

CSED499I-01

## Research Project Final Report

### **Embedded *MDNet***

*Mobile Embedded Multi-Domain Convolutional Neural Networks for Visual Tracking*

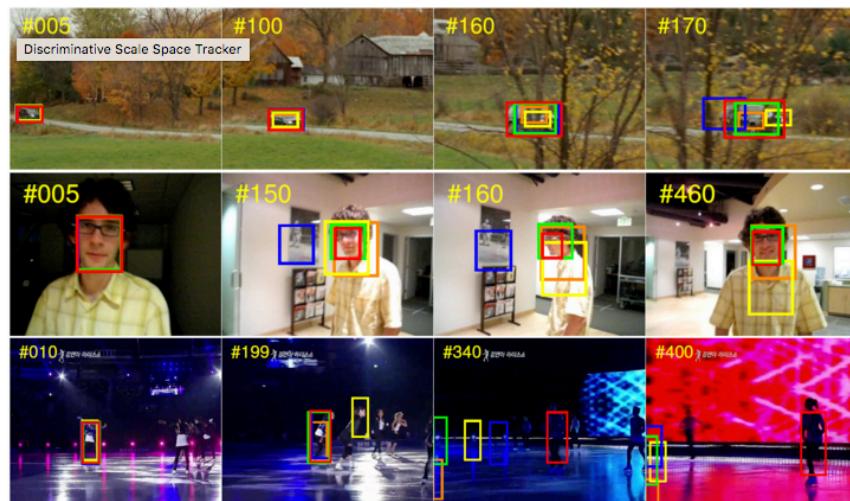
20080650 CSE Jeongbin Choe

Advisor: Prof. Bohyung Han, CV Lab

## 1. Introduction

### A. Visual Tracking

Visual Tracking은 Computer Vision의 한 분야로, 연속된 이미지 시퀀스에서 움직이는 오브젝트를 인식하고 그 위치를 추적하는 영상 추적 기술이다. 일반적으로 매 프레임마다 해당 오브젝트를 인식하며, 관련하여 수많은 알고리즘이 존재한다.



### B. Convolutional Neural Network (CNN)

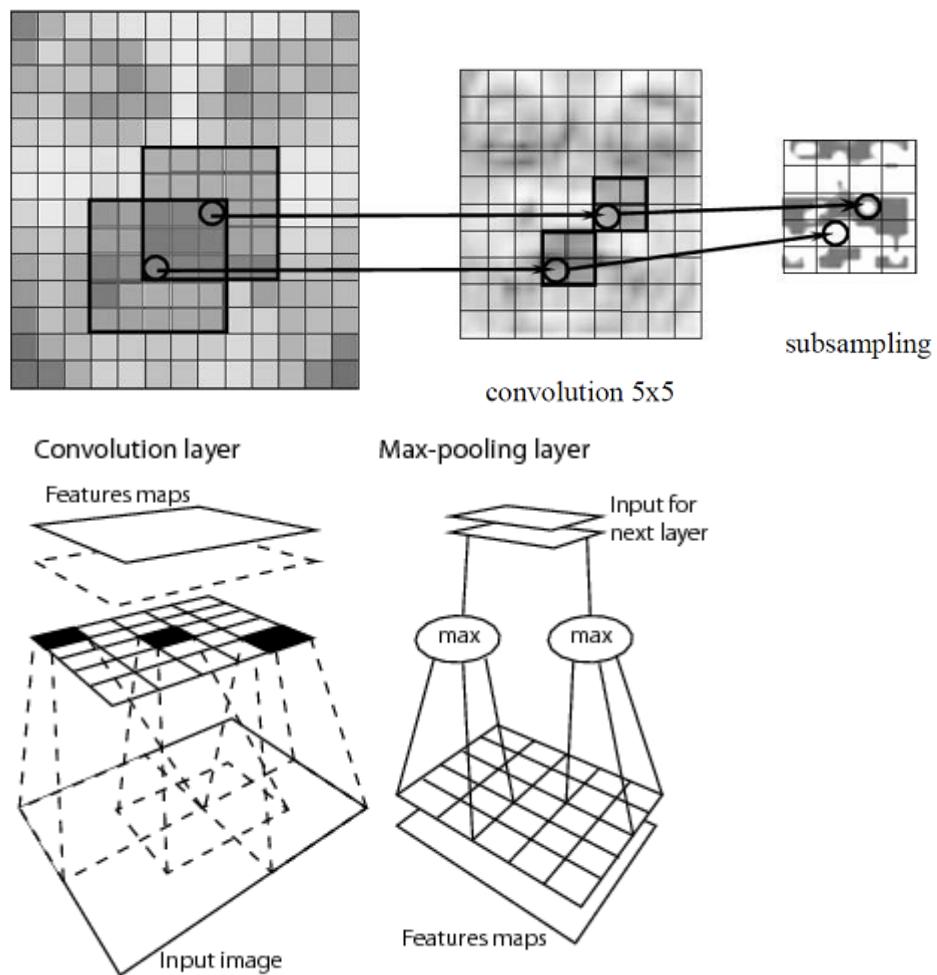
1989년 Y. LeCun의 논문에서 처음 소개가 된 Deep Neural Network의 일종이다. 최초에는 필기체 zip code 인식을 위한 프로젝트에서 개발이 되었지만 이후 다양한 분야에 이용되고 있다.

CNN은 기존 Multi-layered Neural Network에 비해 아래의 두 가지 특징을 갖는다.

