Project Report for: Deformable Convolutional Networks

Sayyed Ali Kiaian Mousavy **Rui Qu**

D2424 Deep Learning Course Group 17

Abstract

We implemented a Deformable Convolution as described in Dai. et. al. paper from the scratch using pytorch backend. After the implementation of vanilla version according to section 2, we evaluated our using the VOC2012 dataset on a pair of different popular existing neural networks, namely Deeplab and RPN(Region Proposal Network). We also have measured the accuracy of our network against the reference implementation in MXNet.

1 Introduction

A key challenge in visual recognition is how to accommodate geometric variations or model geometric transformations in object scale, pose, viewpoint, and part deformation. And, regular Convolutional neural networks are limited to model geometric transformations due to the fixed geometric structures in their building modules.

The basic idea of this project is to improve a Convolutional neural network using geometeric deforming convolutions. First, to detect the main region of interest in images, a Special deformed sampling is invoked. As a result, our network begins to detect our object of interest, as well as it's possible transformation in different images. An object of interest could have been scaled or rotated, or some combination of both.

In order to apply a deformable convolution, a N to 2N channel convolution is applied. Then, it uses the new convolved channels as a grid for applying bilinear (interpolative) sampling to the original neural network prior to it's convolution.

Bilinear sampling is rather a popular method used for software up/downsampling and skewing of images. And, learning such deformations performed by software could be rather useful for improved learning, especially since these deformed samples are easy to generate via software. Therefore, the performance of the neural network can be improved without the need to obtain any additional samples.

Bilinear sampling works by applying bilinear interpolation of data based on an input and a grid, where the grid is 2 matrices with same dimensions as the input in 2D case . Each entry in 2 matrices represent the x and y(one for x, and the other for y) of new point inside our input grid, where it is passed as same index in new output data. By adding index of original input to the grid, the new sampled points will be relevant to their original input points, which is accordant to what is called "offset" in the original article.

Dai et. al. proposed that deformable convolution and deformable pooling layers are fully compatible with current convolutional networks, as they can be trained using currently available back-propagation algorithms. Therefore, they can be considered drop-in replacements for their plain convolutional neurons counterpart and can be readily deployed as convolutional neural network layers without any additional consideration.

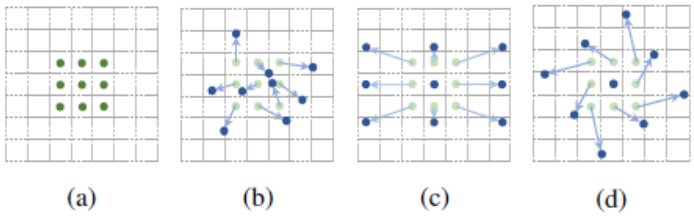


Figure 1: Illustration of the sampling locations with 3×3 kernel in standard and deformable convolutions. (a) is regular convolution while (b), (c) and (d) are deformable ones. In deformable convolution, the grid is transform to compensate for image transformation

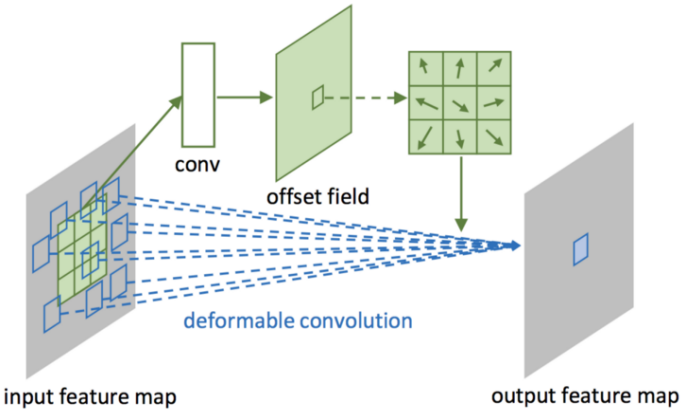


Figure 2: Illustration of deformable convolution with 3×3 kernel

2 Background

Convolutional neural networks are inherently limited in modeling transformations. The limitation originates from the fixed geometric structures of their modules: a convolution unit samples the input at fixed locations; Then, a pooling layer reduces the spatial resolution at a fixed ratio. In other words, Internal mechanisms to handle the geometric transformations are lacking.

