Object Detection in Video using Faster R-CNN

Prajit Ramachandran[*]
University of Illinois at Urbana-Champaign
prmchnd2@illinois.edu

Abstract

Convolutional neural networks (CNN) currently dominate the computer vision landscape. Recently, a CNN based model, Faster R-CNN [1], achieved state-of-the-art performance at object detection on the PASCAL VOC 2007 and 2012 datasets. It combines region proposal generation with object detection on a single frame in less than 200ms. We apply the Faster R-CNN model to video clips from the ImageNet 2015 Object Detection from Video challenge [2]. By leveraging additional temporal information from the video, we ranked 3$^{\text{rd}}$ place in terms of mean average precision. This work is being presented as a poster at the ICCV 2015 ImageNet and MS COCO Visual Recognition Challenges Joint Workshop.

1 Introduction

Object detection is the task of recognizing and localizing objects in an image. Traditionally, sliding windows or region proposals have been used to generate object hypotheses, which are then classified using a model like the Deformable Parts Model [3]. The meteoric rise of convolutional neural networks [4][5] for computer vision has prompted research into applying CNNs for objection detection. Recently, Faster R-CNN [1] was proposed to accurately detect objects in pictures. It achieved a state-of-the-art 73.2 mean average precision (mAP) on the PASCAL VOC 2007 dataset. By combining both region proposals and the classifier into one large network, it is able to automatically learn good feature representations for the task.

With the release of the ImageNet VID dataset [2], we apply the Faster R-CNN model to the video clips. We find that Faster R-CNN has good performance without any modifications for video (e.g. treating each frame independently). However, if we incorporate temporal information into the model, we are able to significantly boost performance.

2 Model

2.1 Convolutional Neural Networks

CNNs differ from feedforward neural networks because of their convolutional and pooling layers. In convolutional (conv) layers, independent filters (usually square) are convolved with all the previous layer's channels. For example, in the very first conv layer, filters are convolved with the input image's red, green, and blue channels. The filter is of size (h, w, c), where h and w are the height and width of the filter respectively, and c is the number of input channels (e.g. 3 for the very first conv layer). The responses across channels are then summed after the linear filtering. This process is repeated for multiple independent filters, and the activations of each filter serve as the channels for the next layer. Unlike using pre-made filters like the Sobel operator, the filter weights are learned during training. Stochastic gradient descent with backpropagation is used to learn the

[*]Work done in collaboration with Professor Thomas Huang's Image Formation and Processing Group (IFP) at UIUC. Submission under the name **UIUC-IFP**.

filter parameters. By stacking multiple conv layers, it is possible to learn higher order features in the deeper layers [6]. These conv features far outperform hand-crafted features, which has led to the dominance of CNNs in computer vision [7].

Pooling is often used in CNNs. Pooling is an operator (e.g. max or average) that is applied to spatial neighborhoods in order to reduce dimensionality. Each channel is pooled independently, and is the pooling is applied in strides, which corresponds to the magnitude of dimensionality reduction. For example, a pooling layer with a stride of 2 by 2 will reduce the size of the width and height of the layer input by 2. This dimensionality reduction is useful because images are high dimensional inputs, and it is computationally expensive to apply hundreds of convolutions to an image of size 224 by 224 pixels. Furthermore, in the case of max pooling, a degree of translation invariance is had because the largest response in an neighborhood is selected. This means that it does not matter where in the local neighborhood an object is in, creating translational invariance.

2.2 Fast R-CNN

Fast R-CNN [8] (distinct from Faster R-CNN, which is described below) is an improvement on the original R-CNN [9] designed to speed up the detection network. First, bounding boxes are generated for the image using bounding box proposal methods like Selective Search [10] or Edge Boxes [11]. Next, the entire image is passed into the CNN, up through the last conv layer (e.g. through conv5 in VGGNet). Then for each bounding box, Region of Interest (RoI) pooling is applied to the final layer conv features to create a fixed size vector. This RoI vector is then passed to a classification network and regressor network. The classification layer is a fully connected layer followed by a softmax over all object classes. The regressor network is a 2-layer network that fine-tunes the bounding box coordinates and dimensions. The fine-tuning produces notably improved bounding boxes. Finally, non-maxima suppression is applied over all boxes for each class independently.