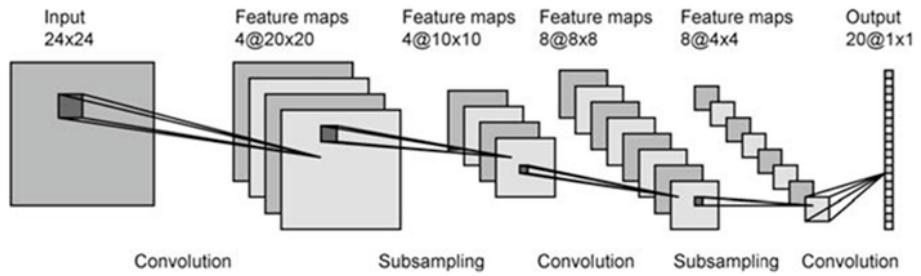
- **Locality:** CNN은 영상 분석시 공간적으로 인접한 정보를 활용하는데, Subsampling 과정을 거치면서 영상의 크기를 줄이고 local feature 들에 대한 filter 연산을 반복적으로 적용하며 점차 global feature 를 얻는다.
- **Shared Weights:** 동일한 weight 를 갖는 filter 를 전체 영상에

반복적으로 적용함으로써 변수의 수를 획기적으로 줄일 수 있으며, topology 변화에 무관한 invariance 를 얻을 수 있게 된다.

CNN 은 Convolution Layer 와 Pooling Layer 로 구성되는데, Convolution Layer 에서는 feature 를 추출하기 위한 filter 가 적용되고, Pooling Layer 에서는 max-pooling 방식으로 subsampling 과정을 거친다.



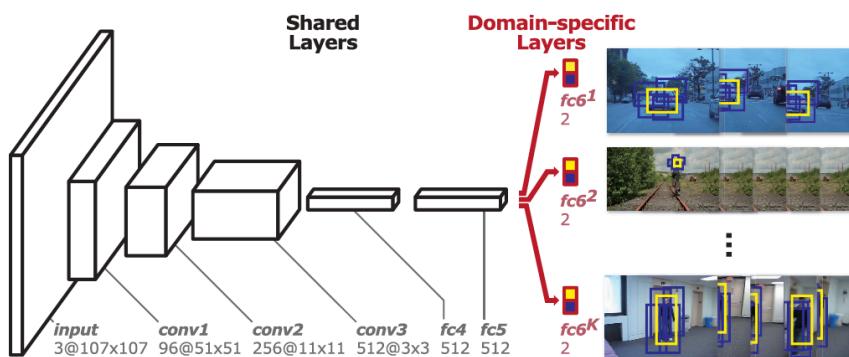
결과적으로, input 영상으로부터 convolution 을 수행해 feature map 을 만들고, pooling 을 통해 feature map 의 크기를 줄인다. 보통의 경우에는 1 개의 convolution 에 대해 1 개의 pooling 연산을 수행한다.

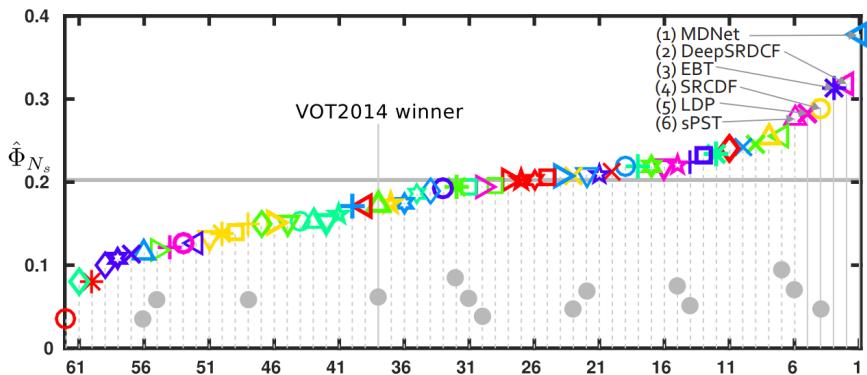


여러 단의 convolution + pooling 과정을 거치면, feature map의 크기는 줄어들고 전체를 대표할 수 있는 global 한 feature 들이 남게 된다. 이렇게 얻어진 feature 가 fully connected network 의 입력으로 연결이 되어, 최적의 인식 결과를 낼 수 있게 되는 것이다. 이러한 특징 덕에, CNN 은 특히 Computer Vision 분야에서 널리 사용되고 있다.

### C. MDNet

*MDNet*은 Multi-Domain Learning Network로, 2015년 H. Nam의 논문에서 소개된 Visual Tracking 알고리즘이다. 3 단의 CNN과 3 단의 fully-connected layer로 이루어져 있으며, individual 한 training sequences 에 대해 1 대 1로 대응하는 domain-specific layers 가 output 단에 존재한다. 이 알고리즘은 VOT2015에서 여러 state of the art 알고리즘들을 제치고 최종 winner 가 되었다.





이 알고리즘은 MATLAB 으로 구현되어 있으며, 전체 코드는 Github 에 공개되어 있다. 논문에 의하면, 8 cores Intel Xeon Processor 와 NVIDIA Tesla GPU 의 환경에서 약 1 fps 로 작동한다.

