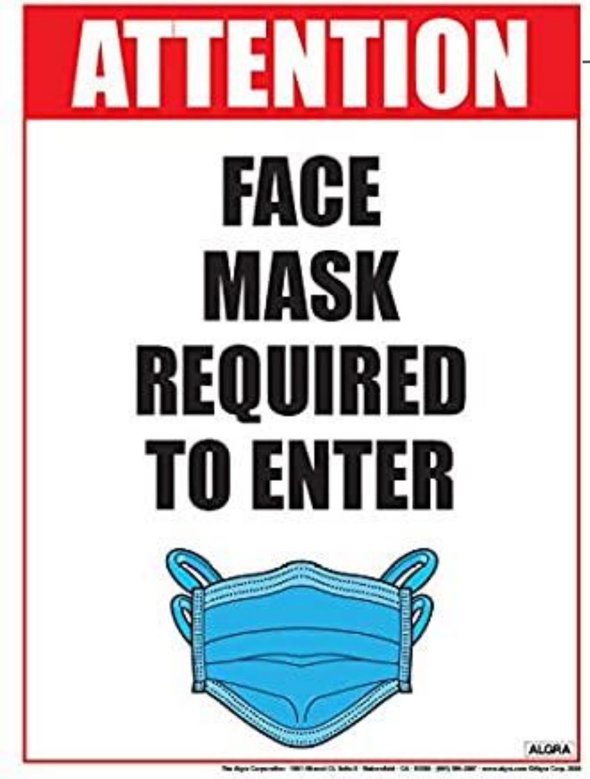
Transformer

Limitations of CNN and RNN

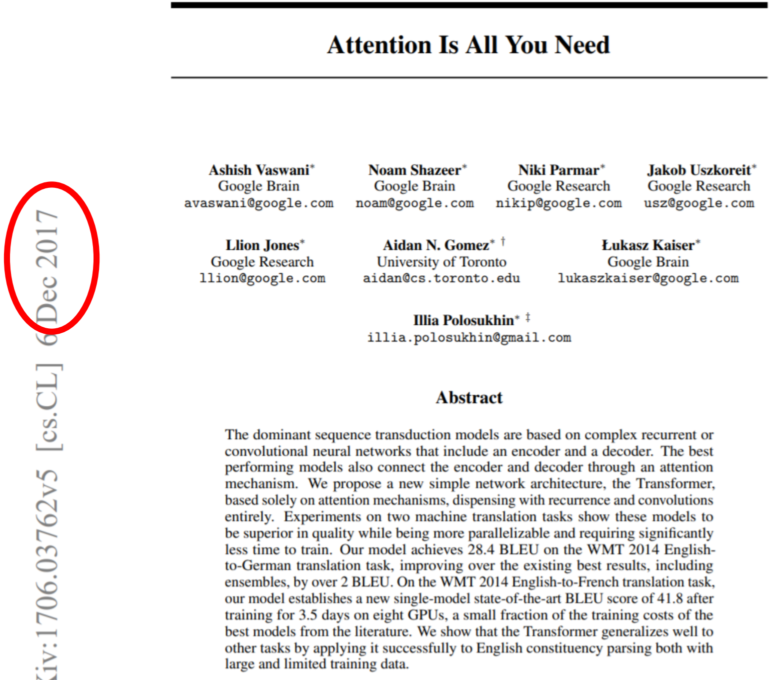
1. “Locality” of the convolution operation
    a. Reduce dimension (compared to fully-connected layers) while maintaining useful local information
    b. It could NOT see two pixels that are far away

2. “Recurrence” of recurrent neural network
    a. It can take an input with arbitrary size (length)
    b. “Vanishing of gradient” problem when sequence length is too long (during backpropagation)

