머신러닝 딥러닝을 활용한 영상처리

Application Engineer

김종남
Machine Learning
What Can You Do from Machine Learning?

- Object Detection
- Object Recognition or Classification

Object Recognition or Classification

Object Detection
Machine Learning

Machine learning uses data and produces a program to perform a task.

Task: Image Category Recognition

If brightness > 0.5
then ‘hat’
If edge_density < 4 and
major_axis > 5
then “boat”

...
Machine Learning Workflow Using Images

Training Data → Feature Extraction → Learning or Modelling → Training

Input Image → Feature Extraction → Classification

Classifier / Model → ‘hat’
Cascade of Classifiers in CascadeObjectDetector

- Each stage of cascade is Gentle Adaboost, an ensemble of weak learners
- Each stage rejects negative samples using a weighted vote of these weak learners
- The samples not rejected are passed to the next stage
- Positive detection means the sample passed all stages of the cascade
Challenges: Machine Learning Workflow Using Images

Training Data

Feature Extraction

Learning or Modelling

Challenge 1

Challenge 2

Challenge 3

Challenge 2

Classifier / Model

‘hat’

Input Image

Feature Extraction

Classification

Demo
Challenges: Machine Learning Workflow Using Images

Training Data

Feature Extraction

Learning or Modelling

Challenge 1

Challenge 2

Challenge 3

Input Image

Feature Extraction

Classification

Classifier / Model

‘hat’
Common Challenges for Machine Learning with Images

- **Challenge 1:** Handling large sets of images

- **Challenge 2:** How to extract discriminative information from images

- **Challenge 3:** How to model problem using machine learning techniques

- Easy to handle large sets of images
  - `imageSet`

- Bag of words for feature extraction
  - More available in Computer Vision System Toolbox
Deep Learning
Deep Learning is Ubiquitous

**Computer Vision**
- Pedestrian and traffic sign detection
- Landmark identification
- Scene recognition
- Medical diagnosis and drug discovery

**Text and Signal Processing**
- Speech Recognition
- Speech & Text Translation

**Robotics & Controls**

and many more…
What is Deep Learning?

Deep learning performs **end-end learning** by learning **features, representations and tasks** directly from **images, text and sound**.

Traditional Machine Learning

- Manual Feature Extraction
- Classification
- Machine Learning
  - Car ✓
  - Truck ✗
  - Bicycle ✗

Deep Learning approach

- Convolutional Neural Network (CNN)
- End-to-end learning
- Feature learning + Classification
  - Car ✓
  - Truck ✗
  - Bicycle ✗
Demo: Live Object Recognition with Webcam
Why is Deep Learning so Popular?

- **Results:** Achieved substantially better results on ImageNet large scale recognition challenge
  - 95% + accuracy on ImageNet 1000 class challenge

- **Computing Power:** GPU’s and advances to processor technologies have enabled us to train networks on massive sets of data.

- **Data:** Availability of storage and access to large sets of labeled data
  - E.g. ImageNet, PASCAL VoC, Kaggle

### Table: Year vs. Error Rate

<table>
<thead>
<tr>
<th>Year</th>
<th>Error Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre-2012 (traditional computer vision and machine learning techniques)</td>
<td>&gt; 25%</td>
</tr>
<tr>
<td>2012 (Deep Learning)</td>
<td>~ 15%</td>
</tr>
<tr>
<td>2015 (Deep Learning)</td>
<td>&lt;5%</td>
</tr>
</tbody>
</table>
Two Approaches for Deep Learning

1. Train a Deep Neural Network from Scratch
   - Convolutional Neural Network (CNN)
     - Learned features
     - Lots of data
     - New Task
     - Car ✔
     - Truck ❌
     - Bicycle ❌

2. Fine-tune a pre-trained model (transfer learning)
   - Pre-trained CNN
   - Fine-tune network weights
   - Medium amounts of data
   - New Task
   - Car ✔
   - Truck ❌
Two Deep Learning Approaches

Approach 1: Train a Deep Neural Network from Scratch

- **Training data**: 1000s to millions of labeled images
- **Computation**: Compute intensive (requires GPU)
- **Training Time**: Days to Weeks for real problems
- **Model accuracy**: High (can over fit to small datasets)

**Recommended only when:**

<table>
<thead>
<tr>
<th>Training data</th>
<th>1000s to millions of labeled images</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computation</td>
<td>Compute intensive (requires GPU)</td>
</tr>
<tr>
<td>Training Time</td>
<td>Days to Weeks for real problems</td>
</tr>
<tr>
<td>Model accuracy</td>
<td>High (can over fit to small datasets)</td>
</tr>
</tbody>
</table>
Two Deep Learning Approaches

Approach 2: Fine-tune a pre-trained model (transfer learning)

