Deformable Convolution Networks

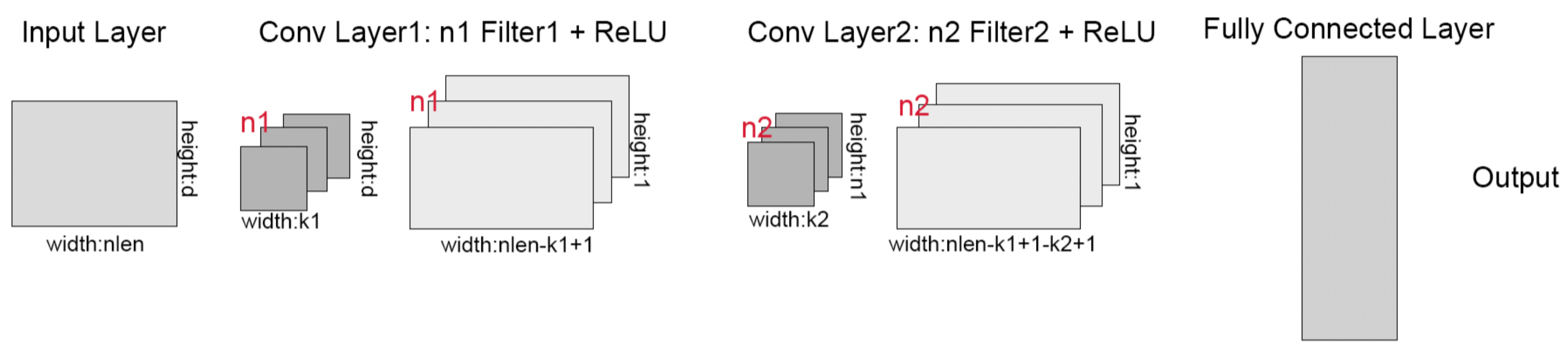
Rui Qu, Sayyed Ali Kiaian Mousavy

Drawback of standard CNN

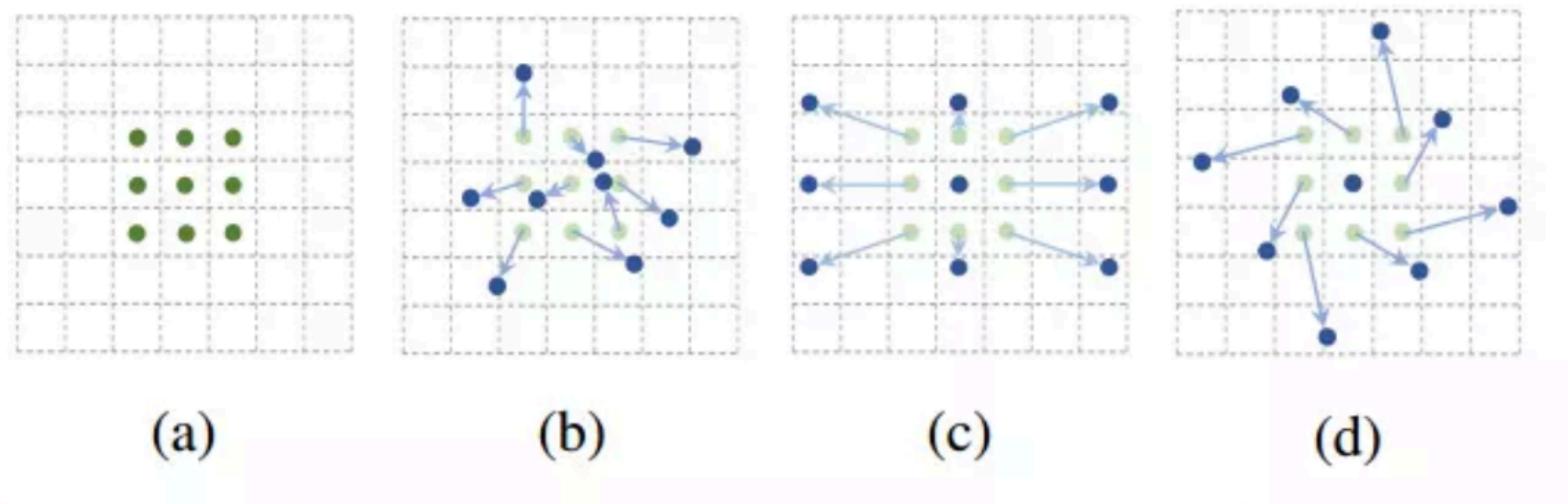
CNN is limited to model geometric transformation due to fixed kernels, pre-defined pooling mechanism for dealing with variations in the spatial arrangement of data.

Not robust, e.g. CNN cannot identify an image if deformation, scaling, cropping, rotations…

To improve the level of CNN's generalization, we take the idea of Deformable Convolution Networks by Jifeng Dai, 2017