Well, forget about convolution and recurrent



IS ALL YOU NEED



Xiv:1706.03762v5 [cs.CL] 6 Dec 2017

Attention Is All You Need

Ashish Vaswani*
Google Brain
avaswani@google.com

Noam Shazeer*
Google Brain
noam@google.com

Niki Parmar*
Google Research
nikip@google.com

Jakob Uszkoreit*
Google Research
usz@google.com

Llion Jones*
Google Research
llion@google.com

Aidan N. Gomez* †
University of Toronto
aidan@cs.toronto.edu

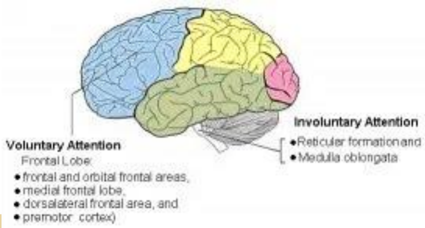
Łukasz Kaiser*
Google Brain
lukaszkaiser@google.com

Illia Polosukhin* ‡
illia.polosukhin@gmail.com

Abstract

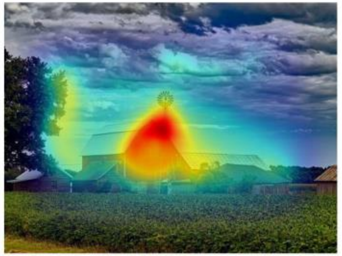
The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.8 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature. We show that the Transformer generalizes well to other tasks by applying it successfully to English constituency parsing both with large and limited training data.

What is attention?



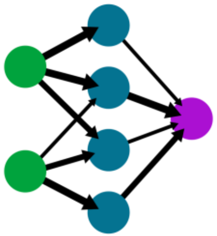
Psychology

Eye-tracking

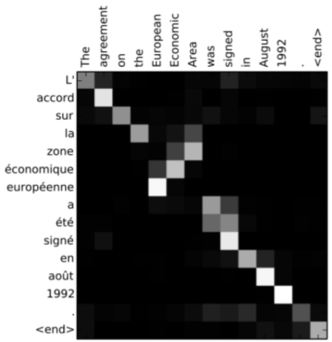


Computer Vision
(Saliency Map)

Computer Vision
(Backpropagation)

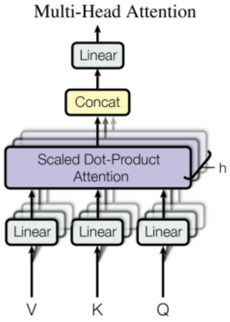


Neural Network
(weights)

