# Deeply Cascaded Networks

**Eunbyung Park**
Department of Computer Science
University of North Carolina at Chapel Hill
eunbyung@cs.unc.edu

## 1 Introduction

After the seminal work of Viola-Jones[15] fast object detection algorithm, cascade classifier frameworks have been dominant in many of computer vision applications, especially for the ones that real time processing is necessary( e.g. pedestrian detection for autonomous cars and face detection for mobile devices). The cascade framework consists of many of weak and strong features and classifiers top of them, and they are ordered by their performance (or computational cost). Most of image patches are filtered out by weak classifiers at early stages of the cascade and only a few of them survive throughout the cascade. That being said, it can significantly reduce computational cost by removing the necessity of strong classifier evaluations on every image patches.

More advanced features have been adopted in this framework to improve the performance in the past decade. More recently, deep convolutional neural network(CNN) has been prevalent method because of its overwhelming discriminative power and it started to emerge as a strong classifier at late stages in the cascade at the expense of its expensive computation[1, 10, 3]. Common approaches are either combination with traditional hand-crafted features for early stage classifiers[1, 3] or presence of smaller and shallower networks followed by larger and deeper networks[10].

Although late stage deep CNNs are responsible for a few of patches that passed through the cascade, their computational complexity are still concerned. Therefore, restricted size of CNNs were encouraged in current approaches, e.g. smaller size of filters, thinner layers and shallower networks[1, 10, 3]. However, it is widely accepted view that deeper and larger networks outperform shallower and smaller networks[7, 13, 14, 5]. Thus, it is necessary to adopt deeper and larger networks in order to build more accurate system. In this work, we propose deeply cascaded networks that allows us to deploy deeper and larger CNNs in the cascade framework with much lower computational complexity.

We construct the cascade inside a deep CNN rather than using multiple CNNs to construct the cascade[1]. Each layer of the network has its own classifier and each classifier at each layer works as a stage of the cascade. In other words, an image patch is examined by each layer classifier to see if it should pass to the next layer. For example, binary classification of image patches, such as pedestrian detection, the only patches that passed all the layers are considered as positive examples. And, many of patches that failed to reach the top of the network will be considered as false examples. So, computational complexity will be significantly reduced if we can successfully reject many of negative examples in lower layers of the network.

In addition, the proposed approach has more efficient feature extraction process. Each layer features that are used by each layer classifiers are extracted based on previous layers. Unlike other approaches that the network should extract the features from the scratch, features for each layers are shared across the layers. Hence, only one convolutional layer operations are required to extract upper level features no matter how many layers are used in higher layer classifiers.

[1] Recently, similar idea was presented in [16].

Finally, we propose to use the soft cascade objective[2, 12] that enable us to jointly train all the classifiers together in the network. Not only it provided a convenient way of training process compared to hard cascade training, but also it gave an additional performance.

## 2 Deeply Cascaded CNN

### 2.1 Soft Cascade Training

We assume typical binary classifier on the top of patches produced by existing object proposal methods. Given a training set of positive and negative examples $D = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$, where $\mathbf{x}_i \in \mathbb{R}^d$ is an image patch and $y_i \in \{0, 1\}$ is binary label. In soft cascade, an image patch will be positive if all classifiers in the cascade predict it as positive. Thus, the probability for an image patch to be positive will be

$$p(y_i = 1|\mathbf{x}_i, \mathbf{w}) = \prod_{t=1}^{T} p_t(y_i = 1|\mathbf{x}_i, \mathbf{w})$$

, where $T$ is the number of stage in the casecade and $\mathbf{w}$ is the model parameter. We define our loss function as negative log likelihood. Given the assumption that image patches are independent and identically distributed, the loss function over the training set $D$ can be written as

$$\begin{aligned} L(D; \mathbf{w}) &= -\log \prod_{i=1}^{N} \prod_{t=1}^{T} p_t(y_i = 1|\mathbf{x}_i, \mathbf{w}) \\ &= -\sum_{i=1}^{N} \sum_{t=1}^{T} \log p_t(y_i = 1|\mathbf{x}_i, \mathbf{w}) = -\sum_{t=1}^{T} \sum_{i=1}^{N} \log p_t(y_i = 1|\mathbf{x}_i, \mathbf{w}) \\ &= \sum_{t=1}^{T} L_t(D; \mathbf{w}) \end{aligned}$$

Therefore, we can easily construct the overall cascade loss function as sum of each stage loss functions. We minimize this loss function with traditional stochastic gradient descent method and gradients of loss functions at each stage depends only on the gradients of loss functions at higher stage. Hence, the gradient w.r.t the model parameter at stage $k$ is

$$\frac{\partial L}{\partial \mathbf{w}_k} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial \mathbf{w}_k} = \sum_{t=k}^{T} \frac{\partial L_t}{\partial \mathbf{w}_k}$$

Since the features of higher stage classifiers depend only on the features of the previous stage, computing gradients of the cascade can perfectly be embedded in standard backpropagation algorithm. In practice, we can simply insert branches at each layers of the network and put classifiers followed by individual loss layers. This formulation is very related to recent works[14, 8]. However, they only used intermediate loss function as auxiliary loss function to help to efficiently train final stage classifier. In our case, every intermediate classifiers plays important role in entire cascade framework.