Kernels

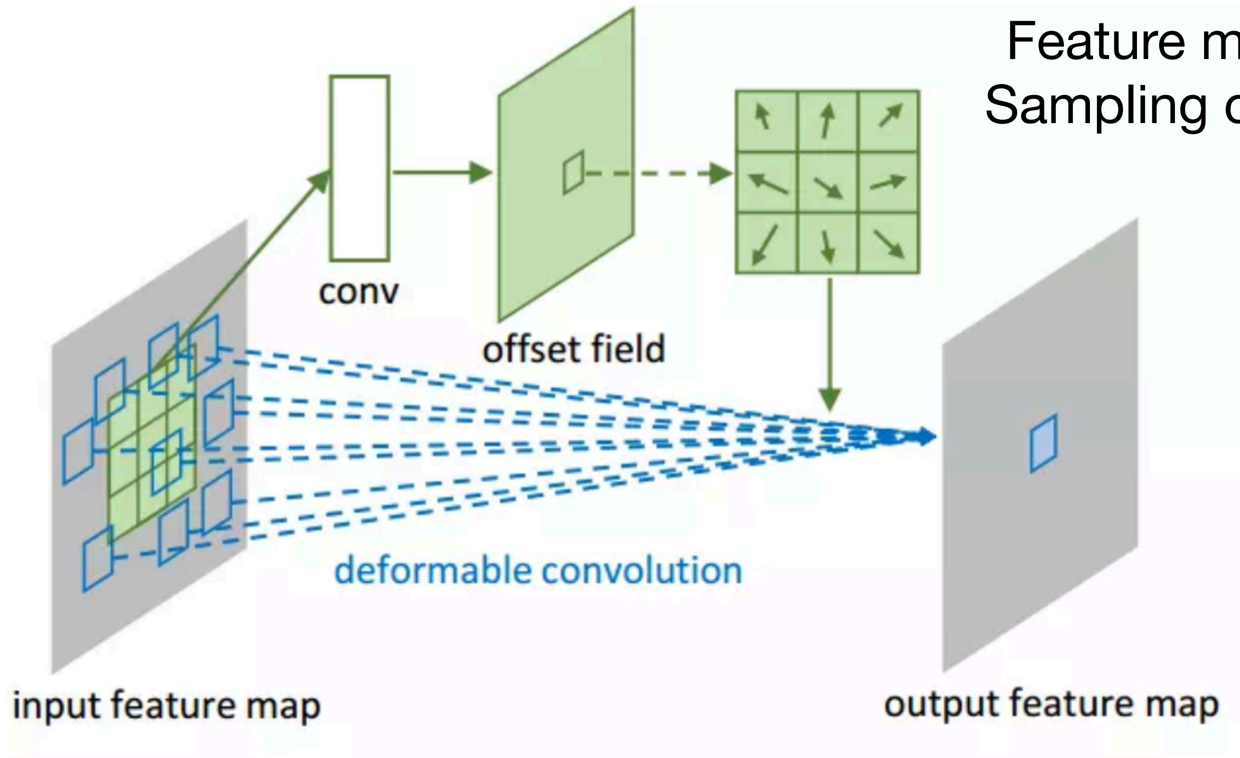


(a) Regular sampling grid with 3x3 fixed kernel
(b) Deformable sampling grid with augmented offsets
(c) Deformable sampling grid with dilation
(d) Deformable sampling grid with rotation

Key idea: Integrate pixels in the input image before convolution

Deformable Convolution

Feature map is augmented with offsets
Sampling coord:original sampling+offset



$$y(p_0) = \sum_{p_n \in R} w(p_n)\ x(p_o + p_n + \delta p_n)$$

$$\delta p_n \ is\ 2D, use\ \delta p_n\ to\ denote\ \delta p_n^x\ \delta p_n^y$$

δp_n is added to every point in the grid.

Divide convolution into 2 directions:

1. Learnable offsets $\delta p_n,\ h \times w \times 2n, n\ denotes\ number\ of\ pixels,\ 2n\ denotes\ x\ \&\ y\ directions$

With offsets, green window is changed into irregular blue window

2. Get all integrated pixels into a new image as input of next layer. The rest is same as CNN

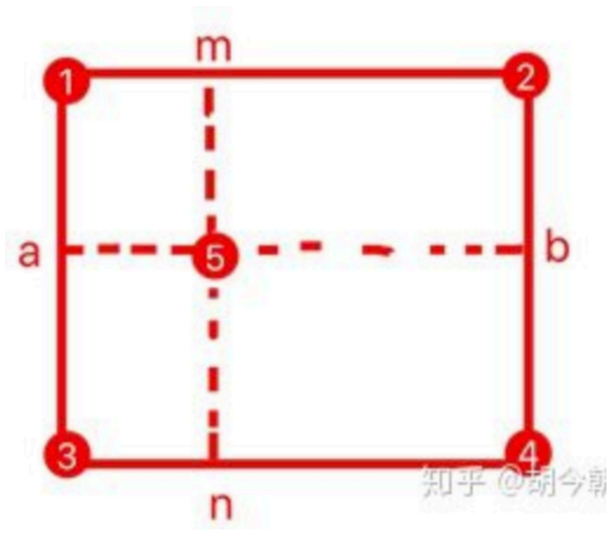
NB δp_n are mostly fractional instead of integers, Bilinear Interpolation to get p, Where p is original point, q is new point. G is Bilinear Interpolation kernel

$$x(p) = \sum_{p} G(q,p)\ x(q)$$

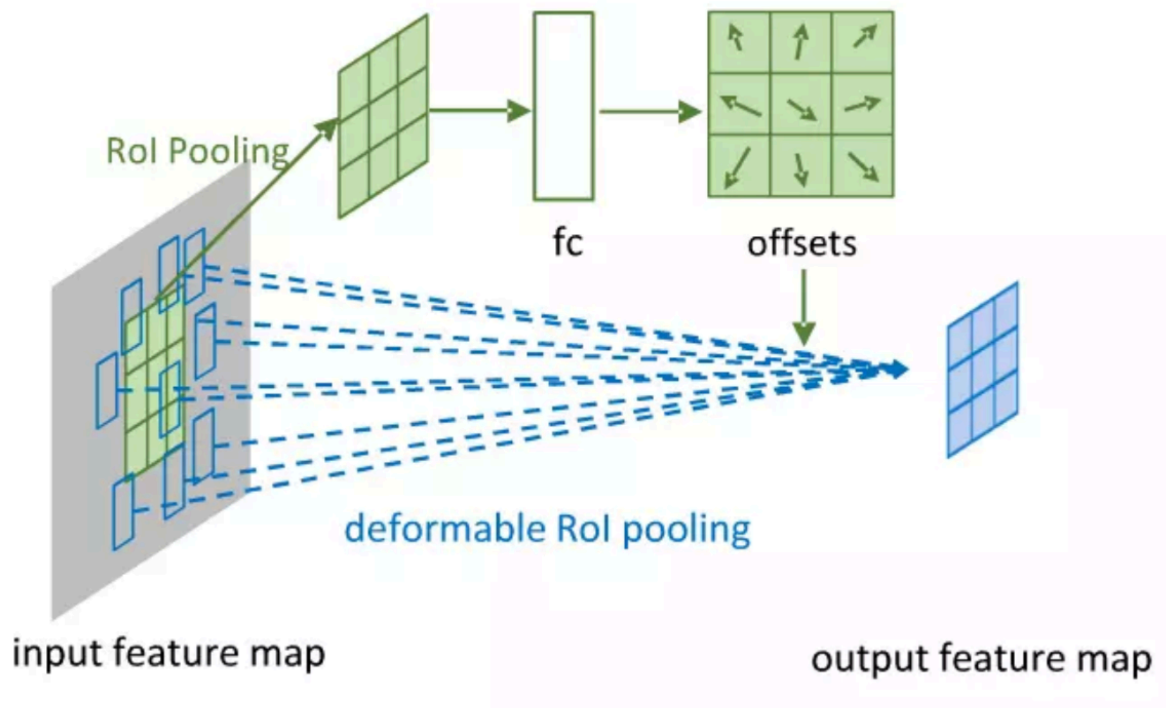
$$G(q,p) = g(q_x, p_x) g(q_y, p_y),\ g(q,p) = max(0, 1 - |q - p|)$$