2.1 Deformable Convolutional Networks

Deformable Convolutional Networks is an attempt to make a network invariant to image transformation by applying convolution layers that resample the input feature map. This enhances the convolutional neural network's ability to recognize geometerically transformed images and objects of interests.

2.2 Deformable Convolution

The 2D deformable convolution consists of two steps: 1) sampling using a regular grid R over the input feature map x; 2) summation of sampled values weighted by w(Like a regular convolution).

A regular convolution looks like:

$$y(p_0) = \sum_{p_n \in \mathbf{R}} w(p_N) \cdot x(p_0 + p_n), \tag{1}$$

where p_n enumerates the locations in $\mathbf{R}$.

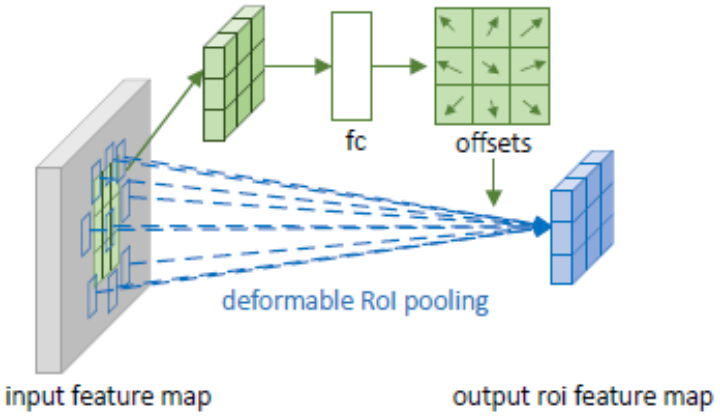


Figure 3: Illustration of deformable pooling with 3×3 kernel

In deformable convolution, the grid $\mathbf{R}$ is augmented with offsets:

$$y(p_0) = \sum_{p_n \in \mathbf{R}} w(p_N) \cdot x(p_0 + p_n + \Delta p_n), \tag{2}$$

where Δp_n is the offset that is added to every point in the grid.

Since our deltas are mostly fractional instead of integers, the new grid must be calculated via bilinear interpolation:

$$x(p) = \sum_q G(q, p) \cdot x(q), \tag{3}$$

where q is our original point and p is our new point that must be interpolated. Here, G is our bilinear interpolation kernel.

2.3 Deformable Pooling

Pooling is used in all region proposal based object detection methods. It converts an input rectangular region of arbitrary size into fixed size features.

In pooling, feature map is divided into bins of $k \times k$ bins and outputs a $k \times k$ feature map, where k is a free parameter. For (i, j)-th bin we have:

$$y(i, j) = \sum_{p \in bin(i,j)} x(p_0 + p_n)/n_{ij}, \tag{4}$$

where n_{ij} is our pixel index in the bin.

Similar to Eq. 1, Δp_{ij} is added to the binning position indexes. As a result, Eq. 4 becomes:

$$y(i, j) = \sum_{p \in bin(i,j)} x(p_0 + p_n + \Delta p_{ij})/n_{ij}, \tag{5}$$

Where Δp_{ij} is calculated with Eq. 3, much similar to a deformable convolution.

3 Our Approach

3.1 Implementation

We implemented this project using pytorch, which is one of the recommended deep learning frameworks for python purposed by the lecturer of this course.

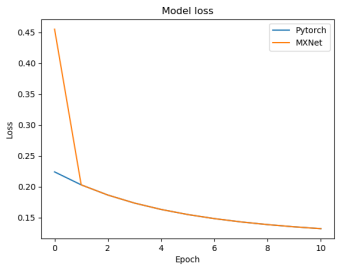


Figure 4: Illustration of Comparison of reference MXNet loss function vs. our Pytorch version in a similarly designed simple network trained over MNIST images

Dai. et. al. have also managed to implement their own article (Link to project). But, their implementation is based on another framework named MXNet. They also have developed their deformable convolution layer in native C++ for performance reasons. And then wrapped it in Cython.

For the reasons stated above, We did the core part of the re-implementation of this project from scratch. Although, the original code by Dai. et. al. did influence our implementation to some extent, due to the similar objectives they are aiming to achieve.