#### D. Project Goal

*MDNet* 이 모바일 디바이스 환경에서 구현되었을 때 어떤 성능을 보여줄지 궁금했고, 그에 의해 *MDNet* 을 모바일 디바이스에 porting 하는 것이 이번 과제연구의 주제가 되었다. 개발에 사용한 디바이스는 Apple iPhone 6+이며, 주제를 정리하면 아래와 같다.

- *MDNet* 의 네트워크와 알고리즘을 그대로 iPhone 6+에 porting 한 후, performance accuracy 를 최대한 유지하면서 optimization 을 수행하여, 원래의 1 fps 에 가깝게 작동하도록 만든다.

(iPhone 6+ 환경: Apple A8 processor Dual-core 1.4GHz Typhoon, PowerVR GX6450)

- *MDNet* based real-time learning and object tracking application 을 제작하여, 디바이스의 카메라를 통해 real-time 으로 visual tracking 을 수행해본다.

이 보고서를 쓰는 시점에서 Embedded *MDNet* 의 결과물은 약 0.40 fps 의 performance 를 보여주었다.

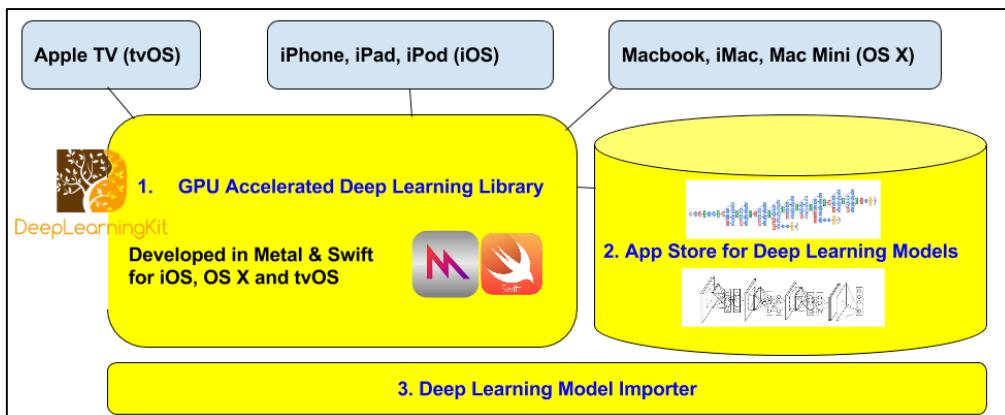
## 2. Requirements

이번 프로젝트를 진행하기 위해서는 아래의 두 가지가 요구되었다.

- 1) iOS에서 작동할 수 있는 Swift나 Objective-C로 구현된 CNN framework
- 2) 성능 확보를 위한 Apple의 GPU 가속 library인 Metal을 지원하는 framework

*DeepLearningKit*이라는 Open Source framework가 위의 두 가지 요구조건을 충족하는 것을 확인했고, 그 외에는 두 조건을 충족하는 framework를 찾을 수 없었다. 결과적으로, 이번 프로젝트의 구현에 *DeepLearningKit*이 사용되었다.

*DeepLearningKit*은 2015년 12월에 시작된 Open Source 프로젝트로, Apple의 개발 언어인 Swift로 제작된 CNN framework이며, Metal을 이용해 GPU 가속을 지원한다. 위의 요구 조건을 충족하며, 더구나 *MDNet*에도 사용되었던 Caffe 기반의 CNN이었기에, 이번 프로젝트에 가장 적합하다고 생각했다.



## 3. Implemented Architectures

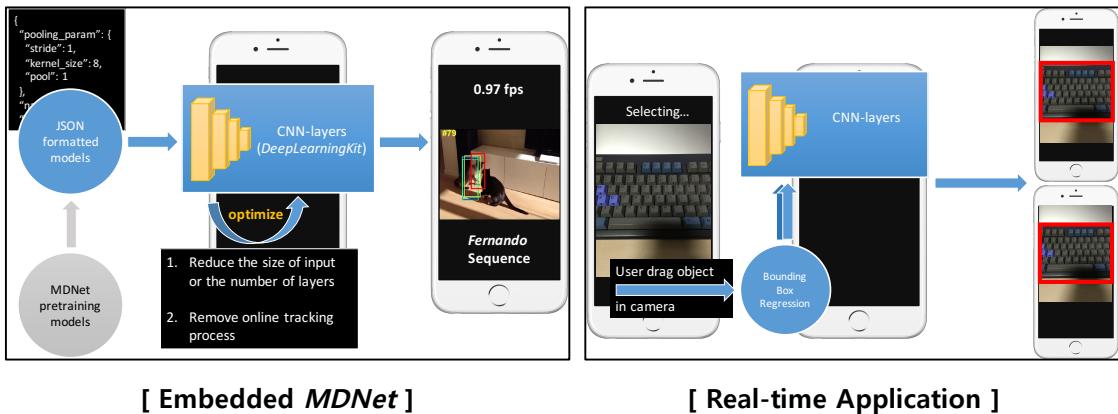
- 1) Abstract

이번 프로젝트의 구현에 사용한 환경은 아래와 같다.

- Environments: OS X El Capitan 10.11, Xcode 7.3

- Frameworks: Metal, OpenCV 2 for iOS, DeepLearningKit
- Languages: Swift, Objective-C

이번 프로젝트는, 1) 우선 MDNet의 알고리즘과 네트워크를 그대로 iOS에 올리고, 2) optimize를 수행하는 순서로 진행되었다. 구현에 있어 여러 어려움이 있었지만 그에 대해서는 후술하고, 결과적으로 구현된 아키텍처를 이미지로 표현하면 아래와 같다.