### 2.2 Network Architecture

In this work, we explored and built DC-CNN on top of two most popular CNN architectures[13, 5], depicted in figure 1. For the VGG16 network, each convolutional layer is considered as one stage of the cascade. Input size will shrink by factor of 2 after every max-pooling layer(in total 5, so factor of 32) and we put the classifiers after the pooling layers. On the other hand, for the residual network, each basic building block is considered as one stage of the cascade. Each basic building block are constructed by two convolutional layers and one shortcut layer. No pooling layers were used and strided convolution was used for size reduction. Even if the number of layers in residual network is higher than vgg16, it requires lower computational cost and has smaller number of parameters. In our experiments, DC-CNN based on vgg16 networks performed slightly better than one based on residual network. For classifiers at each stage, we used simple average pooling layer followed by linear classifier, which was explored in several places [5, 11].

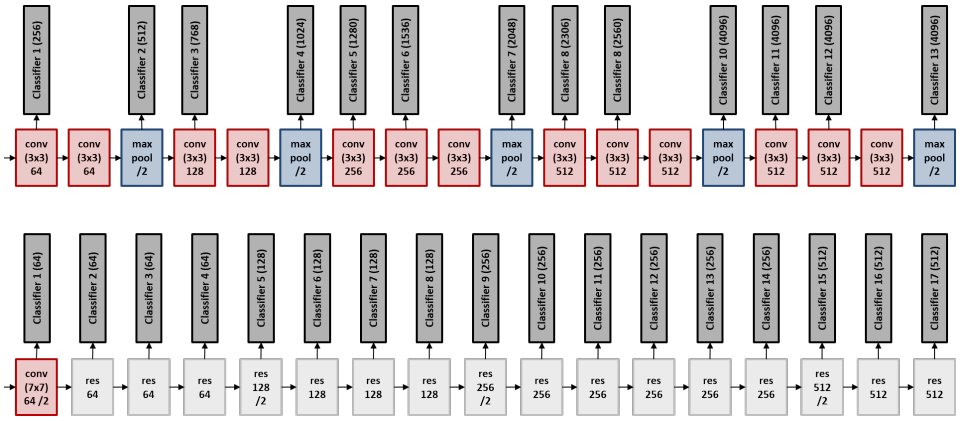


Figure 1: Deeply Cascaded Convolutional Neural Network(DC-CNN): **Top.** DC-CNN based on vgg16 network[13]. In total, it has 13 stage classifiers. 12 of them are located right after the convolutional layers and final one is two fully connected layer appeared in original vgg16 network. **Bottom.** DC-CNN based on 34 layer residual network[5]. It has 17 stage classifiers. The first one after the first convolutional layer and rest of them are located after every basic building blocks of residual network. Each basic building block consists of two convolutional layers and one shortcut layer. ReLU and batch normalization layers were omitted for simplicity.

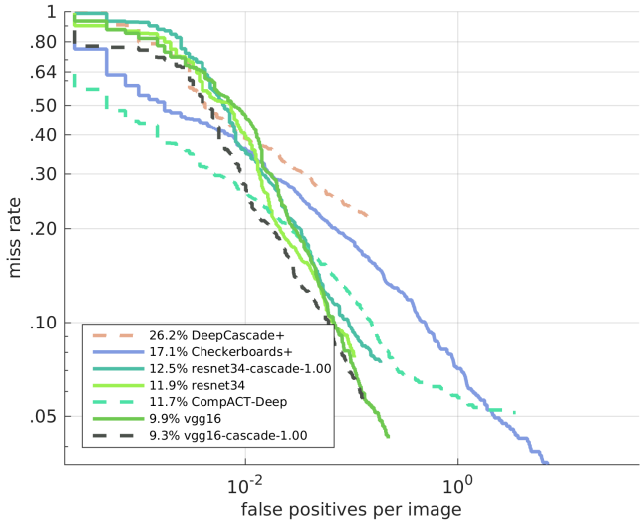


Figure 2: Results of our baseline method on the Caltech testset: *vgg16* is normal deep network with only one classifier at the end of the network. *vgg16-cascade-1.00* is trained with intermediate classifiers via joint soft cascade training, but only last classifier was used for testing. *Checkerboards+* is our proposal method[6]. *DeepCascade+* is a cascade method that used two small deep networks and fast proposal method[1]. And *CompACT-Deep*[3] combines deep CNN features and other handcraft features together with complexity aware boosting mechanism. The number next to the name of methods is area under curve(AUC).