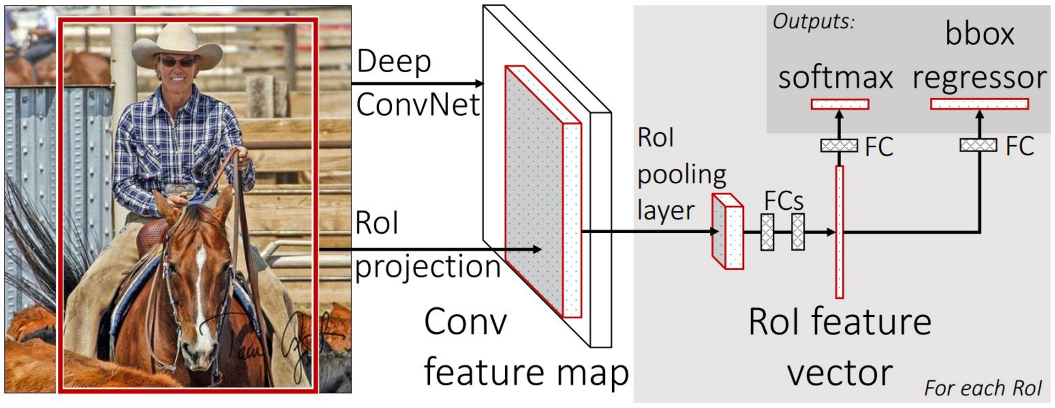

Figure 1: Fast R-CNN architecture (credit [8]).

In RoI pooling, the boundaries of the bounding box are projected onto the conv layer. This is not a one-to-one mapping because the height and width of the final conv layer is significantly smaller than the original input dimensions due striding in conv and pooling layers. To account for this, let the striding factor S be the product of strides sizes in all the conv and pooling layers (for simplicity, this assumes the horizontal and vertical strides are equal, but it is trivial to extend to non-equal strides). For example, in VGGNet, this would be 16, because of the 2 by 2 strides in each of the four pooling layers prior to conv5. Then, divide the top-left and bottom-right coordinates of the original bounding box by S, and take the floor and ceiling of the values respectively. This defines the bounding box region in conv feature map coordinates. Finally, split the conv bounding box region into H by W equal sized windows and max pool inside each block. This results in an H x W fixed size vector for each channel. H and W are constant hyperparameters that depend on the network architecture (for example, VGGNet requires $H = W = 7$ to serve as input to the fully-connected layer).

The network is trained with a multi-task loss,

$$L(p, c, t^u, b) = L_{cls}(p, c) + \lambda[c \neq \text{background}]L_{reg}(t^u, b)$$

p is the probability distribution over $K + 1$ object categories predicted by the classification layer. The distribution includes the background category (i.e. no object in box). t^u is the bounding box predicted by the regression network. c is the ground truth class and b is the ground truth bounding box. $[c \neq \text{background}]$ is an indicator function which takes a value of 1 iff the proposed bounding box contains an object; otherwise, it is a background box and the indicator function takes a value of 0, so regression loss is not used for training on background boxes.

The key advantages of Fast R-CNN are its speed and trainability. Fast R-CNN is faster than R-CNN because it reuses the same conv features for each bounding box, meaning the network only has to do expensive convolutions over the input image once. Whereas R-CNN requires running the convolutional layers for each bounding box proposal. Furthermore, with the exception of the generation of bounding boxes (which comes from an external method), the entire network is trainable end-to-end.

2.3 Region Proposal Network

Unfortunately, the method of relying on external bounding boxes proposals is both slow and suboptimal. It is slow because the bounding boxes have to be generated on the CPU while the model runs on the GPU. It is also suboptimal because the proposal method is not trained jointly with the network. Thus, Faster R-CNN uses a Region Proposal Network (RPN) in order to generate the bounding box proposals. Faster R-CNN is simply the RPN combined with a Fast R-CNN detector network. Notably, the RPN and Fast R-CNN network share the same convolutional layers, allowing for joint training.