MDNet의 기존 알고리즘과, optimize가 수행되어 변경된 embedded MDNet의 알고리즘은 아래와 같다.

**Algorithm 1** Online tracking algorithm

```

Input : Pretrained CNN filters { $w_1, \dots, w_5$ }
          Initial target state  $x_1$ 
Output: Estimated target states  $x_t^*$ 
1: Randomly initialize the last layer  $w_6$ .
2: Train a bounding box regression model.
3: Draw positive samples  $S_1^+$  and negative samples  $S_1^-$ .
4: Update { $w_4, w_5, w_6$ } using  $S_1^+$  and  $S_1^-$ .
5:  $\mathcal{T}_s \leftarrow \{1\}$  and  $\mathcal{T}_l \leftarrow \{1\}$ .
6: repeat
7:   Draw target candidate samples  $x_t^i$ .
8:   Find the optimal target state  $x_t^*$  by Eq. (1).
9:   if  $f^+(x_t^*) > 0.5$  then
10:    Draw training samples  $S_t^+$  and  $S_t^-$ .
11:     $\mathcal{T}_s \leftarrow \mathcal{T}_s \cup \{t\}$ ,  $\mathcal{T}_l \leftarrow \mathcal{T}_l \cup \{t\}$ .
12:    if  $|\mathcal{T}_s| > \tau_s$  then  $\mathcal{T}_s \leftarrow \mathcal{T}_s \setminus \{\min_{v \in \mathcal{T}_s} v\}$ .
13:    if  $|\mathcal{T}_l| > \tau_l$  then  $\mathcal{T}_l \leftarrow \mathcal{T}_l \setminus \{\min_{v \in \mathcal{T}_l} v\}$ .
14:    Adjust  $x_t^*$  using bounding box regression.
15:   if  $f^+(x_t^*) < 0.5$  then
16:     Update { $w_4, w_5, w_6$ } using  $S_{v \in \mathcal{T}_s}^+$  and  $S_{v \in \mathcal{T}_s}^-$ .
17:   else if  $t \bmod 10 = 0$  then
18:     Update { $w_4, w_5, w_6$ } using  $S_{v \in \mathcal{T}_l}^+$  and  $S_{v \in \mathcal{T}_l}^-$ .
19: until end of sequence

```

**Algorithm 1** Online tracking algorithm

```

Input : Pretrained CNN filters (Reduce)
          Initial target state  $x_1$ 
Output: Estimated target states  $x_t^*$ 
1: Randomly initialize the last layer  $w_6$ .
2: Train a bounding box regression model.
3: Draw positive samples  $S_1^+$  and negative samples  $S_1^-$ .
4: Update (Reduce) using  $S_1^+$  and  $S_1^-$ ;
5:  $\mathcal{T}_s \leftarrow \{1\}$  and  $\mathcal{T}_l \leftarrow \{1\}$ .
6: repeat
7:   Draw target candidate samples  $x_t^i$ .
8:   Find the optimal target state  $x_t^*$  by Eq. (1).
9:   if  $f^+(x_t^*) > 0.5$  then
10:    Draw training samples  $S_t^+$  and  $S_t^-$ .
11:     $\mathcal{T}_s \leftarrow \mathcal{T}_s \cup \{t\}$ ,  $\mathcal{T}_l \leftarrow \mathcal{T}_l \cup \{t\}$ .
12:    if  $|\mathcal{T}_s| > \tau_s$  then  $\mathcal{T}_s \leftarrow \mathcal{T}_s \setminus \{\min_{v \in \mathcal{T}_s} v\}$ .
13:    if  $|\mathcal{T}_l| > \tau_l$  then  $\mathcal{T}_l \leftarrow \mathcal{T}_l \setminus \{\min_{v \in \mathcal{T}_l} v\}$ .
14:    Adjust  $x_t^*$  using bounding box regression.
15:   if  $f^+(x_t^*) < 0.5$  then
16:     Update { $w_4, w_5, w_6$ } using  $S_{v \in \mathcal{T}_s}^+$  and  $S_{v \in \mathcal{T}_s}^-$ .
17:   else if  $t \bmod 10 = 0$  then
18:     Removed Update { $w_4, w_5, w_6$ } using  $S_{v \in \mathcal{T}_l}^+$  and  $S_{v \in \mathcal{T}_l}^-$ .
19: until end of sequence

```

원래의 알고리즘에서, 10 번째 iteration 마다 수행하는 long-term network update 과정이 optimization 을 위해 최종적으로 제거되었다. 자세한 내용은 후술하겠다.

## 2) Porting

MDNet에서 train을 위한 CNN은 .mat 파일에 정의되어 있었고 (imagenet-vgg-m-conv1-3.mat), 이를 DeepLearningKit의 input으로 제공하기 위해 json 포맷으로 변경해야 했다.

### imagenet\_convert.json

```
"layer": [
  {
    "name": "conv1",
    "pad": 0,
    "type": "Convolution",
    "stride": 2,
    "blobs": [
      {
        "shape": {
          "dim": [
            96,
            3,
            7,
            7
          ]
        },
        "data": [...]
      },
      {
        "shape": {
          "dim": [
            96
          ]
        },
        "data": [...]
      }
    ]
  },
  {
    "name": "pool1",
```

```
        "stride": 2,  
        "pad": [0,0,0,0],  
        "type": "Pooling",  
        "method": "max",  
        "pool": [3,3]  
    },  
    ...
```

위 json 파일을 input 으로 하여 네트워크를 로드한 후, MATLAB 코드 mdnet\_run.m 의 로직을 그대로 수행한다.