# 3 Experiments

## 3.1 Datasets

We performed the experiments on standard Caltech Pedestrian detection dataset[4]. It is collected from a dashboard camera and contains suburban and city scenes. It has 11 video streams. The first five videos are used for training, the sixth training video is used for validation, and rest of videos are

Table 1: Performance of stage classifiers on validation set: *vgg16* model has 13 stage classifiers and *resnet-34* model has 17 stage classifiers. We measured accuracy on positive examples(*pos*) and negative(*neg*) examples. *total* indicates accuracy for both of them. *complexity* represents floating point operations required, scaled 0 to 1.

| | **vgg16** | | | | **resnet-34** | | | |
|---|---|---|---|---|---|---|---|---|
| stage | pos | neg | total | complexity | pos | neg | total | complexity |
| 1 | 0.710 | 0.590 | 0.600 | 0.004 | 0.582 | 0.621 | 0.618 | 0.008 |
| 2 | 0.590 | 0.680 | 0.670 | 0.093 | 0.671 | 0.681 | 0.680 | 0.073 |
| 3 | 0.650 | 0.800 | 0.790 | 0.137 | 0.757 | 0.673 | 0.679 | 0.137 |
| 4 | 0.767 | 0.832 | 0.827 | 0.226 | 0.830 | 0.676 | 0.688 | 0.202 |
| 5 | 0.735 | 0.929 | 0.915 | 0.271 | 0.874 | 0.683 | 0.697 | 0.252 |
| 6 | 0.839 | 0.935 | 0.928 | 0.360 | 0.892 | 0.751 | 0.761 | 0.316 |
| 7 | 0.878 | 0.951 | 0.945 | 0.450 | 0.913 | 0.807 | 0.815 | 0.381 |
| 8 | 0.886 | 0.961 | 0.955 | 0.494 | 0.926 | 0.811 | 0.819 | 0.445 |
| 9 | 0.891 | 0.972 | 0.966 | 0.586 | 0.932 | 0.817 | 0.825 | 0.497 |
| 10 | 0.881 | 0.980 | 0.973 | 0.679 | 0.931 | 0.852 | 0.858 | 0.562 |
| 11 | 0.889 | 0.981 | 0.975 | 0.705 | 0.925 | 0.899 | 0.901 | 0.626 |
| 12 | 0.894 | 0.980 | 0.974 | 0.734 | 0.917 | 0.932 | 0.931 | 0.690 |
| 13 | 0.921 | 0.974 | 0.970 | 1.000 | 0.915 | 0.945 | 0.943 | 0.755 |
| 14 | | | | | 0.914 | 0.950 | 0.947 | 0.819 |
| 15 | | | | | 0.918 | 0.951 | 0.948 | 0.871 |
| 16 | | | | | 0.921 | 0.951 | 0.949 | 0.935 |
| 17 | | | | | 0.920 | 0.950 | 0.948 | 1.000 |

used for testing. Inspired by [6], we increase the training data tenfold by sampling one out of three frames(instead of one out of thirty frames in the standard setup). We used the proposal method that was used in [6]. By using this proposal method, we collected negative samples. In total we could collect total 17,000 positive samples and 230,000 negative examples.

### 3.2 Baseline results compared to existing methods

Figure 2 shows the results of our baseline methods. We could achieve better results than current state-of-the-art results by [3]. It is mainly because we used deeper layer network(*vgg16*). Surprisingly, when we trained network with intermediate classifiers jointly, we could get better results than the one with only one classifier at the end of the network(compare *vgg16* and *vgg16-cascade-1.00*). We believe it is because the fact that intermediate losses are sometimes helpful to learn better feature representation[14, 8].

### 3.3 Performance of stage classifiers

Table 1 shows classification accuracy of positive and negative examples from validation set, when we used only each stage classifier. As we expected, classifiers in deeper stage tend to achieve better accuracy.

### 3.4 Performance of deeply cascaded networks

Figure 3 and 4 shows the results of deeply cascaded networks. We set the threshold of each layers manually and the computational complexity was presented in the legends. For examples, *vgg16-cascade-0.5* means it only requires half of computational complexity compared to vgg16 network without cascade. For both of *vgg16* and *resnet-34* models we could reduce the computation to almost half of the original model without losing accuracy.