In the integration of image pixels, the pixel needs to be offset, and the generation of the offset leads to a floating point type. The offset must be converted to integer, or the back propagation cannot be performed.

e.g. For a coordinate (a, b), we get floor(a), ceil(a), floor(b), ceil(b) and integrate into (floor(a),floor(b)), ((floor(a),ceil(b)), ((ceil(a),floor(b)), ((ceil(a),ceil(b)) these 4 coordinates represent pixels in input image



Deformable ROI Pooling



$$y(i,j) = \sum_{p \in bin(i,j)} x(p_0 + p_n + \delta p_{i,j})/n_{i,j}$$

δp_{ij} is added to the binning position indexes.

With RoI Pooling, input feature map is shaped into k*k. With FC, it gets a k*k offsets

Offsets normalization:

It's necessary to make offset learning invariant to RoI size

$$\delta p_{ij} = \gamma \; \delta \hat{p}_{ij} o(w,h), \; \gamma = 0.1$$

Back-propagation

1. Deformable Convolution, gradient w.r.t the offset δp_n

$$y(p_0) = \sum_{p_n \in R} w(p_n)\ x(p_o + p_n + \delta p_n)$$

$$\frac{\partial y(p_0)}{\partial \delta p_n} = \sum_{p_n \in R} w(p_n) \cdot \frac{\partial x(p_0 + p_n + \delta p_n)}{\partial \delta p_n}$$

$$= \sum_{p_n \in R} [w(p_n) \cdot \sum_{q} \frac{\partial G(q, p_0 + p_n + \delta p_n)}{\partial \delta p_n} x(q)]$$

2. Deformable RoI Pooling, gradient w.r.t the offset δp_{ij}

$$y(i, j) = \sum_{p \in bin(i,j)} x(p_0 + p_n + \delta p_{i,j}) / n_{i,j}$$

$$\frac{\partial y(i, j)}{\partial \delta p_{ij}} = \frac{1}{n_{ij}} \sum_{p \in bini,j} \frac{\partial x(p_o + p_n + \delta p_{ij})}{\partial \delta p_{ij}}$$

$$= \frac{1}{n_{ij}} \sum_{p \in bini,j} [\sum_{q} \frac{\partial G(q, p_0 + p_n + \delta p_{ij})}{\partial \delta p_{ij}} x(q)]$$

Experiment 1:

1. Environment with: Numpy=1.14.5, MXNet=1.14.0 Pytorch=0.4.1

2. Downloaded and compiled core official deformable convolution written in MXNet

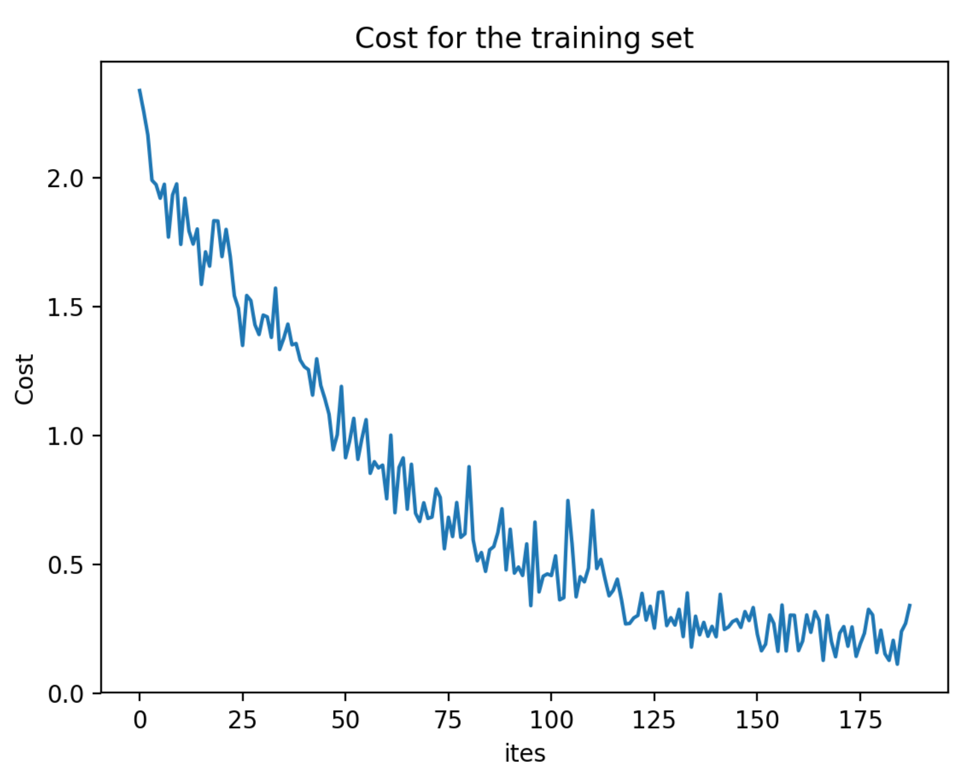
3. Wrote a new deformable convolution module from the scratch based on Pytorch