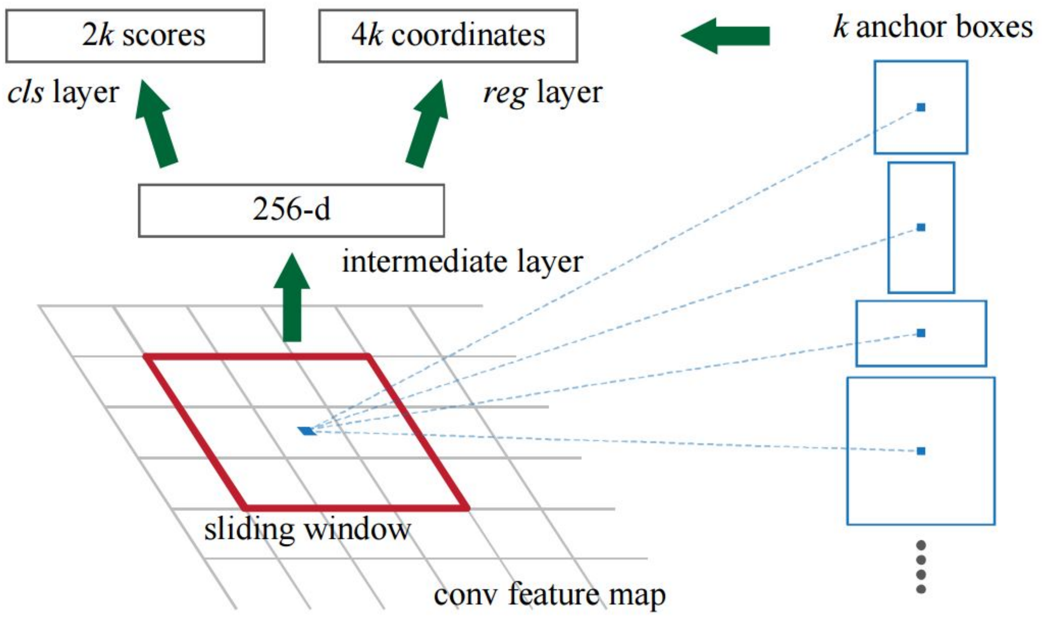

Figure 2: The RPN generating bounding boxes for one position in the final layer convolutional feature map (credit [1]).

The RPN utilizes a sliding window approach in order to generate k bounding boxes for each position in the last layer conv feature map (e.g. conv5 for VGGnet). First, an n by n filter is convolved with last layer conv feature map. Then the result is projected to a lower dimensional space by convolving with a 1 by 1 filter (which just linearly combines the channels for each position independently), resulting in a fixed-size vector for each position. The vector is separately passed into a box-regression layer and a box-classification layer. In the box-regression layer, k bounding boxes are generated relative to the current position in the conv feature map (the current anchor point). For example, if the vector for position (x_a, y_a) is passed into the box-regression layer, then the layer will have $4k$ outputs. Each set of of 4 outputs parameterizes a bounding box relative to (x_a, y_a). The four parameters are t_x, t_y, t_w, t_h, where

$$t_x = (x - x_a)/w_a \qquad t_y = (y - y_a)/h_a \qquad t_w = \log(w/w_a) \qquad t_h = \log(h/h_a)$$

Each of the k generated bounding boxes will have a different fixed w_a and h_a, which results in boxes with different areas and aspect ratios (because of different absolute and relative magnitudes of w_a and h_a).

The box-classification layer generates $2k$ outputs, where each pair of 2 outputs is the probability that the corresponding bounding box has an object in it or is just background. That is, the sum of each pair of outputs is 1, and is the probability distribution over whether the bounding box contains an object or not. To reduce the number of bounding box proposals, non-maxima suppression is used on proposals that have high intersection-over-union (IoU) with each other. The boxes are ranked based on the object probability score. Finally, the top N proposals are taken from the remaining proposals, again ranked by object probability score.

The RPN is trained using a multi-task loss extremely similar to that of the detector Fast R-CNN network. A proposal is designated as a positive training example if it overlaps with a ground-truth box with an IoU greater than a predefined threshold (e.g. 0.7), or if it is the bounding box that has the highest IoU with a ground truth. A proposal is designated as a negative example if its maximum IoU with any ground truth box is less than another predefined threshold (e.g. 0.3). Having a large margin between the two thresholds avoids learning on ambiguous cases. The ground truth regression outputs t^*_x, t^*_y, t^*_w, and t^*_h are parameterized in the same manner as the normal outputs (i.e. compared against the anchor location (x_a, y_a), box width w_a, and box height h_a). The multi-task loss has a classification component, and a regression component that is only trained on for positive examples, similar to the Fast R-CNN detector network loss.