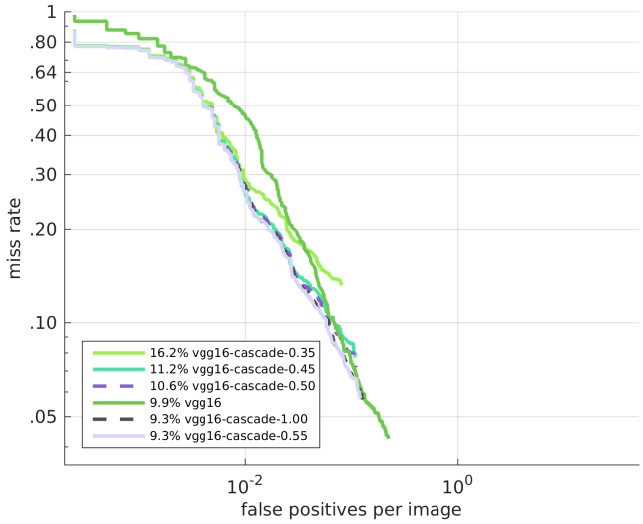


Figure 3: Results of deeply cascaded network(vgg16)

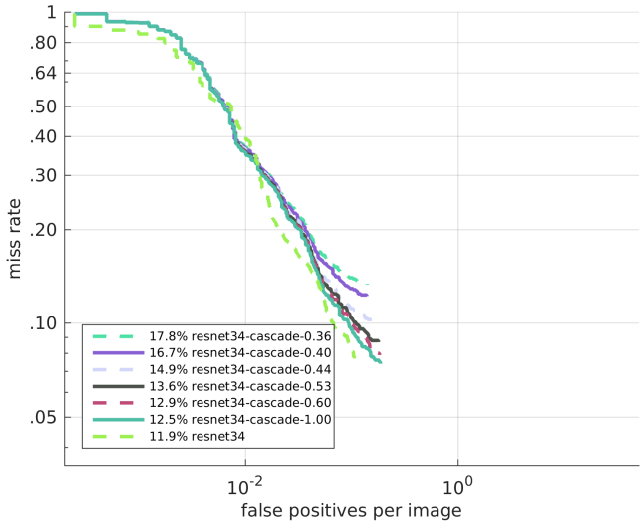


Figure 4: Results of deeply cascaded network(resnet-34)

## 4 Conclusion and Future Work

We introduced new cascade framework within a single deep convolutional neural network and showed that it can achieve same accuracy with half of the computation complexity. Followings are several suggestions for further improvement.

First, the proposal method we used in this work is already very good detector, which takes more than 1 seconds to process an image on fast GPU. For real-time purpose, this proposal method is not suitable. Thus, we need to use more computationally efficient but less accurate proposal methods, such as the one used in [1]. In this case, the proposal method will produce more negative examples and many of them will be rejected in the early stage of the cascade, which leads much dramatic computational complexity reduction in our cascade framework.

Second, we could explicitly introduce boosting method in our soft cascade training. Joint boosting method[9] could be incorporated in our training procedure so that earlier stage classifiers focus on the rejection of easy negative examples.

## References


[1] Anelia Angelova, Alex Krizhevsky, Vincent Vanhoucke, Abhijit Ogale, and Dave Ferguson. Real-time pedestrian detection with deep networks. In *BMVC*, 2015.

[2] Lubomir Bourdev and Jonathan Brandt. Robust object detection via soft cascade. In *CVPR*, 2005.

[3] Zhaowei Cai, Mohammad Saberian, and Nuno Vasconcelos. Learning complexity-aware cascades for deep pedestrian detection. In *ICCV*, 2015.

[4] P. Dollár, C. Wojek, B. Schiele, and P. Perona. Pedestrian detection: A benchmark. In *CVPR*, 2009.

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *arXiv prepring arXiv:1506.01497*, 2015.

[6] J. Hosang, R. Benenson, M. Omran, , and B. Schiele. Taking a deeper look at pedestrians. In *CVPR*, 2015.

[7] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

[8] Chen-Yu Lee, Saining Xie, Patrick Gallagher, Zhengyou Zhang, and Zhuowen Tu. Deeply-supervised nets. In *AISTATS*, 2015.

[9] Leonidas Lefakis and Francois Fleuret. Joint cascade optimization using a product of boosted classifier. In *NIPS*, 2010.

[10] Haoxiang Li, Zhe Lin, Xiaohui Shen, Jonathan Brandt, and Gang Hua. A convolutional neural network cascade for face detection. In *CVPR*, 2015.

[11] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. 2013.

[12] Vikas C. Raykar, Balaji Krishnapuram, and Shipeng Yu. Designing efficient cascaded classifiers: Tradeoff between accuracy and cost. In *KDD*, 2010.

[13] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

[14] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *CVPR*, 2015.

[15] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *CVPR*, 2001.

[16] Fan Yang, Wongun Choi, and Yuanqing Lin. Exploit all the layers: Fast and accurate cnn object detector with scale dependent pooling and cascaded rejection classifier. In *CVPR*, 2016.