#### MDMainViewController.swift, MDSimulationViewController.mm

```
opts = Options()  
deepNetwork = DeepNetwork()  
var caching_mode = false  
  
// JSON 을 통해 CNN 을 로드한다.  
deepNetwork.loadDeepNetworkFromJSON("imagenet_convert", inputImage: nil,  
inputShape: imageShape, caching_mode:caching_mode)  
  
...  
if (opts.bbreg) {  
    posSamples = samples('uniform', location: bb_loc, count: opts.bbreg_nSamples*10,  
    opts: opts)  
    // Uniform distribution 으로 positive sample 을 만들고, bounding box regressor 를  
    train 한다.  
    boundingBox = BoundingBox(pos_samples)  
}  
...  
let targetScore : Int = startTracking() // 재귀적으로 tracker 의 processFrame 을  
호출하는 시작점  
...  
if (opts.bbreg && targetScore > 0) {  
    // Bounding box regression  
    deepNetwork.updateLayer(boundingBox.predict(deepNetwork.conv[0]))  
}  
...  
if (targetScore > 0) {  
    // Positive sample: Gaussian dist
```

```

// Negative sample: Uniform dist
deepNetwork.update(tmpPosSamples, negative: tmpNegSamples)
}

...
// invoke
// main 의 startTracking()에서 호출됨
- (void)invoke:(cv::Mat &)image {
    Mat img_grey;
    cvtColor(image, img_grey, CV_RGB2GRAY); // Change color space

    if (_beginInitialize) {
        if (_tracker != NULL) {
            delete _tracker;
        }
        _tracker->initialize(img_grey, _box);
        _startTracking = YES;
        _beginInitialize = NO;

        NSLog(@"Tracker initialized");
    }

    if (_startTracking) {
        __weak typeof(self) weakSelf = self;
        __block typeof(image) blockImage = image;
        dispatch_async(dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_HIGH,
0), ^{
            if (weakSelf.disabled) {
                return;
            }
            NSLog(@"processing... ");
            NSDate *start = [NSDate date];
            _tracker->processFrame(img_grey); // tracker 는 deepNetwork 를 참조하여
deepNetwork.classify()를 호출한다.

            RotatedRect rect = _tracker->bb_rot;
            Point2f vertices[4];
            rect.points(vertices);
        });
    }
}

```

```

NSTimeInterval end = [[NSDate date] timeIntervalSinceDate:start];
 NSLog(@"%@", end);

    _strong typeof(weakSelf) strongSelf = self;
    dispatch_sync(dispatch_get_main_queue(), ^{
        _timeLabel setText:[NSString stringWithFormat:@"%.3f fps", 1.0f/end]];
        [strongSelf showImage:blockImage];

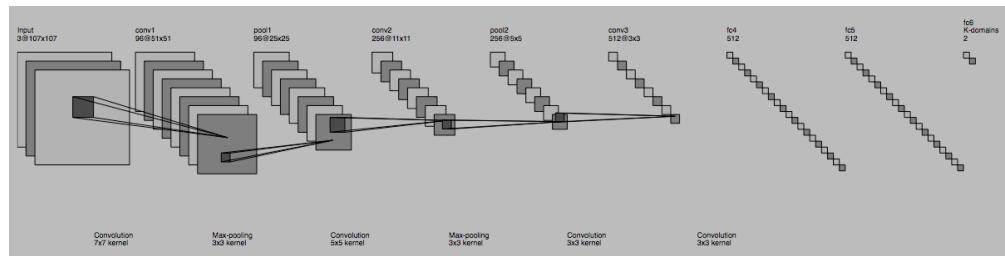
        if (_imageIndex < NUM_IMAGES) {
            // 다음 이미지로 tracking 을 수행한다.
            [_indexLabel setText:[NSString stringWithFormat:@"%ld/%d",
            _imageIndex+1, NUM_IMAGES]];
            UIImage *nextImage = [UIImage imageNamed:[NSString
            stringWithFormat:@"%@", (long)(++_imageIndex+1)]];
            cv::Mat newImage;
            UIImageToMat(nextImage, newImage);
            [strongSelf invoke:newImage];
        }
        else {
            _startTracking = NO;
        }
    });
});
}
}

```

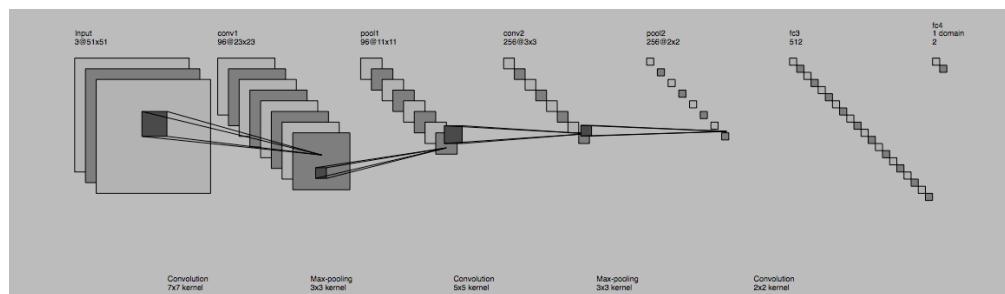
### 3) Optimization

Performance 향상을 위한 optimization 을 trial-and-error 과정을 통해 진행했다. Convolution layer 의 parameter 와 개수 조절은 imagenet\_convert.json 파일 내용을 수정하면서 수행했다.