These proposals are then passed into the Fast R-CNN detector network and classified as described in the previous section. Since the RPN and detector networks share the same convolutional layers, they can be trained jointly. However, since the R-CNN network expects fixed proposals, training may not converge if the RPN is training simultaneously, since training will change the distribution of proposals. Thus, in practice the networks are trained in multiple stages. First, the RPN is trained by itself. Then a detector network with separate conv layers is trained using the RPN's proposals. Afterwards, an RPN is trained using the detector network's conv layers. However, the conv layers are fixed, so only the RPN specific layers are trained. Finally, the first detector network's fully connected layers are fine-tuned, using the new RPN's proposals.

3 Leveraging temporal information

Faster R-CNN is designed for object detection in a single independent frame. Since the competition is for object detection in videos, it would be a waste of salient information to process frames independently. The most obvious idea of extending Faster R-CNN to leverage temporal information is by adding a recurrent layer between consecutive frames. This merge of CNN and recurrent neural network (RNN) is ideal in the sense that the network will learn how to pass important information between frames without needing hand-tuned features.

However, the CNN-RNN hybrid is infeasible because of hardware and training limitations. The activations of the convolutional layers in CNNs take up a large amount of memory, and they are required to be held in memory for backpropagation. However, GPU memory is one of the biggest limitations in the training of very large networks. The amount of memory required to hold convolutional activations for every frame in a video at the same time is far beyond the the memory capacity of current GPUs. Furthermore, it is unclear apriori if it is even possible to train such a large network to convergence. The deeper the network is, the harder it is to train, and the RNN aspect makes the CNN-RNN extremely deep. For these reasons, we opted for simpler methods of leveraging temporal information.

3.1 Neural Network Suppression

One problem we noticed with Faster R-CNN on the validation set was that non-maxima suppression (NMS) frequently chose the wrong bounding box after object classification. It would choose boxes that were overly large, resulting in a smaller intersection-over-union (IoU) with the ground truth box because the union of areas was large. The large boxes often had very high object scores, possibly because more information is available to be extracted during RoI pooling.

In order to combat this problem, we attempted to use temporal information to re-rank boxes. The hypothesis is that neighboring frames should have similar objects, and their bounding boxes should be similar in position and size. The idea is that if we could take advantage of the multitude of bounding boxes over time, we could more accurately localize the object in any given frame. Most of the information would come from neighboring frames. It is also possible for a distant frame to provide valuable information, such as in the case where an object becomes partially occluded in the middle of the clip, making it hard for a single frame detector to classify the object.

We propose neural network suppression (NNS), an RNN that attempts to use temporal information to improve bounding boxes. There are three main design choices for this model.

1. What information is passed into the network, and how? Possible choices could be passing all the bounding boxes or passing a pre-selected subset of bounding boxes. The first requires the use of another sub-RNN to process the variable number of bounding boxes, while the latter requires a robust algorithm to pick bounding boxes without losing good candidates.
2. How can information from other frames be utilized? The simplest method is to use recurrent connections on the higher level features computed by the network.
3. What should the network output? This can either be new scores for each bounding box to be used for re-ranking, or new box coordinates for every object detected in the frame, which corresponds to another regression layer.