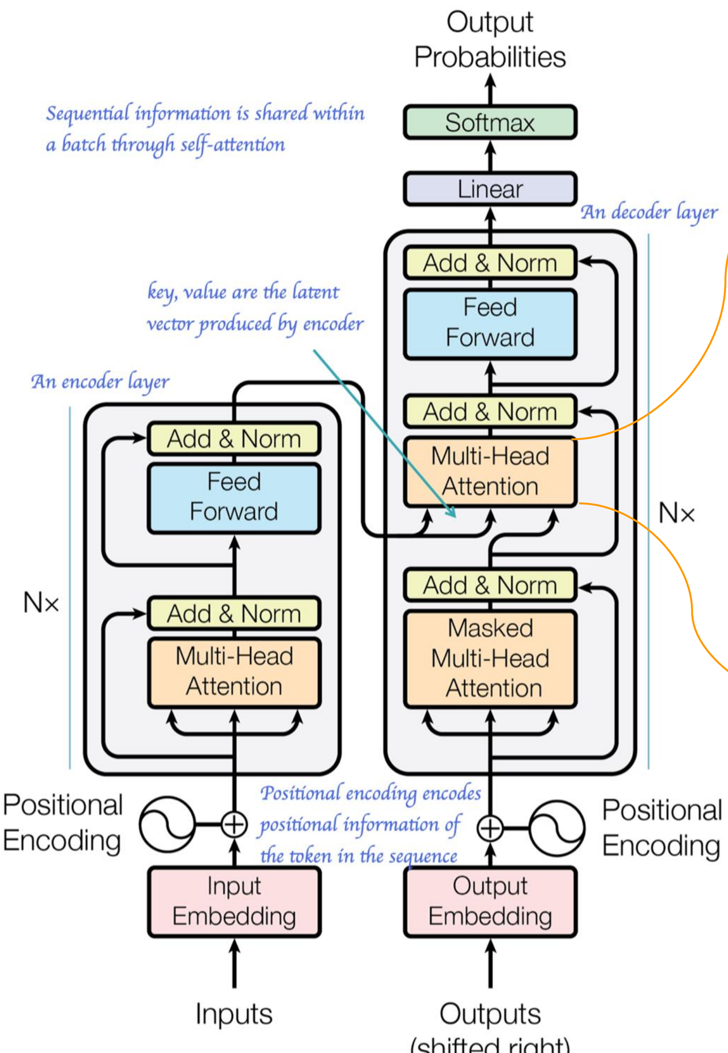


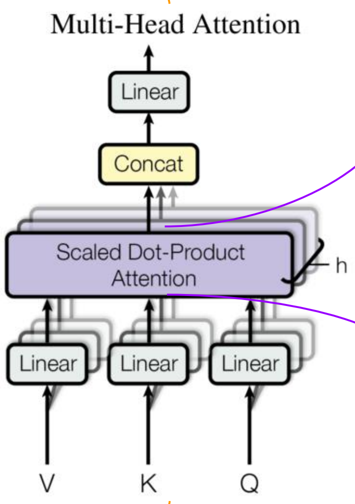
NLP (align)
Bahdanau, Cho, Bengio, 2015, ICLR