CNN trained on massive sets of data
- Learned robust representations of images from larger data set
- Can be fine-tuned for use with new data or task with small – medium size datasets

New Data

Recommended when:

<table>
<thead>
<tr>
<th>Training data</th>
<th>100s to 1000s of labeled images (small)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computation</td>
<td>Moderate computation (GPU optional)</td>
</tr>
<tr>
<td>Training Time</td>
<td>Seconds to minutes</td>
</tr>
<tr>
<td>Model accuracy</td>
<td>Good, depends on the pre-trained CNN model</td>
</tr>
</tbody>
</table>
Convolutional Neural Networks

- Train “deep” neural networks on structured data (e.g. images, signals, text)
- Implements Feature Learning: Eliminates need for “hand crafted” features
- Trained using GPUs for performance
Convolution Layer

- Core building block of a CNN
- Convolve the filters sliding them across the input, computing the dot product

Intuition: learn filters that activate when they “see” some specific feature
Rectified Linear Unit (ReLU) Layer

- Frequently used in combination with Convolution layers
- Do not add complexity to the network
- Most popular choice: $f(x) = \max(0, x)$, activation is thresholded at 0
**Pooling Layer**

- Perform a **downsampling** operation across the spatial dimensions
- Goal: progressively decrease the size of the layers
- Max pooling and average pooling methods
- Popular choice: Max pooling with 2x2 filters, Stride = 2

### Max pooling

```
  4  8
  5  6
```

![Max pooling example](image)

### Average pooling

```
  2  5
  3  4
```

![Average pooling example](image)
# Challenges using Deep Learning for Computer Vision

<table>
<thead>
<tr>
<th>Steps</th>
<th>Challenge</th>
</tr>
</thead>
<tbody>
<tr>
<td>Importing Data</td>
<td>Managing large sets of labeled images</td>
</tr>
<tr>
<td>Preprocessing</td>
<td>Resizing, Data augmentation</td>
</tr>
<tr>
<td>Choosing an architecture</td>
<td>Background in neural networks (deep learning)</td>
</tr>
<tr>
<td>Training and Classification</td>
<td>Computation intensive task (requires GPU)</td>
</tr>
<tr>
<td>Iterative design</td>
<td></td>
</tr>
</tbody>
</table>
Demo: Classifying the CIFAR-10 dataset

Objective: Train a Convolutional Neural Network to classify the CIFAR-10 dataset

Data:

<table>
<thead>
<tr>
<th>Input Data</th>
<th>Thousands of images of 10 different Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Response</td>
<td>AIRPLANE, AUTOMOBILE, BIRD, CAT, DEER, DOG, FROG, HORSE, SHIP, TRUCK</td>
</tr>
</tbody>
</table>

Approach:

- Import the data
- Define an architecture
- Train and test the CNN

Demo: Classifying the CIFAR-10 dataset

%% Download the CIFAR-10 dataset
if ~exist('cifar-10-batches-mat','dir')
cifar10Dataset = 'cifar-10-matlab';
disp('Downloading 174MB CIFAR-10 dataset...');
websave(fullfile(cifar10Dataset, 'train.mat'))
Demo

Fine-tune a pre-trained model (transfer learning)

New Data

Pre-trained CNN
(AlexNet – 1000 Classes)

Car

SUV

New Task – 2 Class Classification
Demo

Fine-tune a pre-trained model (transfer learning)

% This example shows how to fine-tune a pre-trained deep convolutional neural network (CNN) for a new recognition task.

%% Load network

cnnMatFile = fullfile(pwd, 'imagenet-cnn.mat');
if ~exist(cnnMatFile, 'file')
    disp('Run downloadAndPrepareCNN.m to download and prepare the network.')
else
    load(cnnMatFile)
end
## Addressing Challenges in Deep Learning for Computer Vision

<table>
<thead>
<tr>
<th>Challenge</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Managing large sets of labeled images</td>
<td><code>imageSet</code> or <code>imageDataStore</code> to handle large sets of images</td>
</tr>
<tr>
<td>Resizing, Data augmentation</td>
<td><code>imresize</code>, <code>imcrop</code>, <code>imadjust</code>, <code>imageInputLayer</code>, etc.</td>
</tr>
<tr>
<td>Background in neural networks (deep learning)</td>
<td>Intuitive interfaces, well-documented architectures and examples</td>
</tr>
<tr>
<td>Computation intensive task (requires GPU)</td>
<td>Training supported on GPUs</td>
</tr>
<tr>
<td></td>
<td>No GPU expertise is required</td>
</tr>
<tr>
<td></td>
<td>Automate. Offload computations to a cluster and test multiple architectures</td>
</tr>
</tbody>
</table>
FPGA/ASIC 설계를 위한 모델 기반 설계