4. Wrote an initial test to compare its output with reference MXNet implementation

Structure of network

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 32, 28, 28]             320
       BatchNorm2d-2           [-1, 32, 28, 28]              64
            Conv2d-3           [-1, 64, 28, 28]          18,496
       BatchNorm2d-4           [-1, 64, 28, 28]             128
            Conv2d-5          [-1, 128, 28, 28]          73,856
       BatchNorm2d-6          [-1, 128, 28, 28]             256
            Conv2d-7           [-1, 18, 28, 28]          20,754
         ZeroPad2d-8          [-1, 128, 30, 30]               0
            Conv2d-9          [-1, 128, 28, 28]         147,456
      DeformConv2D-10         [-1, 128, 28, 28]               0
       BatchNorm2d-11         [-1, 128, 28, 28]             256
            Linear-12                  [-1, 10]           1,290
================================================================
Total params: 262,876
Trainable params: 262,876
Non-trainable params: 0
----------------------------------------------------------------
```

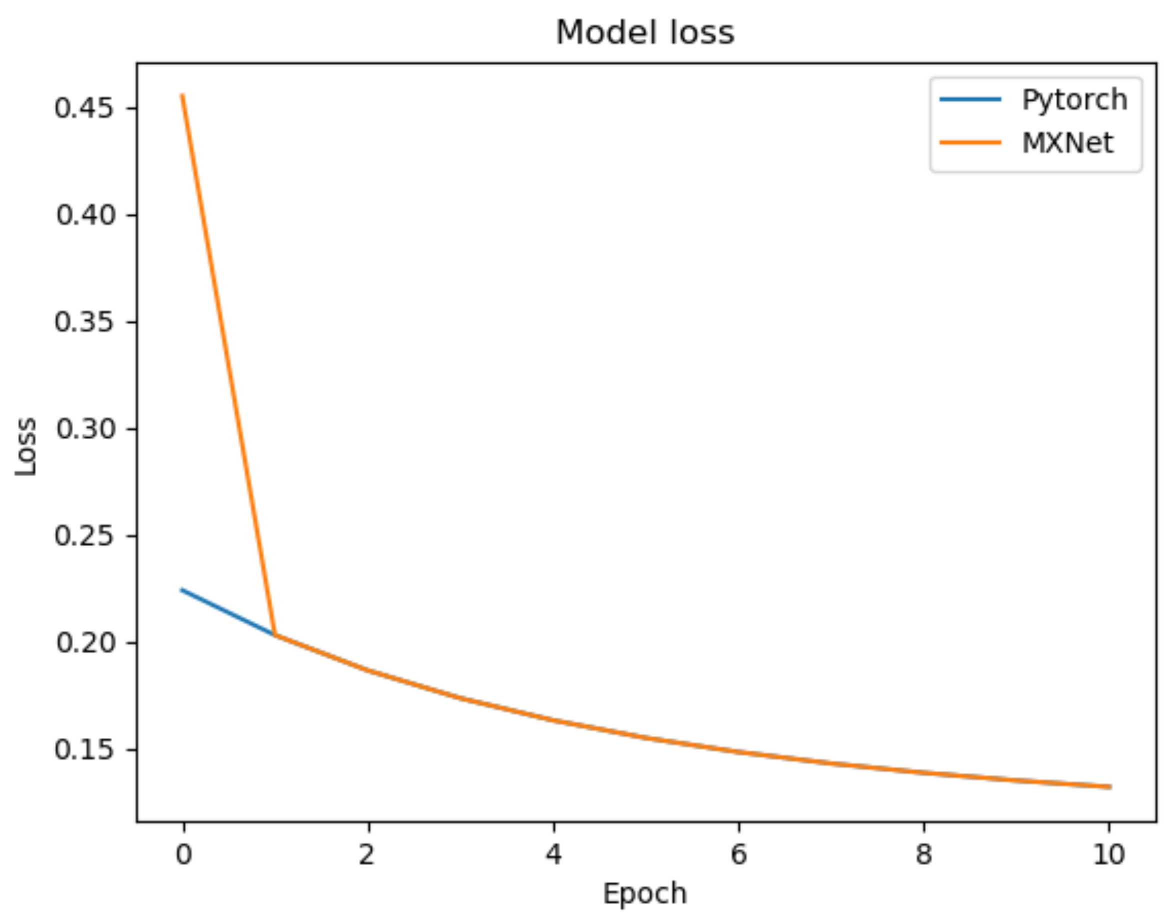
Cost plot



This is the program output(Figure is attached) also added loss to mxnet on iter 1 on purpose to see the merger:

```
Using gpu0
Initial prediction for pytorch:
tensor([[[[0.4386, 0.5196, 1.1148, 0.9907, 1.0584],
          [0.6117, 0.6075, 0.8192, 0.1481, 0.6742],
          [0.5517, 0.2466, 0.5748, 0.9452, 0.8228],
          [0.4648, 0.1409, 0.3565, 0.2767, 0.8366]]]],
          device='cuda:0', grad_fn=<CudnnConvolutionBackward>)


Pytorch output:
tensor([[[[0.3330, 0.3140, 0.8156, 0.6836, 0.6901],
          [0.4699, 0.4132, 0.5052, 0.0176, 0.4540],
          [0.3275, 0.1768, 0.3375, 0.7252, 0.5669],
          [0.4083, 0.1153, 0.2613, 0.1984, 0.5843]]]],
         device='cuda:0', grad_fn=<CudnnConvolutionBackward>)