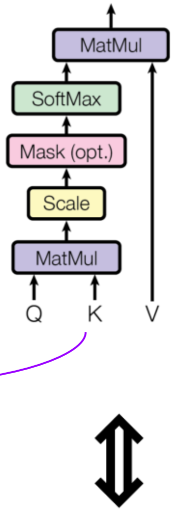
Value, Key, Query



Transformer



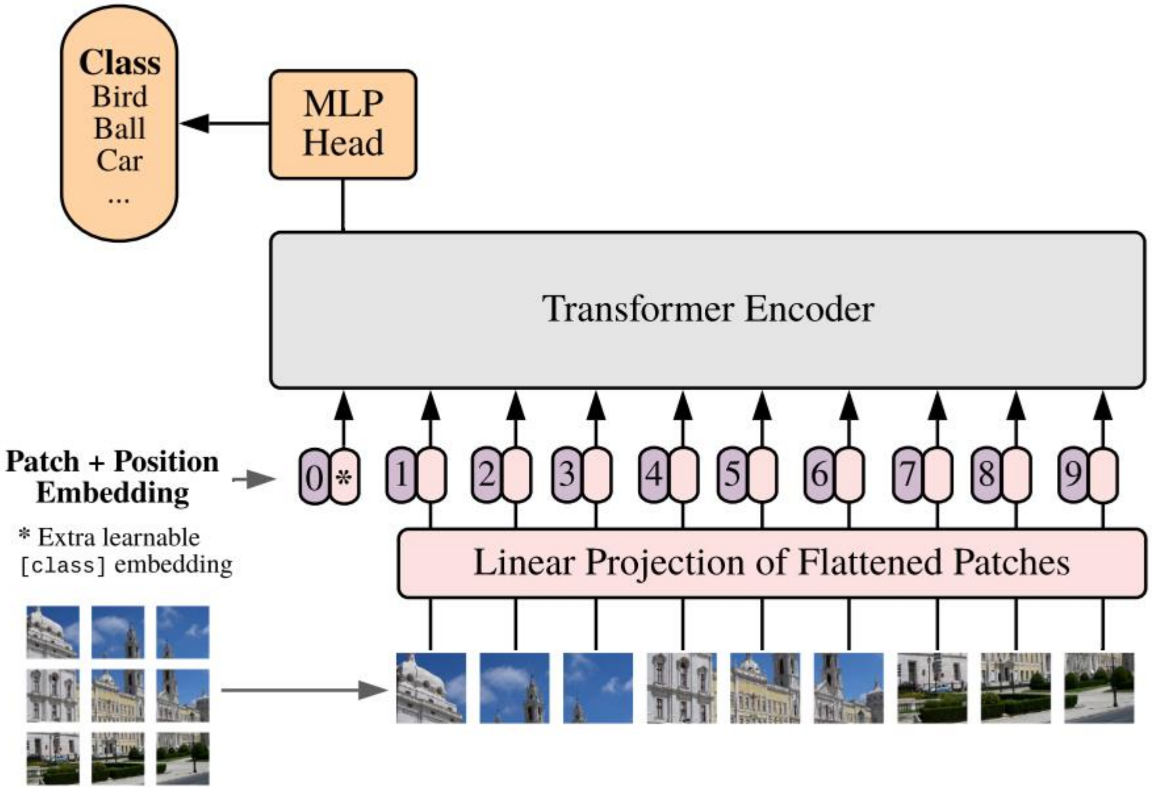




$$\mathrm{Attention}(Q, K, V) = \mathrm{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

http://jalammar.github.io/illustrated-transformer/

Vision Transformer



Limitations of Transformer

1. It cannot learn hierarchical features efficiently (while CNN can)
2. It cannot model periodic finite-state language (while RNN can)
3. It requires lots for computer memory
4. It requires more training data than CNN/RNN (not a big problem)

NN for RL

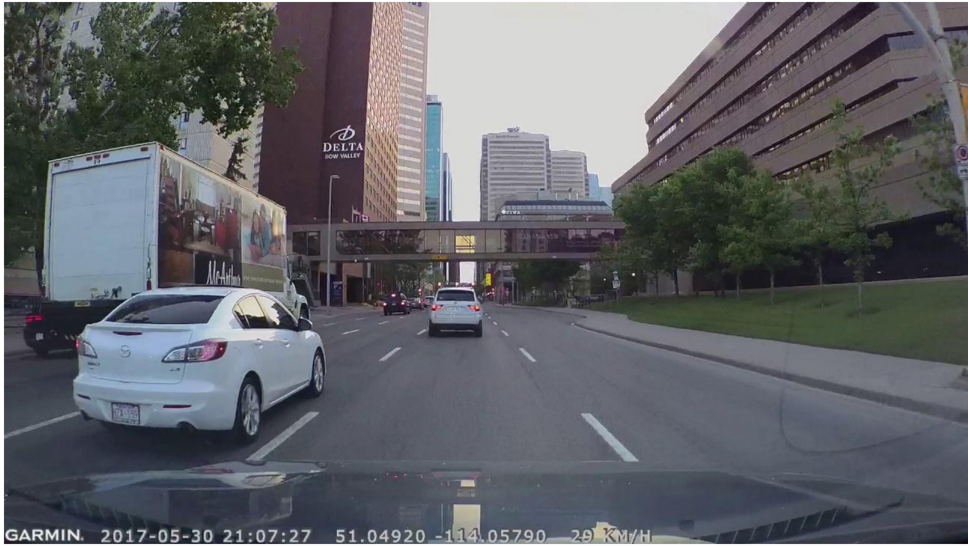
Which Direction

Regression:

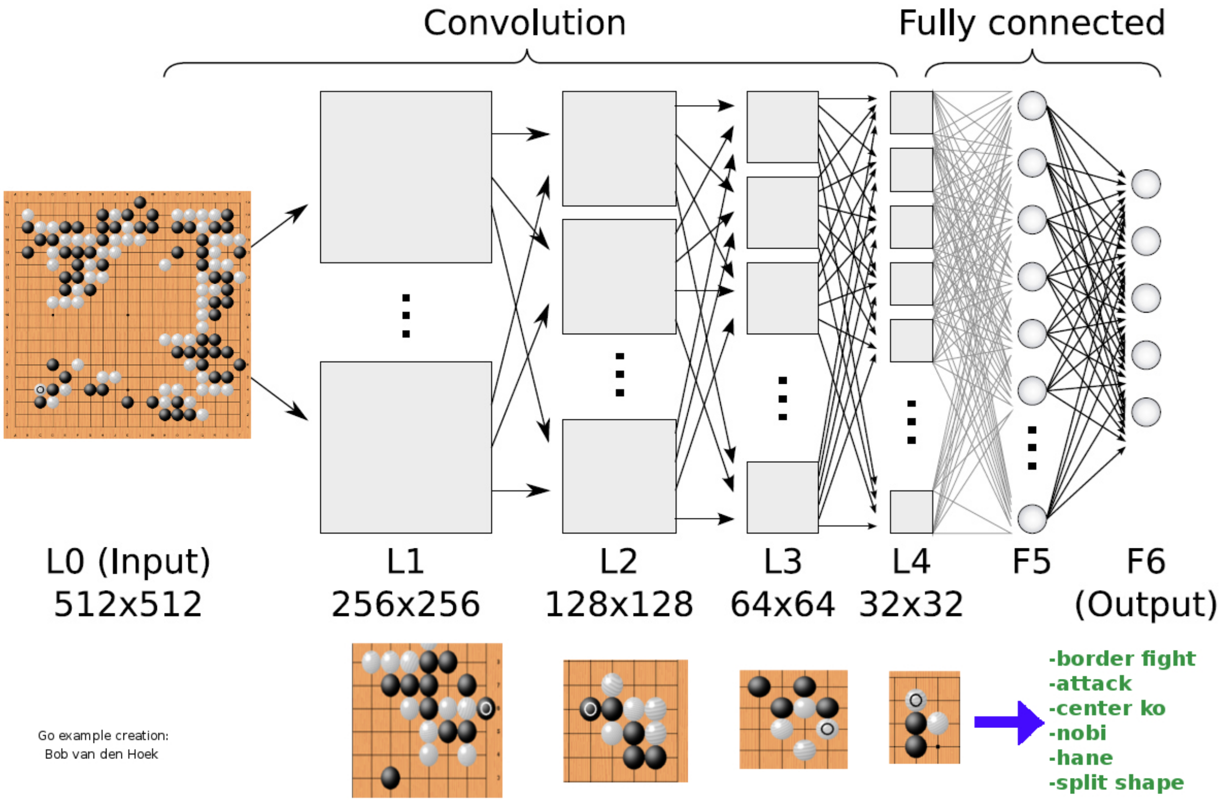
Angle = [-540°, 540°]

Classification:

- Turn left
- Turn right
- Stay Still

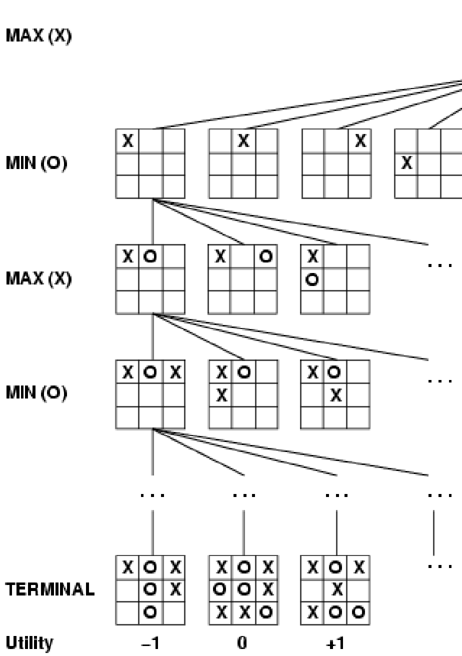


Which Move

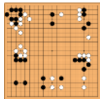


Go example creation:
Bob van den Hoek

Design Utility Function

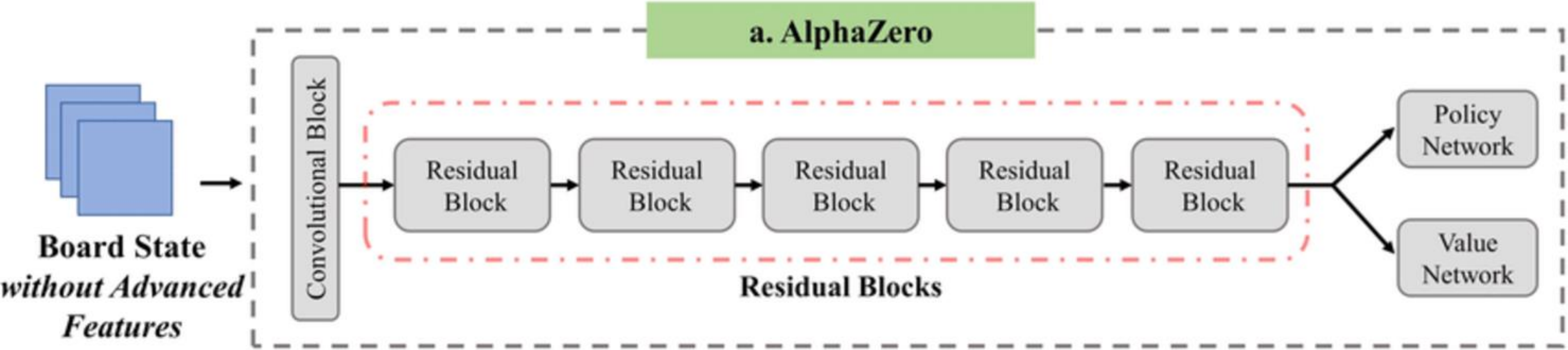


We can train a CNN to replace the manual utility function

Utility = CNN()