We implemented the deformable convolution offset and deformable pooling offset as seperate pytorch layers. As a result, modularity and compatibility with existing pytorch based convolutional neural networks was achieved just like mentioned in paper.

3.2 Deforming layer for convolution

We created a new layer class that extended the regular convolution layer class. But, instead of returning the convolution, it performed a N to 2N channel convolution based on it's input channel size and then performed a bilinear sampling on original input feature map using the convoluted channels as the grid. Index of each point x and y was also added to the grid before interpolation in order to make bilinear sampling relevant to offsets.

3.3 Deforming layer for pooling

For this part, we created a new layer class that extended the FC layer class. The FC layer was replicated to make a proper grid for bilinear sampling. Similar to deforming layer for convolution, index of each point was added to the grid before interpolation in order to make bilinear sampling relevant to offsets.

3.4 Bilinear interpolation

We basically implementated our bilinear interpolation with pytorch (reference implementation in stackoverflow) as opposed to reference implementation by Dai et. al. . In reference implementation, new layers were designed as full layers in C++ Cuda with MXNET wrappers for performance reasons. Therefore, our implementation was about 2.5 times slower as a result due to inefficient matrix multiplication in bilinear interpolation by pytorch.

3.5 Initial tests

In Dai et. al. , gains of newly added layers were marginal and were achieved using very large networks and training epochs. So, in order to test whether our implementation works, we've designated an alternative approach.

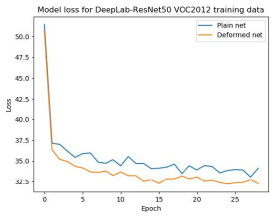
We reasoned that a deformable convolution must be the same in both reference implementation in MXNet and our implementation in pytorch. Therefore, we trained two simple four layer convolutional network with MNIST images, one of which had the MXNet deformable network, and the other one had the pytorch one.

Deeplab-resnet50 With deformable convolution:

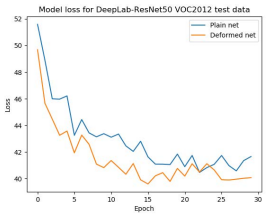
Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 64, 257, 257]	9,408
BatchNorm2d-2	[-1, 64, 257, 257]	128
ReLU-3	[-1, 64, 257, 257]	0
MaxPool2d-4	[-1, 64, 129, 129]	0
Conv2d-5	[-1, 64, 129, 129]	4,096
...		
BatchNorm2d-148	[-1, 2048, 17, 17]	4,096
Conv2d-149	[-1, 18, 17, 17]	331,776
ZeroPad2d-150	[-1, 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

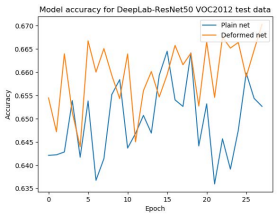
Figure 5: Our deeplab structure in PyTorch



(a) Training loss for Deeplab ResNet-50



(b) Test loss for Deeplab ResNet-50



(c) Test accuracy for Deeplab ResNet-50

Figure 6: 3 Figures showing training and test loss and test accuracy over 30 epochs. The train accuracy is unchanged while difference in test accuracy with zoomed in images becomes further noticeable

We trained both of these networks over 10 epochs using gradient descend. The results was that our pytorch implementation outputted the exact same gradients as the reference MXNet implementation by Dai et. al.

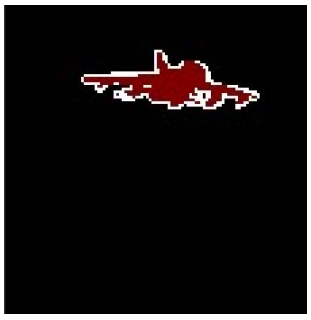
4 Experiments

We first started our experiment with the following specifications:

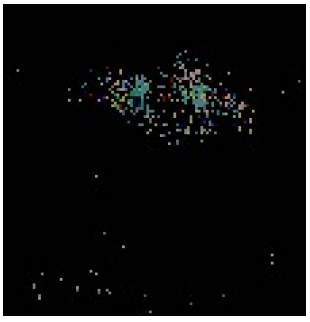
Software: Python 3.6, GCC 5, Pytorch 0.4.1, MXNet 1.14.0.