Initial prediction for MXNet:

[[[[0.43863603 0.5196189  1.1148171  0.99070585 1.0584273 ]
   [0.6117178  0.6074905  0.819201   0.14805761 0.67420554]
   [0.55170524 0.24655274 0.5747814  0.9451818  0.82283825]
   [0.46481684 0.1409446  0.35653543 0.27666456 0.83661854]]]]
<NDArray 1x1x4x5 @gpu(0)>


Reference MXNet output:

[[[[0.3330357  0.3139518  0.8155961  0.683637   0.69011986]
   [0.46991453 0.41321972 0.5052018  0.01761584 0.4540226 ]
   [0.32745734 0.17681491 0.3375253  0.7252236  0.56690156]
   [0.4083497  0.11533365 0.26126528 0.1983664  0.5842552 ]]]]
<NDArray 1x1x4x5 @gpu(0)>
```

Model loss Pytorch vs MXNet



Experiment 2

1.Forked and modified torch vision’s Resnet generator to have an additional parameter.

2. Set that parameter toTrue passes model’s input through additional 3 layers of deformable convolution in last block layer.

3. Also did the same to torch vision’s VGG generator.

Experiments3 Deep-lab(edge detection):

1. Forked and modified a pytorch deeplabv3 implementation from https://github.com/jfzhang95/pytorch-deeplab-xception

2. Modified to swap out it's ad-hoc resnet with our resnet generator

3. Added another parameter to our modified resnet generator

4. Set that parameter to false excludes the classification layer(required for image encoder/decoder like deeplab)

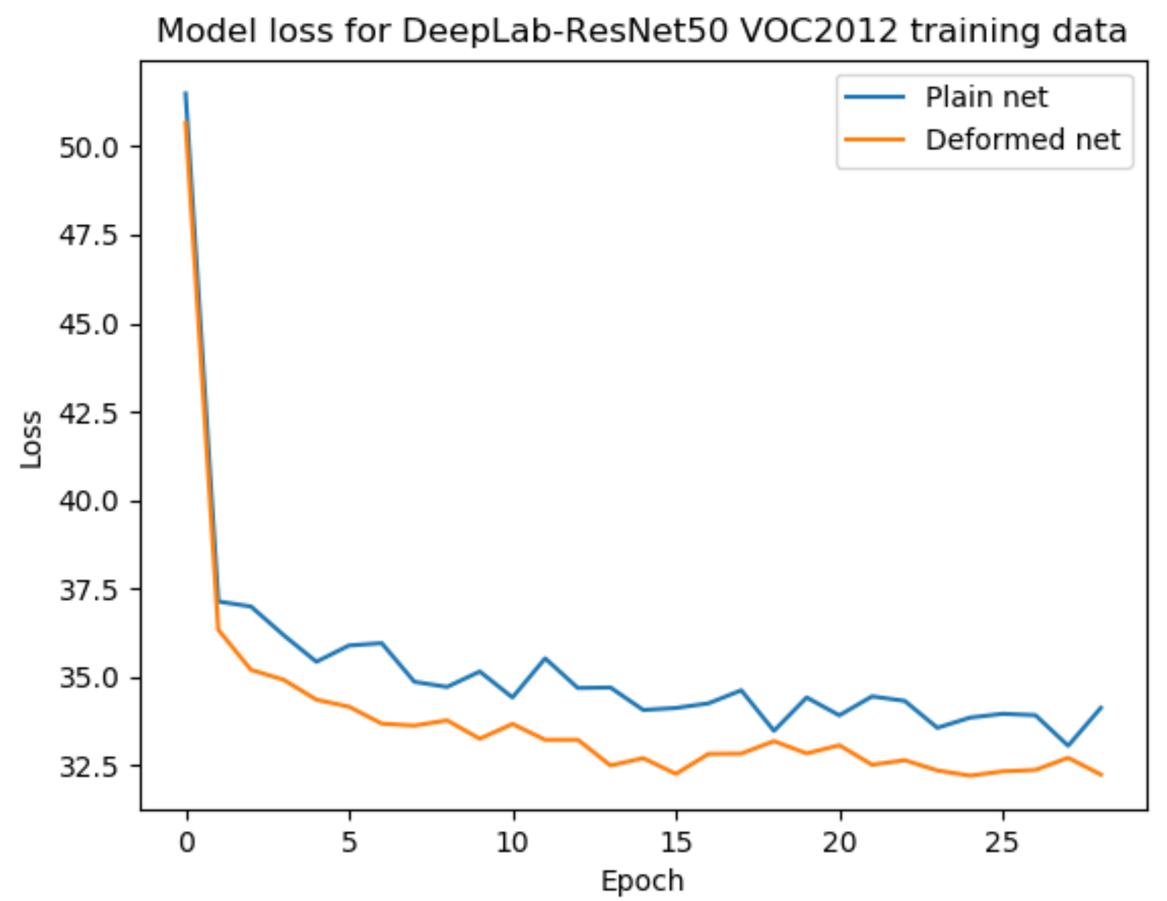
5. Trained both deformed and non deformed version for 30epochs using VOC 2012 dataset.

Deeplab-resnet50 With deformable convolution:

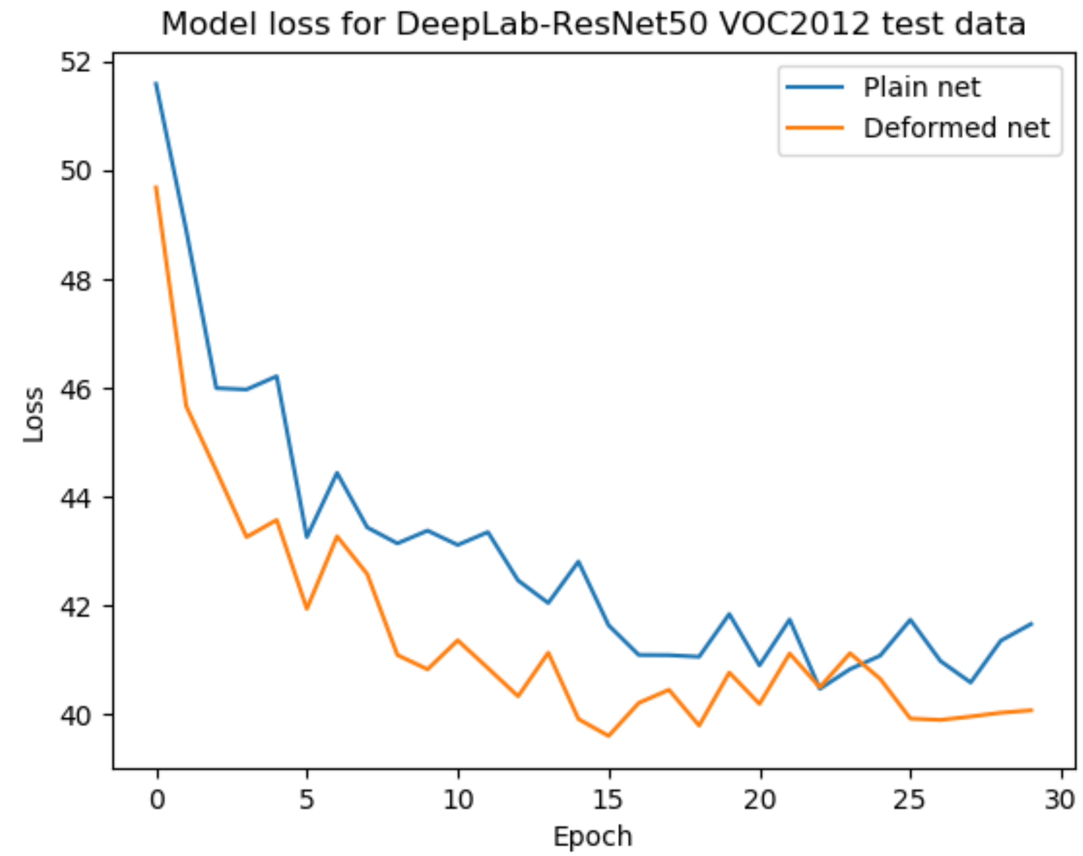
Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 64, 257, 257]	9,408
BatchNorm2d-2	[-1, 64, 257, 257]	128
ReLU-3	[-1, 64, 257, 257]	0
MaxPool2d-4	[-1, 64, 129, 129]	0
Conv2d-5	[-1, 64, 129, 129]	4,096
…		