*MDNet*의 원래의 layer 구조는 아래와 같다.



Embedded *MDNet*에서는, 1) input size 를 107 X 107에서 51 X 51로 줄이고 (51 X 51은 *MDNet*의 conv2 layer의 input size 와 같다), 2) Convolution layer 와 fully-connected layer 를 각각 하나씩 줄여, 최종적으로 아래와 같은 네트워크를 구성했다. (conv1 – conv2 – fc1 – fc2)



#### 4) Real-time Application

Embedded *MDNet*의 network 를 객체로 저장하고, 카메라의 매 frame image 에서, 유저가 화면에서 드래그한 부분의 object 를 tracking 하는 애플리케이션을 구현했다.

##### **MDtrackingViewController.mm**

```
// Configure camera
_videoCamera = [[CvVideoCamera alloc] initWithParentView:self.cameraView];
[_videoCamera setDelegate:self];
[_videoCamera setDefaultAVCaptureDevicePosition:AVCaptureDevicePositionBack];
[_videoCamera setDefaultAVCaptureSessionPreset:AVCaptureSessionPreset1280x720];
[_videoCamera
setDefaultAVCaptureVideoOrientation:AVCaptureVideoOrientationLandscapeLeft];
[_videoCamera setDefaultFPS:30];
...
#pragma mark - Touch
```

```

- (void)touchesBegan:(NSSet<UITouch *> *)touches withEvent:(UIEvent *)event {
    _startTracking = NO;
    _beginInitialize = NO;

    _ltPoint = [[touches anyObject] locationInView:self.cameraView];
    _rbPoint = CGPointMakeZero;
    _selectBox = cv::Rect(_ltPoint.x * _screenRatio, _ltPoint.y * _screenRatio, 0, 0);
}

- (void)touchesMoved:(NSSet<UITouch *> *)touches withEvent:(UIEvent *)event {
    _rbPoint = [[touches anyObject] locationInView:self.cameraView];
}

- (void)touchesEnded:(NSSet<UITouch *> *)touches withEvent:(UIEvent *)event {
    _rbPoint = [[touches anyObject] locationInView:self.cameraView];
    _selectBox.width = abs(_rbPoint.x * _screenRatio - _selectBox.x);
    _selectBox.height = abs(_rbPoint.y * _screenRatio - _selectBox.y);
    _beginInitialize = YES;
    _initBox = _selectBox;
}

...
#pragma mark - Delegate
- (void)processImage:(cv::Mat &)image {
    if (_ltPoint.x > 0 && _ltPoint.y > 0 && _rbPoint.x > _ltPoint.x && _rbPoint.y > _ltPoint.y) {
        if (!_beginInitialize && !_startTracking) {
            rectangle(image,
                      cv::Point(_ltPoint.x * _screenRatio, _ltPoint.y * _screenRatio),
                      cv::Point(_rbPoint.x * _screenRatio, _rbPoint.y * _screenRatio),
                      Scalar(0, 0, 255));
        }
        // 매 프레임마다 image에서 [ltPoint-rbPoint]의 영역에 존재하는 object 를
        tracking 한다.
        [self invokeTracking:image];
    }
}

```

## 4. Technical Issues

이번 프로젝트를 진행하면서 기술적으로 가장 문제가 되었던 부분은

*DeepLearningKit* 자체에 있었다. *DeepLearningKit*은 2015년 12월에 런칭하여, 2016년 2월까지는 활발히 개발되던 framework였다. 하지만 그 후, 과제연구 발표일까지도 더 이상의 업데이트가 없었고, 따라서 CNN의 몇몇 parameter들에 대한 구현이 되어 있지 않은 미완성 상태의 framework를 사용해 프로젝트를 구현해야 했다.



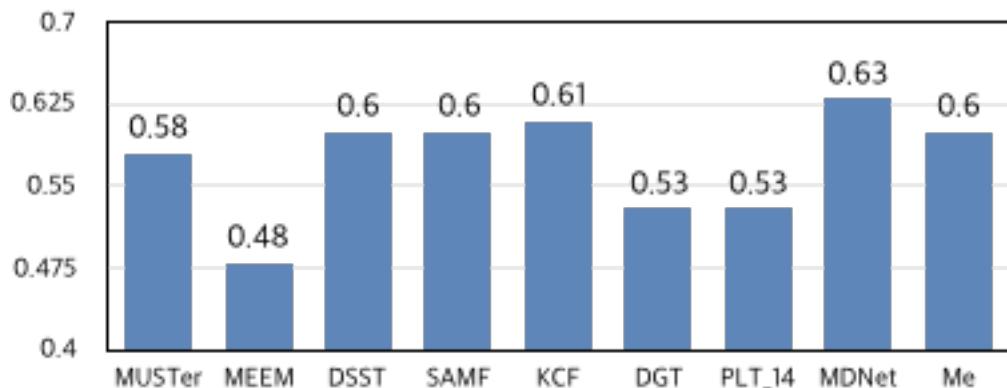
*MDNet*의 porting을 위해 몇몇 parameter는 다른 코드를 참고하여 직접 구현해야 했고, 구현하지 못했던 나머지 parameter-'sync', 'disable\_dropout', 'feat', 'conserve\_memory' 등-는 미지원 상태 그대로 사용해야 했던 점이 이번 프로젝트를 진행함에 있어서 가장 어려운 점이 되었다. 특히 코드를 분석하고 직접 구현에 쏟은 시간이 이번 프로젝트 구현 전체 시간의 많은 부분을 차지했기 때문에, 상대적으로 optimization에 투자할 수 있는 시간이 적어진 부분이 가장 아쉬웠다. (사실 *DeepLearningKit*은 6월 1일에 한 번의 업데이트가 이루어졌지만, 해당 업데이트는 단순히 코드를 Swift 3에 compatible하도록 바꾼 내용이기 때문에 위의 issue는 여전히 해결되지 않았다.)