Hardware: Two VMs in Google Cloud Platform, both having single Tesla V100, 16GB of RAM and 4 VCPU

(a) Original Image



(b) Ground Truth



(c) Detection results

Figure 7: Example detection using deformable Deeplab-ResNet-50

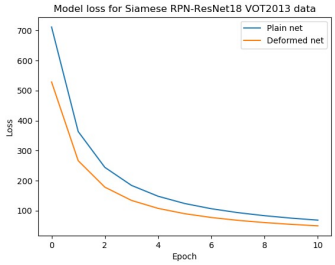
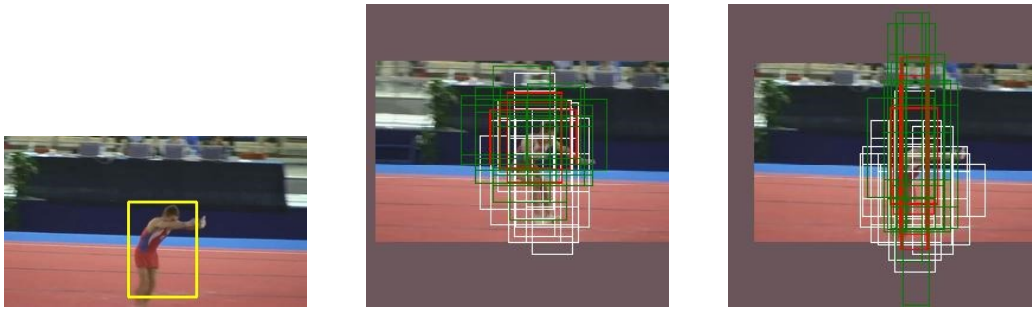


Figure 8: Illustration of Model training loss for both plain RPN network(blue) vs. deformable RPN network(oranage) for 10 epochs



(a) Ground Truth

(b) Detection results for plain network

(c) Detection results for deformable network

Figure 9: Example detection using Siamese RPN-ResNet-18. White and green boxes are positive and negative indices. They are then added together to produce the final, red box detection results.

We've simply replicated tests performed by Dai et. al. In Dai et. al. , deformable convolution was applied and tested four popular state of the art convolutional neural networks. Namely, DeepLab, RPN(Region proposal network), R-CNN and R-FCN. In Dai et. al., Resnet-101 feature extraction network was used as a basis and VOC 2012 was used as training data. Also, VOC 2007 was used as tesing data. In the experiment, deformable layers were applied to 3 last layers for each CNN.

Since we did not have enough resources to do our trainings via an actual ResNet-101, we used ResNet-50 for our feature extraction network instead. Also, we did our experiments only on DeepLab and RPN. And for RPN, we did not use a full feature RPN and used a lighter Siamese RPN instead. Although, we did use VOC2012 as our training data and VOC2007 as our test data just like what was represented by Dai. et. al. for DeepLab. But, for our RPN, we used VOT2013 instead.

First we started implementaing a modified ResNet generator from Torchvision's ResNet generator, where we added 3 layers of offset deformation in it's last block layer(one per each convolutional block in last layer).

Then, we've forked an existing DeepLab implementation. We swapped out it's ad hoc feature extraction ResNet with our ResNet generator. We also have further modified our ResNet generator to become a feature extraction network instead of a classification one. After that, we trained two instances of DeepLab on two VMs in Google Cloud Platform. One used the ResNet generator generating plain ResNet-50 DeepLab and another one generating Deformable ResNet-50 DeepLab. We only trained them for 30 Epochs as opposed to 250 epochs presented by Dai. et. al.

Results of this experiment show that the deformable network is 3 percent more accurate and it's loss function numbers are also almost 3 percent smaller.

Then, we forked an existing Siamese RPN implementation and modified it's feature extraction to use our ResNet generator ResNet-18. We used ResNet-18 from our generator since the network was too small to carry a ResNet-50. We also trained two instances of RPN in two VMs a similar way that we did to DeepLab, one carrying the plain feature extraction ResNet-18 and another one carrying

deformable one. We have trained those two instances for only 10 Epochs, since we reasoned that this must be enough training for training a small RPN. Also, we used our entire data for training in this network, and used the transformed images of the same training dataset for validation. We also have used VOT2013 dataset instead of VOC2012.