Application engineer
김 종 남
Agenda

- Introduction to Model Based Design
- System Design and HDL Code Generation
- HDL Co-simulation and FPGA in-the-loop
- Summary
Workflow Without HDL Coder

- Long development cycles
- Prevents short iteration cycles
- Difficult to optimize the algorithm at a system level

HDL Code Creation

Fixed Point Conversion

FPGA Verification

HDL Refinement

HDL Verification

HDL Verification

Algorithm Development

MATLAB Simulink Stateflow

DESIGN
Separate Views of DSP Implementation

System Designer

- Algorithm Design
  - Fixed-Point
  - Timing / Control Logic
  - Architecture Exploration
  - Algorithms / IP
- System Test Bench
  - Environment Models
  - Analog Models
  - Digital Models
  - Algorithms / IP
- FPGA Requirements
  - Hardware Specification
  - Test Stimulus

FPGA Designer

- RTL Design
  - IP Interfaces
  - HW Architecture
- Implement Design
  - Synthesis
  - Map
  - Place & Route
- Verification
  - Behavioral Simulation
  - Functional Simulation
  - Static Timing Analysis
  - Timing Simulation
  - Back Annotation
- FPGA Hardware
Model-Based Design for Implementation

MATLAB® and Simulink®
Algorithm and System Design

Algorithm Design
- Fixed-Point
- Timing / Control Logic
- Architecture Exploration
- Algorithms / IP

System Test Bench
- Environment Models
- Analog Models
- Digital Models
- Algorithms / IP

FPGA Requirements
- Hardware Specification
- Test Stimulus

RTL Design
- IP Interfaces
- Hardware Architecture

Verification
- Behavioral Simulation
- Functional Simulation
- Static Timing Analysis
- Timing Simulation
- Back Annotation

Implement Design
- Synthesis
- Map
- Place & Route

FPGA Hardware

Algorithm and System Design

MATLAB® and Simulink®
Algorithm and System Design

Algorithm Design
- Fixed-Point
- Timing / Control Logic
- Architecture Exploration
- Algorithms / IP

System Test Bench
- Environment Models
- Analog Models
- Digital Models
- Algorithms / IP

FPGA Requirements
- Hardware Specification
- Test Stimulus

RTL Design
- IP Interfaces
- Hardware Architecture

Verification
- Behavioral Simulation
- Functional Simulation
- Static Timing Analysis
- Timing Simulation
- Back Annotation

Implement Design
- Synthesis
- Map
- Place & Route

FPGA Hardware
Model-Based Design for Implementation

MATLAB® and Simulink®
Algorithm and System Design
Model Refinement for Hardware

Automatic HDL Code Generation

RTL Design
- IP Interfaces
- Hardware Architecture

Verification
- Behavioral Simulation
- Functional Simulation
- Static Timing Analysis
- Timing Simulation
- Back Annotation

Implement Design
- Synthesis
- Map
- Place & Route

FPGA Hardware

MathWorks
Model-Based Design for Implementation

MATLAB® and Simulink®
Algorithm and System Design
Model Refinement for Hardware

Automatic HDL Code Generation → HDL Co-Simulation → Behavioral Simulation

Verification
- Behavioral Simulation
- Functional Simulation
- Static Timing Analysis
- Timing Simulation
- Back Annotation

Implement Design
- Synthesis
- Map
- Place & Route

FPGA Hardware
Model-Based Design for Implementation

MATLAB® and Simulink®
Algorithm and System Design
Model Refinement for Hardware

Automatic HDL Code Generation
HDL Co-Simulation

Behavioral Simulation
Back Annotation

Implement Design
Verification

Functional Simulation
Static Timing Analysis
Timing Simulation
Back Annotation

Synthesis
Map
Place & Route

FPGA Hardware
Model-Based Design for Implementation

- MATLAB® and Simulink®
  - Algorithm and System Design
  - Model Refinement for Hardware

- Automatic HDL Code Generation

- HDL Co-Simulation
  - Behavioral Simulation
  - Back Annotation

- Implement Design
  - Synthesis
  - Map
  - Place & Route

- Verification
  - Functional Simulation
  - Static Timing Analysis
  - Timing Simulation

- FPGA Hardware
  - FPGA-in-the-Loop
Model-Based Design for Implementation

Integrated Workflow

- MATLAB® and Simulink®
  Algorithm and System Design
  Model Refinement for Hardware

- Automatic HDL Code Generation

- HDL Co-Simulation
  Behavioral Simulation

- Implement Design
  Synthesis
  Map
  Place & Route

- Verification
  Functional Simulation
  Static Timing Analysis
  Timing Simulation

- FPGA Hardware
  FPGA-in-the-Loop

All steps from 1 single GUI
Agenda

- Introduction to Model Based Design
- System Design and HDL Code Generation
- HDL Co-simulation and FPGA in-the-loop
- Summary
Simulink at a glance