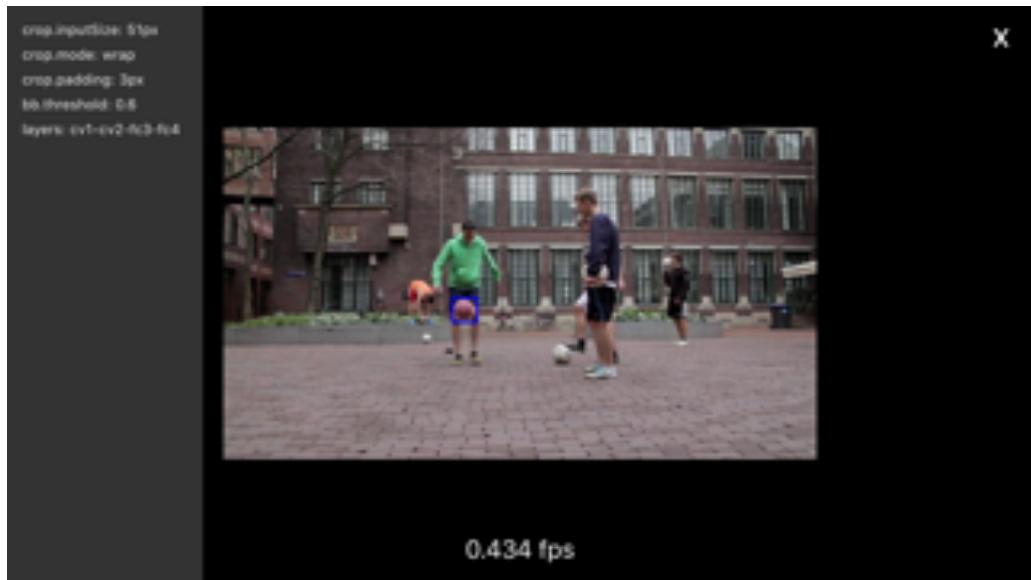
## 5. Results

*MDNet* 프로젝트에 포함되어 있는 Dataset 인 VOT2015.ball1 sequence 로 Embedded *MDNet* 의 accuracy 와 작동 시간을 측정하였다 (논문에서는 함께 측정했던 Region\_noise 는 측정하지 못했다). Accuracy 의 유도식은 다음과 같다.

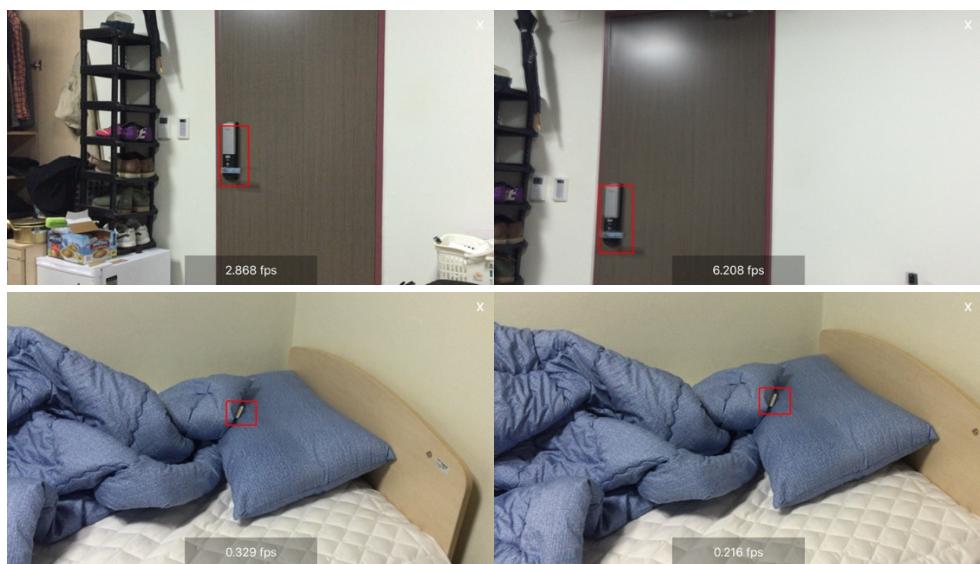
$$(Accuracy) = \frac{(Area \text{ of the ground truth box} \wedge Area \text{ of tracked box})}{(Area \text{ of the ground truth box})}$$

결과적으로, Embedded *MDNet* 은 평균 0.4 fps 정도의 performance 를 보여주었고, ball1 sequence 에 대한 accuracy 는 평균 0.60 으로 측정되었다. 이는 논문에서 언급한 다른 state of the art 알고리즘과 비교해 높은 수치이며, 기존의 *MDNet*의 accuracy 인 0.63 보다 4.8% 정도 하락한 수치이다. 물론 앞서 언급했듯이, Embedded MDNet 의 accuracy 는 오직 ball1 sequence 로만 측정한 결과이므로 그 비교의 의미가 크게 중요하지 않을 수 있다.