Results of this experiment shows that the deformable network initially had less loss points, but two networks eventually merged regarding loss function points.

Our results overall showed that the deformable version of our network were able to learn quicker when counting training steps. Although, deformable version of our network was trained 30 percent slower. Also, our deformable version managed to detect transformed images in test dataset more accurately comparing to plain version. This result is more present in DeepLab resulting images.

For RPN, plain network was able to get the dimension of ground truth better, but the bounding box is a bit off when the image is transformed. Deformable version of network however, had rather incorrect dimensions for bounding box. But, it had managed to detect the overall area of ground truth better than the plain version in transformed images.

5 Related works

Various attempts have been made to address this issue in convolutional neural networks. Among them were training the network with more software generated transformed images, or, adding new convolutional layers that make the network invairant to transformations present in images.

5.1 Scale invariant feature transform(SIFT)

The scale-invariant feature transform (SIFT) is a feature detection algorithm in computer vision to detect and describe local features in images. It was patented in Canada by the University of British Columbia and published by David Lowe in 1999.

In SIFT, keypoints of objects are first extracted from a set of reference images and stored in a database. An object is recognized in a new image by individually comparing each feature from the new image to this database and finding candidate matching features based on Euclidean distance of their feature vectors.

This implementation is one of the inspirations for deformable convolutional neural network development. And the thoughts and considerations are somewhat similar to DCNN. But, SIFT is rather not applied to a CNN and was instead only applied a hand crafted parameter to a regular image. Also, it cannot handle unknown transformations in the new tasks.

5.2 Spatial Transformer Networks (STN)

Spatial Transformer Networks is another CNN improvement which results in models which learn invariance to translation, scale, and rotation, similar to a deformable convolutional network.

Just like a deformable convolutional network, bilinear sampling of data is also used. But, it had applied the feature in an exhaustive feature warping in order to learn a dataset and it's different variations and transformations via a global parametric transformation. Therefore, it's computation cost is much more than a deformable convolution. In other words, deformable convolutional network can be considered a lighter version of spatial transformer network.

6 Conclusions

Our experiments shown that the results gained from applying deformable convolution is rather marginal. However, researches shown that the gains are similar in deeper networks and the cost of the deformation becomes less of a burden in a deeper network since one deformed convolution consists of 2 convolutions and 1 bilinear resampling which almost costs as much as 3 convolutions combined but the cost is less significant considering that we have some networks with more than a hundreds of convolution layers.

The experiment also shown that the difference between a normal network and a deformable network is more significant for unknown and unpredicted transformations compared to known and trained ones. When we also train our normal network with different transformations, the normal network also becomes more resilient against those predicted transformations. But, it still remains vulnerable to unpredicted ones.

7 Future work

At the current state, it was not possible to train RoI pooling as well as ResNet 101 version of our networks due to time and resources limit. We've figured out that deformable pytorch models are inherently bigger, and, 16 GB of GPU memory was not enough for the deformable ResNet 101 DeepLab and RPN. Therefore, we've decided to limit our experiments to only two sets and with ResNet 50 instead.

Possible future work could include deploying the deformable version of our ResNet 101 model to multiple GPUs in order to combat memory limits, as well as coming up with a more efficient implementation of RoI pooling for faster R-CNN. We could also plan to run our experiments for longer periods and for more predictable test suites in order to come up with more accurate results.

References

[1] Dai, J., Qi, H., Xiong, Y., Li, Y., Zhang, G., Hu, H., & Wei, Y. (2017). Deformable convolutional networks. In *Proceedings of the IEEE international conference on computer vision* (pp. 764-773).

[2] Chen, L. C., Papandreou, G., Kokkinos, I., Murphy, K., & Yuille, A. L. (2017). Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE transactions on pattern analysis and machine intelligence, 40(4)*, 834-848.

[3] Dai, J., Li, Y., He, K., & Sun, J. (2016). R-fcn: Object detection via region-based fully convolutional networks. In *Advances in neural information processing systems* (pp. 379-387).

[4] Jaderberg, M., Simonyan, K., & Zisserman, A. (2015). Spatial transformer networks. In *Advances in neural information processing systems* (pp. 2017-2025).