*The leading environment for system-level modeling and simulation*

- Block-diagram environment
- Model, simulate and analyze dynamic systems
- Extensive library of predefined blocks
- Fully integrated with MATLAB
Fixed-Point Analysis
Simulink Fixed-Point

- Convert floating-point to **optimized** fixed-point models
  - Automatic tracking of signal range for both Simulink blocks **and** MATLAB function block
  - Using simulation and/or static analysis
  - Word / Fraction lengths proposal
Fixed-Point Analysis

Simulink Fixed-Point

- Automatically compare simulation results, e.g. Floating-Point vs. Fixed-Point
Automatic HDL Code Generation

HDL Coder

Automatically generate bit-true, cycle-accurate HDL code from Simulink, MATLAB and Stateflow

Full bi-directional traceability!!
Speed optimization
Use pipelining to improve speed

Critical Path highlighting makes it easier to identify the true bottlenecks of the system

Advanced Pipelining options for pipeline distribution and automatic delay compensation
Critical Timing Path Analysis
Resource Folding Algorithms

- **Goal**
  - Area reduction

- **Means**
  - Time-multiplexed re-use of resources

- **Algorithms**
  - Resource Sharing
    - Re-use of identical operators or atomic subsystems within algorithm
  - Resource Streaming
    - Re-use of vectorized operators or subsystems
Area optimization
Use sharing and streaming to reduce area

Use sharing and streaming to reduce area

<table>
<thead>
<tr>
<th>Resource</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multipliers</td>
<td>10</td>
</tr>
<tr>
<td>Adders/Subtractors</td>
<td>36</td>
</tr>
<tr>
<td>Registers</td>
<td>292</td>
</tr>
<tr>
<td>RAMs</td>
<td>2</td>
</tr>
<tr>
<td>Multiplexers</td>
<td>116</td>
</tr>
</tbody>
</table>

Automatically generated validation models

Resource utilization reports provide early feedback on resource utilization
Agenda

- Introduction to Model Based Design
- System Design and HDL Code Generation
- HDL Co-simulation and FPGA in-the-loop
- Summary
Verification Landscape:

Model
- Requirements
- Functional
- Equivalence
- Coverage

VHDL / Verilog
- Requirements
- Equivalence
- Coverage

FPGA
- Equivalence
- Regression
- Timing Analysis
- Optimization
HDL Co-Simulation to verify HDL
Re-use System Level Test Bench for HDL Verification

Model
- Requirements
- Functional
- Equivalence
- Coverage
- Property Proving
- Virtual Platforms

VHDL / Verilog
- Requirements
- Equivalence
- Coverage
- Assertions

FPGA
- Equivalence
- Regression
- Timing Analysis
- On target prototyping
HDL Co-Simulation to verify HDL
Re-use System Level Test Bench for HDL Verification

- Re-use test benches for equivalence checking
- Integrate with HDL code coverage analysis
- Flexible test bench creation: closed loop, multi domain
- Also works with handwritten code
- Integrate with Modelsim/Questa and Incisive
Integrate with HDL Code Coverage

HDL Verifier

- Re-use system level test bench
- Combine analysis in HDL Simulator and MATLAB/Simulink
FPGA-in-the-loop
Enable regression testing with FPGA-in-the-loop simulation

Model
- Requirements
- Functional
- Equivalence
- Coverage
- Property Proving
- Virtual Platforms

VHDL / Verilog
- Requirements
- Equivalence
- Coverage
- Assertions

FPGA
- Equivalence
- Regression
- Timing Analysis
- On target prototyping
FPGA-in-the-loop
Enable regression testing with FPGA-in-the-loop simulation

- Re-use test benches for regression testing
- Integrate with Altera / Xilinx FPGA Development Boards
- Flexible test bench creation: closed loop, multi domain
- Also works with handwritten code
FPGA-in-the-Loop for Regression Tests
HDL Verifier

- Re-use system level test bench
- Accelerate Verification with FPGA Hardware
- Use application specific analysis methods
Model-Based Design for Implementation

Verify Model/HDL/FPGA:
- Structural testing and automatic test generation of Models and HDL
- Regression testing with FPGA-in-the-Loop
- Rapidly prototype algorithms on FPGA evaluation boards
Summary

- **#1: Automatic HDL code generation**
  - Rapidly explore implementation options
  - What-if analysis for Area / Speed / Power

- **#2: Integrate separated workflows**
  - One model, one language, better collaboration
  - Easily make algorithm trade-offs which impact performance

- **#3: HDL/FPGA Co-simulation**
  - Combine Simulink analysis methods with FPGA/HDL analysis
  - Use the best of both worlds ➔ flexible test bench creation
Thank You!

Questions?