BatchNorm2d-148	[-1, 2048, 17, 17]	4,096
Conv2d-149	[-1, 18, 17, 17]	331,776
ZeroPad2d-150	[-1, 2048, 19, 19]	0
Conv2d-151	[-1, 2048, 17, 17]	37,748,736
DeformConv2D-152	[-1, 2048, 17, 17]	0
Conv2d-153	[-1, 2048, 17, 17]	2,097,152
BatchNorm2d-154	[-1, 2048, 17, 17]	4,096
ReLU-155	[-1, 2048, 17, 17]	0
…		
ReLU-220	[-1, 256, 129, 129]	0
Dropout-221	[-1, 256, 129, 129]	0
Conv2d-222	[-1, 21, 129, 129]	5,397
Decoder-223	[-1, 21, 129, 129]	0

Total params: 154,593,717
Trainable params: 154,593,717
Non-trainable params: 0

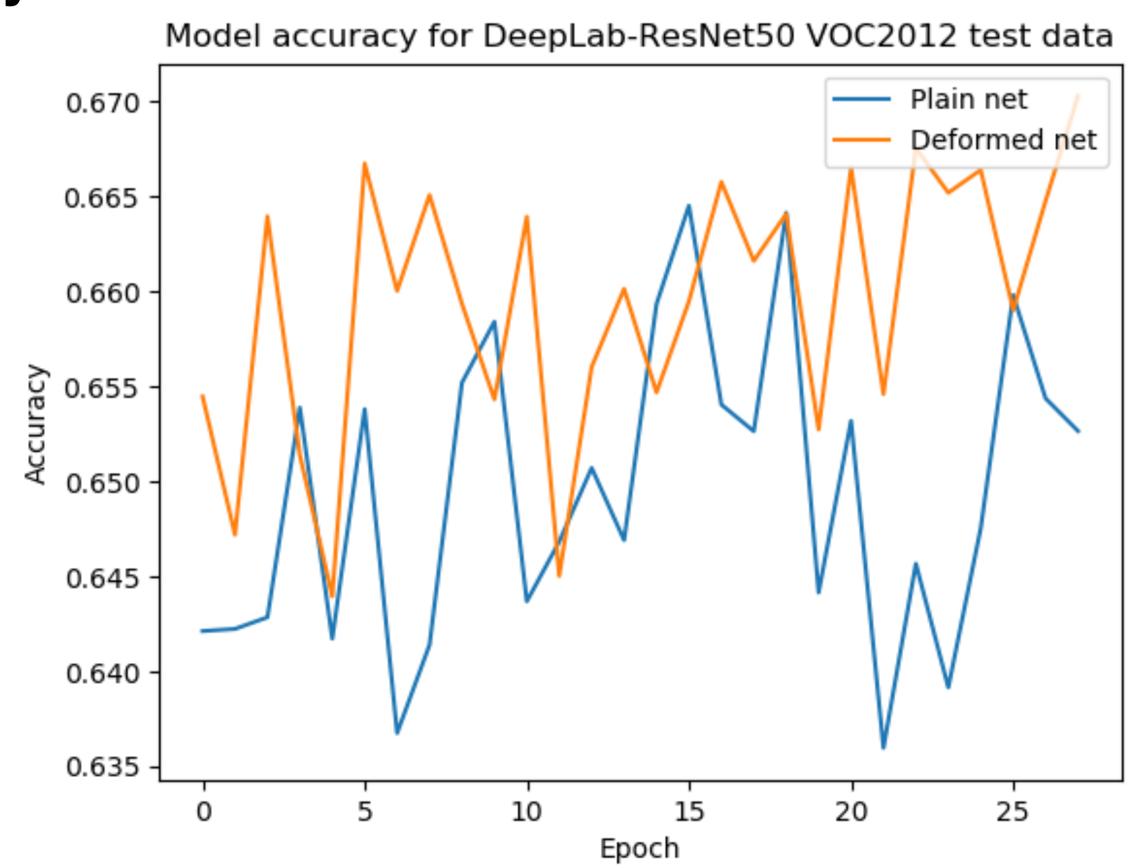
Training Loss：

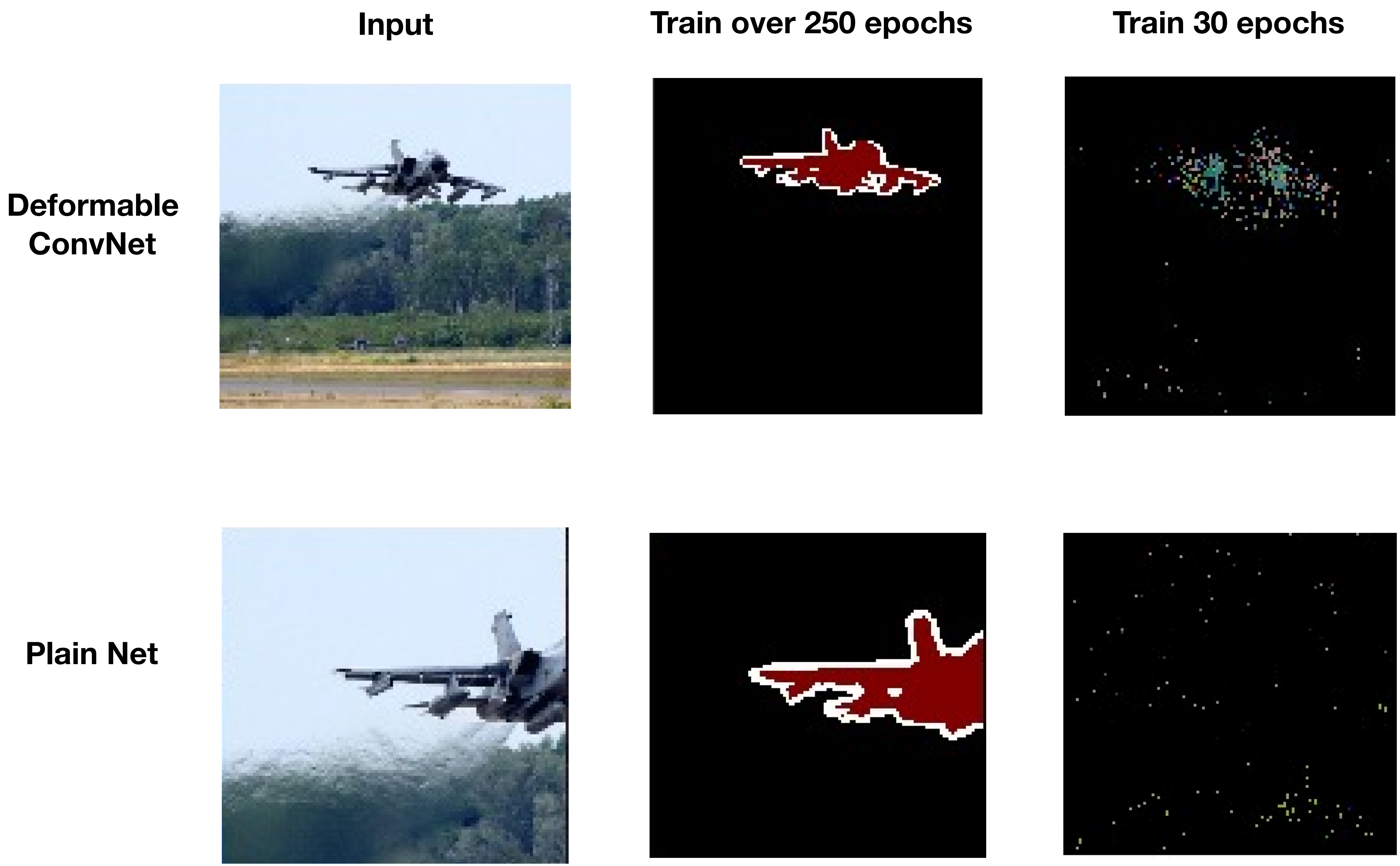


Test Loss:



Test Accuracy:




Input
Train over 250 epochs
Train 30 epochs
Deformable ConvNet
Plain Net

Experiments4 Siamese RPN(motion detection):

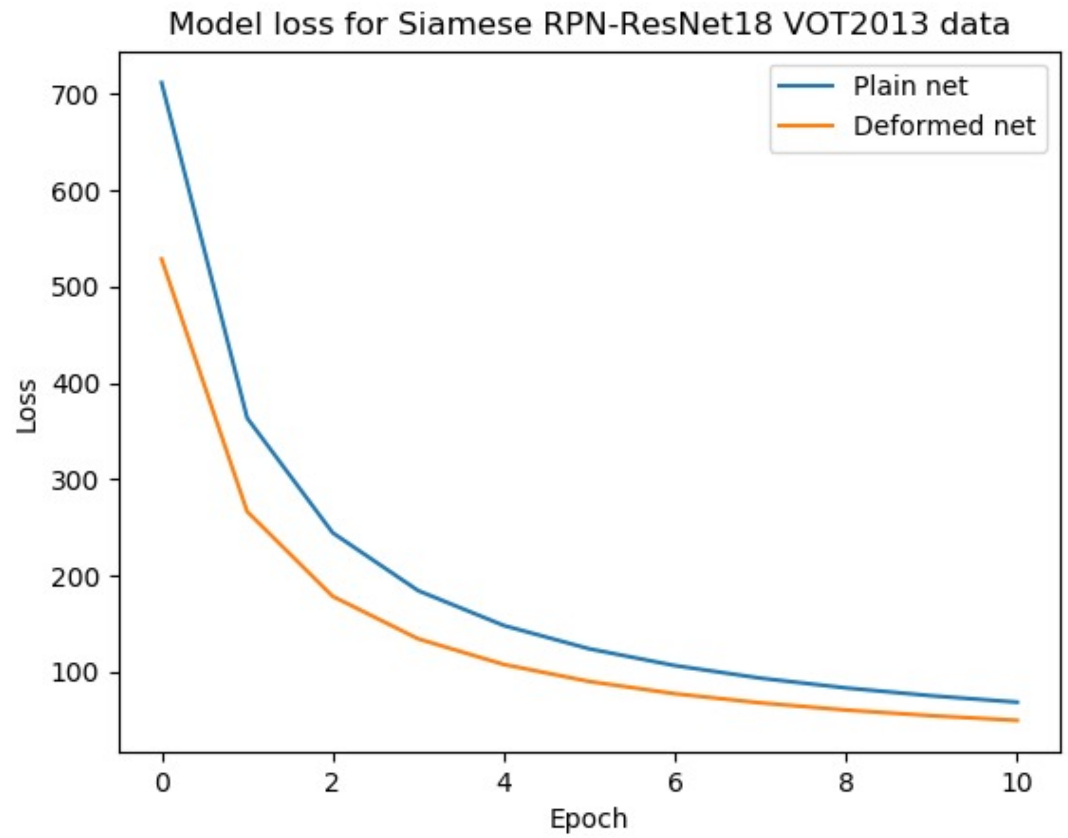
1. Forked and modified a pytorch Siamese RPN implementation from https://github.com/songdejia/Siamese-RPN-pytorch