[5] Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems* (pp. 91-99).

[6] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems* (pp. 1097-1105).

[7] Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *International journal of computer vision, 60(2)*, 91-110.

[8] Parker, J. A., Kenyon, R. V., & Troxel, D. E. (1983). Comparison of interpolating methods for image resampling. *IEEE Transactions on medical imaging, 2(1)*, 31-39.

Appendix: acquired knowledge in deep learning per each student

In this project we (as a team) learned about the development of development of neural network with pytorch as well as MXNet.

7.1 Sayyed Ali Kiaian Mousavy

The first skill that I've gained during the project in deep learning is various frameworks in which deep learning is developed, as well as their advantages and disadvantages. I've initially started developing this project in pytorch, only to realize midway that pytorch cannot preserve enough speed/precision after each interpolation calculation. This was mainly due to the limitation of how pytorch is implemented in CUDA 9. Although, it did fare better in google cloud due to the better implementation in CUDA 10

One of the interesting things I've saw here is that these two frameworks had different mindsets when it came to development. Where, in my opinion, pytorch waws more numpy user friendly while tensorflow is more c++ programmer friendly.

I've also learned about some popular CNN architectures. Namely VGG16 and ResNet. ResNet simply consists of a CNN generator, where the number in ResNet corresponds to the amount of loop in CNN generator + 2 layers for first and last layer. Different experimental analysis has shown that deeper networks can classify images more accurately. But, this relationship diminishes with more number of layers.

I've also learned about various different training algorithms and how they work, e.g region based object detectors like DeepLab and RPN(Region Proposal Network). They are both convolutional neural networks that detect regions of interest(objects) via special convolutions. These networks have a decoder layer that represents their results as another image, which is the same input image with resulting detection marks. The difference between these networks were their layers used for detecting, their resulting decoded images, as well as the ground truth used for each network. But, both of them used same backbone feature extraction networks that enabled us to swap it our with our own deformable one.

Lastly, I've learned about the convolutional neural networks. In terms of neural networks, we provide a complex neural connectivity where neighboring neurons affect each other more than neuron that are far apart. Informally, these networks basically densen feature changes in close proximity via a sliding window kernel. Different types of convolutional neural networks have been developed to further utilize this ability for feature/object extraction, object detection and even object/image synthesis.

Convolutional neural networks require little effort and can usually learn the features that were used to be hand crafted in regular machine learning based algorithms. On the other hand, modifying and crafting a specific CNN's behaviour/feature has proven to be difficult and this is usually the area that improvements are being made in.

An offtopic thing I've learned here is how to do proper research and how important it is in a research to test every step in progress, even those steps that are supposedly marginal. I've also learned how to read and relate to different scientific paper as well as making a connection between materials in order to provide a precise and concrete scientific implementation and conclusion.

7.2 Rui Qu

The first and most important skill I learnt in the course Deep Learning is that to code the neural networks from scratch, which means only import numpy library to implement the classical DL model like multi-layer Nerural Networks, CNN, RNN. It's quite essential for me to understand the mathematical foundations behind and practice programming skills of Python, especially when I played with those hyper-parameters to get better performances. I did all 4 assignments with Python.

Secondly, I learned a lot by reading papers. For example, there're some drawbacks of CNN like it maintains fixed geometric structure and cannot detect geometric transformations. The paper Deformable CNN clarified a significant improvement on CNN and make it easy to utilize in other demos. Thus we chose it as our final project. It's a step-by-step procedure. I did it on the basis of successfully implementing assignment3 option2 to be familiar with CNN. Then I followed the official tutorials of Tensorflow and Pytorch, which are more efficient and effective than programming from scratch. The first Tensorflow code I programmed was to classify MNIST dataset, then I transformed it to classify Cifar10 and got over 70% accuracy which is far better than I got in assignment1 and 2. With some experience of Tensorflow and Pytorch I began to surf in Deformable CNN.

One drawback of my work is that I don't have one GPU device and I have to train networks either on cloud platform which is not intuitive or on CPU which is too slow. But this course opens the gate of deep learning to me. I became enthusiastic to do the most state of art research and gave me inspirations on how to write a technical paper.