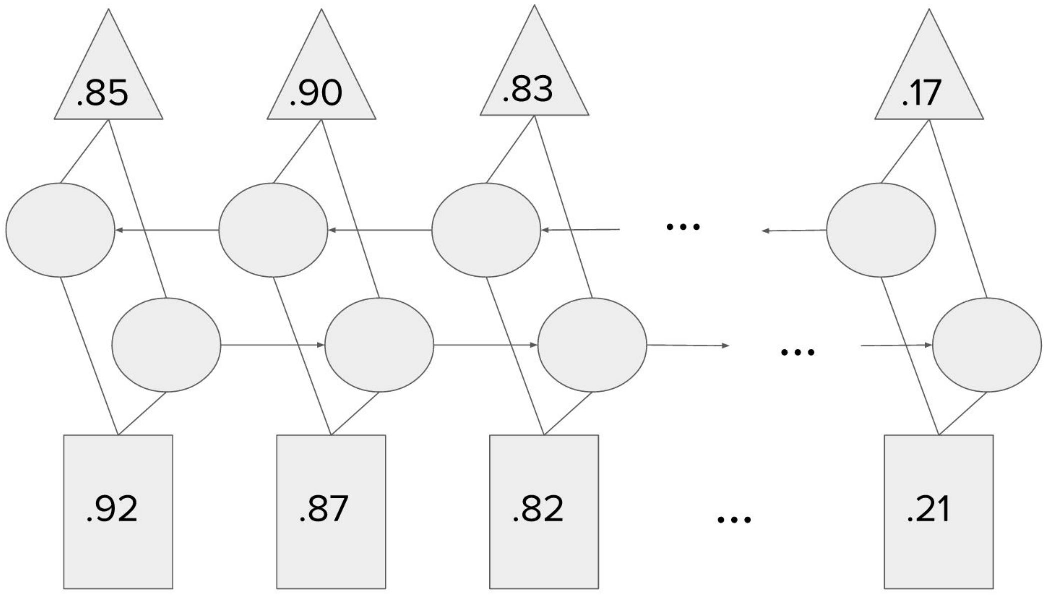

Figure 3: The architecture chosen for NNS. The new object scores leads to different boxes being suppressed by non-maxima suppression.

We were extremely limited by time and computational resources, so we only had the opportunity to experiment with one very basic NNS architecture. As inputs, we passed in the bounding box coordinates and score for the top N boxes, where $N = \min(500, \text{number of proposed boxes})$. In order to reduce the number of parameters and leverage the information across boxes, we connected the inputs with a bidirectional recurrent layer. The boxes were ordered by their detector object score. The hidden layers of the bidirectional layer were concatenated and passed through a fully connected layer with shared parameters across boxes. The result was a new object score which we then applied NMS over. NNS was run over the bounding boxes of each class separately.

Unfortunately, we were not able to get this architecture to work. The network fit well to a subset of the data for a class, but would fail to generalize when applied to a larger dataset. We suspect that

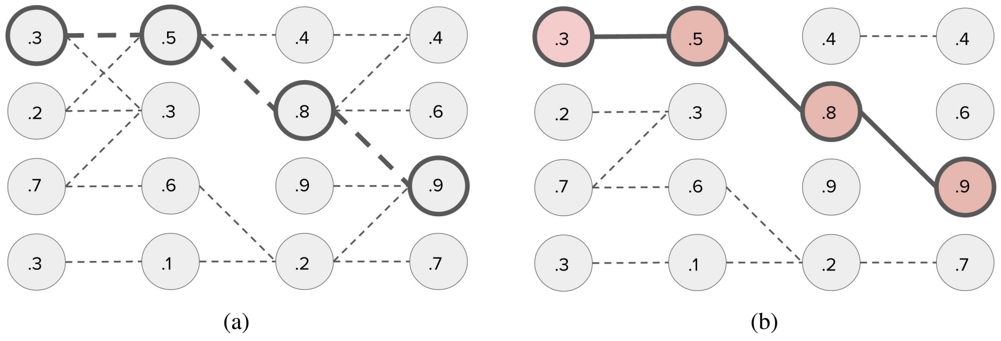

Figure 4: An example of temporal chain re-weighting. Each node represents a bounding box with its object score, each column represents a frame, and edges denote the linkability of two bounding boxes in neighboring frames. In (a), the maximum score chain, with total score $.3 + .5 + .8 + .9 = 2.5$, is found using dynamic programming. In (b), the nodes of the maximum score chain and its corresponding edges are removed from the graph. Not shown is the extra intra-frame suppression step that further prunes nodes and edges. The process is repeated until a chain of length 1 is returned.

improvements could be had if we actually used recurrent connections across time and if we improve the input representation (e.g. pass conv features). We have the entire pipeline to train the various types of NNS, but simply did not have the time and resources to test these different architectures. We hope to continue experimenting with these ideas. Other groups have independently noted that replacing NMS with a neural network is a promising future direction [12].

3.2 Temporal Chain Re-weighting

Instead of NNS, we ended up using a simpler, heuristic method of re-ranking bounding boxes. It involves linking bounding boxes over time and using a function of the object scores across the chain to re-weight each bounding box. After re-weighting, NMS is applied as normal. We found this method gave good results despite its heuristic nature.