2. Implementation was very buggy, modified it to make it work.

3. Swapped out it’s ad hoc feature extractor with our resnet generator small(18 and 34).

4. Trained both deformed and non deformed version for 10 epochs using VOT 2013 dataset.

Model Loss：

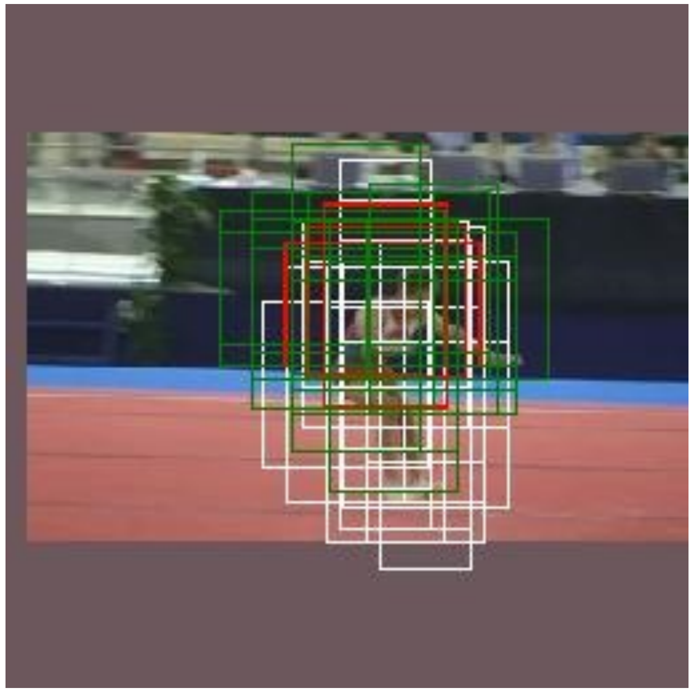


Model loss for Siamese RPN-ResNet18 VOT2013 data
Plain net
Deformed net
Loss
700
600
500
400
300
200
100
0
2
4
6
8
10
Epoch

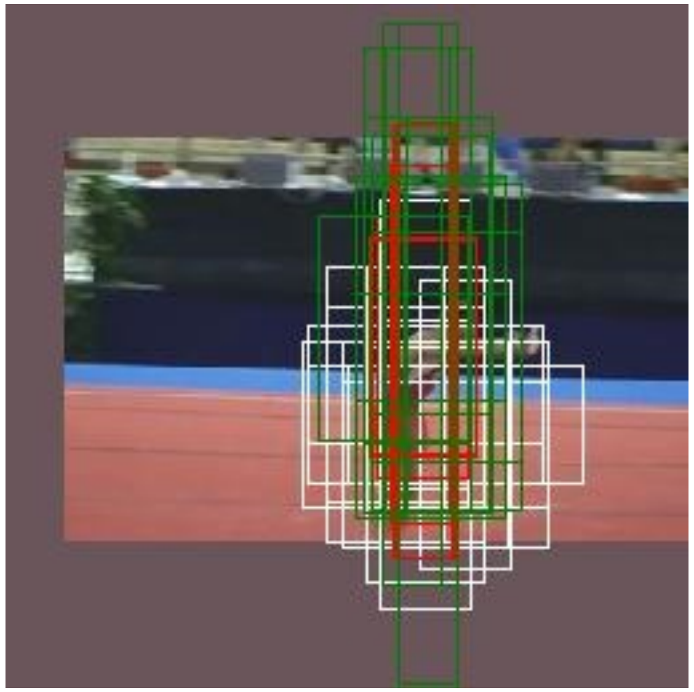
RPN Results

Original image

Plain net



Deform net



Future work on R-CNN(detecting objects of interests):

Fork and modify a pytorchfaster RCNN implementation from https://github.com/jwyang/faster
rcnn.pytorch/tree/pytorch-1.0

Modify to swap out it's ad-hoc resnet with our new resnet generator.

Swap out RoI pooling

Train both deformed and non deformed version using VOC 2012 dataset.