Real-time Application 의 경우, ground truth 를 추출할 수 없기 때문에 따로 accuracy 를 측정하지는 않았다. Real-time Application 에 심어진 Embedded *MDNet* 네트워크는 기본적으로 작동하는 fps 가 매우 낮기 때문에, 특히 빠르게 움직이는 물체의 경우 tracking 을 제대로 수행하지 못하는 경우를 자주 보여주었다. Heuristic 하게 판단할 수 있는 부분도 있었는데, 육안으로 봤을 때 background 와 object 의 구분이 잘 되는 상황에서는 상당히 빠른, 최대 7 fps 정도로 real-time tracking 을 수행하는 모습을 보여주는 경우가 있었고, 반대의 경우에는 tracking 에 성공하더라도 최저 0.15 정도의 매우 낮은 fps 를 보여주었다.



## 6. Discussion and Future Research

미완성 framework 인 *DeepLearningKit*을 사용했고, 상대적으로 optimization 에 투자한 시간이 부족했던 문제가 있었지만, Visual Tracking 알고리즘을 개발할 때 *MDNet*의 접근과 같이 적은 layer 개수의 CNN 을 사용하는 방식이 좋은 결과를 낼 수 있다는 사실을 확인할 수 있었다. 원래의 알고리즘으로부터 fully-connected layer 를 모두 update 하는 long-term network update 과정을 제거했음에도, (단 하나의 sequence 에서만 테스트를 진행했다는 한계가 있지만) accuracy 가 4.8%로 매우 적게 하락한 점을 볼 때, mobile device 와 같이 하드웨어적 한계가 존재하는 플랫폼에서는 long-term update 과정을 제거한 CNN network 로 속도와 정확도를 모두 고려할 수 있다는 결론을 내릴 수 있다. 더해 Embedded *MDNet* 이 올라간 카메라 애플리케이션에서, Real-time visual tracking 이 작동하는 것을 확인할 수 있었고, 그렇기에 추후 상용화 가능한 단계까지 performance 를 끌어올릴 수 있다면 관련 기술을 이용한 상업적인 서비스를 개발하는 것도 가능하다고 결론 지을 수 있다.

개인적으로는 첫 딥러닝 프로젝트였는데, 결과물이 왜 좋게 나오는지, 혹은 왜 나쁘게 나오는지 정확히 알 수 없다는 점이 프로젝트를 진행하면서 가장 특이한 부분이었다. 예컨대, 최종적인 optimization 의 결과로 4 개의 layer 로 이루어진 network 가 나왔지만, 그 전에 fully-connected layer 하나를 더 붙여 5 개의 layer 로 이루어진 network 로 tracking 을 수행하면 결과가 안 좋은 것을 확인할 수 있었다. 왜 fully-connected layer 가 2 개일 때에는 결과가 더 좋은 것이고, 3 개일 때에는 안 좋은 것인지를 알 수가 없었고, 이게 딥러닝의 특징일지도 모른다는 생각을 하게 되었다.

만일 *DeepLearningKit* 이 CNN 의 (Caffe 기반의) 모든 parameter 를 지원하게 된다면, 더 좋은 연구를 할 수 있을 것이다. iOS 에서 CNN 을

지원하는 torch7-ios 와 같은 몇몇 대안이 존재하는 것이 사실이지만, 그 대안들은 모두 GPU 가속을 지원하지 않는다는 점에서 *MDNet*과 같은 deep network 기반의 알고리즘은 사실상 제대로 된 작동이 힘들 것이다. 비록 4 개월 넘게 한 번의 업데이트도 이뤄지지 않았지만, 최근에 한 번 업데이트가 이뤄진 것을 보면 아직 버려진 프로젝트는 아니라고 생각한다. 완성에 가까워진 *DeepLearningKit*를 사용해 다시 한 번 *MDNet*을 iPhone 에 porting 하고 optimization 을 진행하면 더 좋은 결과를 낼 수 있을 것이다. 또한 고성능 Android device 의 경우 평균적으로 iPhone 보다 하드웨어 성능이 더 좋으니, Android 에서도 이 과제연구와 비슷한 porting 및 optimization 을 진행하면 좋은 결과를 낼 수도 있을 것이다.

## 7. References

- [1] Y. LeCun, "Backpropagation Applied to Handwritten Zip Code Recognition," *Neural Computation*, 1989.
- [2] H. Nam, "Learning multi-domain convolutional neural networks for visual tracking," *arXiv:1510.07945*, 2015.
- [3] Y. Wu, "Object tracking benchmark," *TPAMI*, 2015.
- [4] S. Chen, "Convolutional Neural Network and Convex Optimization," 2015, Retrieved from  
<http://acsweb.ucsd.edu/~yuw176/report/ECE273.pdf>
- [5] VOT2015, <http://www.votchallenge.net/vot2015>
- [6] *DeepLearningKit*, <http://deeplearningkit.org>
- [7] *MDNet*, <https://github.com/HyeonseobNam/MDNet>
- [8] torch7-ios, <https://github.com/clementfarabet/torch-ios>