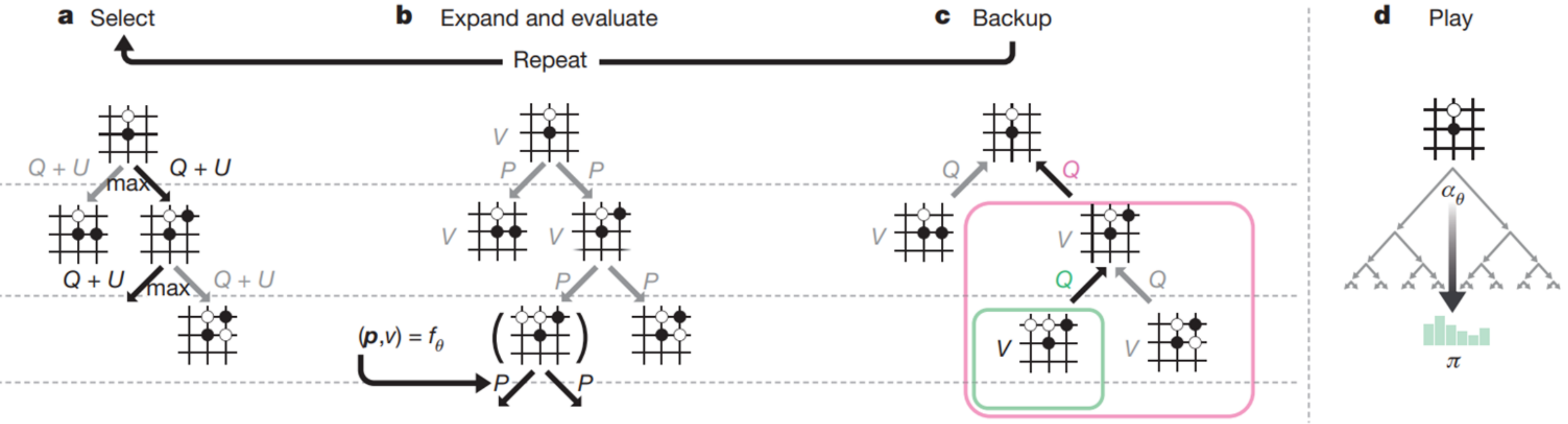
AlphaZero

ResNet backbone
Policy Network
Value Network



Monte Carlo Tree Search

Active Learning to balance Exploration v.s. Exploitation



Intro to PyTorch

Deep Learning Libraries

mxnet

TensorFlow

theano

scikit learn

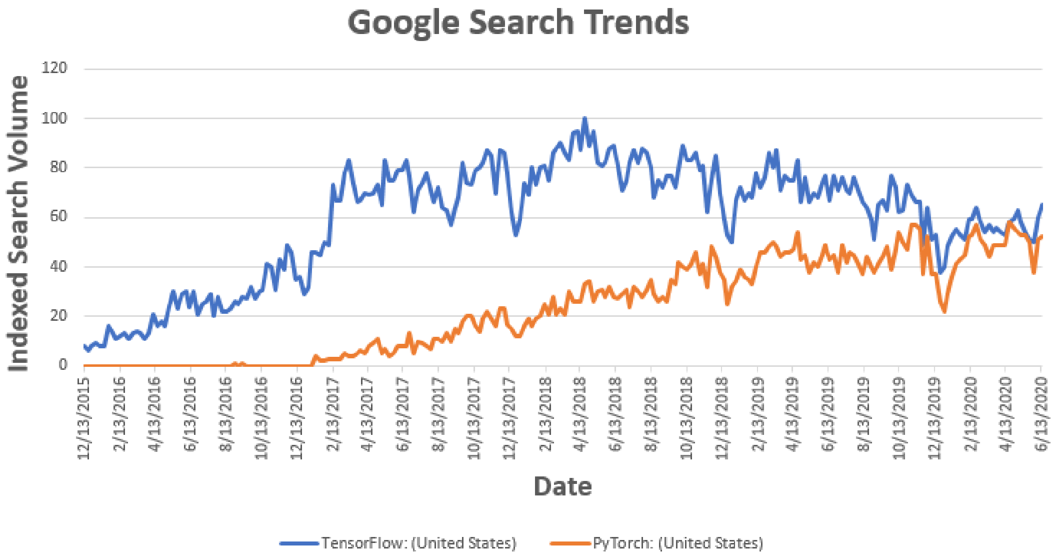
Caffe2

Keras

PYTORCH

Deep Learning Frameworks

- Before 2012: custom C++, MatLab, R, Lua, ... code.
  - Only limited libraries/functions
  - You need to do most things yourself
- MXNet (2015)
- TensorFlow (2015)
- Caffe (2015)
- Torch (2002): Lua
- PyTorch (2016)



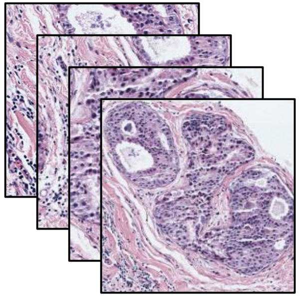
Why PyTorch

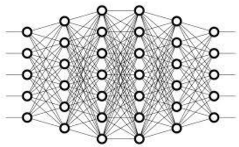
- Autograd
- Dynamic computational graph
- Debugging is easier!
- Data Parallelism (multiple GPU)
- Pythonic-syntax (Python)
- Multiple language support: Python, C++, Java
- Many more!

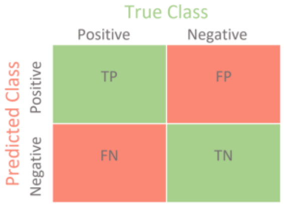
Machine Learning Process

Define the Problem → Collect Data → Train Model → Evaluate → Deploy

Breast Cancer







Model Definition

```python
import torch.nn as nn
import torch.nn.functional as F


class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 16 * 5 * 5)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x


net = Net()
```

Training a model

```
running_loss = 0.0
for i, data in enumerate(trainloader, 0):
    # get the inputs
    inputs, labels = data

    # zero the parameter gradients
    optimizer.zero_grad()

    # forward + backward + optimize
    outputs = net(inputs)
    loss = criterion(outputs, labels)
    loss.backward()
    optimizer.step()

    # print statistics
    running_loss += loss.item()
    if i % 2000 == 1999:    # print every 2000 mini-batches
        print('[%d, %5d] loss: %.3f' %
              (epoch + 1, i + 1, running_loss / 2000))
        running_loss = 0.0
```