For each pair of neighboring frames, a bounding box in the first frame can be linked with a bounding box in the second frame iff their IoU is above some threshold. We find potential linkages in each pair of neighboring frames across the entire clip. Then, we attempt to find the maximum score chain across the entire clip. That is, we attempt to find the sequence of linkages such that the sum of object scores of all the boxes in the sequence is maximized. This can efficiently be found by using a very simple dynamic programming algorithm that maintains the maximum score chain so far at each box. The algorithm returns a set of chained boxes $(b_i, b_{i+1}, \ldots, b_j)$. The chained boxes are then removed from the set of boxes we link over. Furthermore, we apply suppression within frames such that if a bounding box in frame $t, t \in [i, j]$, has an IoU with b_t over some threshold, it is also removed from the set of candidate boxes. This algorithm is then repeated until a length 1 chain is returned. This algorithm can be viewed as applying dynamic programming on a graph, as illustrated by figure 4.

After chains are extracted, the constituent bounding boxes can be reweighted. The simplest reweighting scheme is to set each bounding box's object score to the average of object scores across the chain. Another approach is to set the score to the maximum score across the chain. We experimented with both approaches. Afterwards, NMS is applied to each frame as usual to produce the final bounding boxes.

4 Results

We trained our Faster-RCNN model on a subset of the training data provided by the challenge. We heavily modified the open source Faster R-CNN Caffe code released by the authors[1]. We did not have the time or computational resources to train on the entire dataset, so we chose to only train on the initial dataset released by the challenge (the full dataset was released in two parts).

[1]https://github.com/rbgirshick/py-faster-rcnn

Table 1: Class mean average precision for validation set.

Class	**mAP**	**Class**	**mAP**	**Class**	**mAP**
airplane	0.764	elephant	0.417	red_panda	0.295
antelope	0.783	fox	0.921	sheep	0.768
bear	0.476	giant_panda	0.57	snake	0.526
bicycle	0.543	hamster	0.908	squirrel	0.051
bird	0.357	horse	0.725	tiger	0.195
bus	0.656	lion	0.13	train	0.918
car	0.367	lizard	0.694	turtle	0.757
cattle	0.584	monkey	0.02	watercraft	0.707
dog	0.276	motorcycle	0.672	whale	0.375
domestic_cat	0.627	rabbit	0.132	zebra	0.69

We used VGGNet as our Faster-RCNN convolutional backend. We found using VGGNet gave a substantial improvement over using the architecture described by Zeiler and Fergus [6]. Temporal chain reweighting gave further improvements. We chose either the maximum or average score across the chain separately for each class based on validation performance.

On the validation set, we achieved a mAP of 0.536 (see table 1 for a full breakdown across classes). We performed well for classes like train and hamster, where the object is often prominently featured in each frame. We did poorly on classes like lion and monkey, where the object is either small or heavily occluded. On the test set, we ranked 3rd in terms of mean average precision (mAP). Our submission achieved a mAP of 0.487232[2].

4.1 Discussion

Not being able to train on the entire dataset definitely hurt our performance significantly. Furthermore, due to time constraints, we could not even train until convergence on the smaller dataset. Given more time and computational resources, it is possible that we could have achieved 2nd place in terms of mAP. We also did not have the time or resources to experiment with neural network suppression sufficiently, which may have given a larger boost compared with temporal chain reweighting. Hyperparameter tuning would also improve performance. Future work should focus on addressing classes on which our model does poorly.

4.2 Contribution

I personally worked on the entire neural network suppression pipeline, writing and speeding it up significantly. I wrote a clip based sampling procedure (otherwise training would be biased towards longer clips). I helped write many scripts designed to test research ideas (like an IoU based regression loss function). I also participated in general planning and brainstorming.

5 Conclusion

By using the strong baseline of Faster R-CNN and leveraging additional temporal information, we were one of the top performers in the ImageNet Object Detection from Video challenge. We would like to continue pursuing improvements to our submission, including training on the entire dataset, experimenting with neural network suppression, and performing a deeper analysis on our model designed to elucidate its weaknesses.

References

[1] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *Neural Information Processing Systems (NIPS)*, 2015.

[2]http://image-net.org/challenges/LSVRC/2015/results

[2] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[3] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part based models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(9):1627–1645, 2010.

[4] Yann LeCun, Lon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, volume 86, pages 2278–2324, 1998.

[5] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J.C. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[6] Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *Computer Vision - ECCV 2014 - 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I*, pages 818–833, 2014.

[7] Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. CNN features off-the-shelf: an astounding baseline for recognition. abs/1403.6382, 2014.

[8] Ross Girshick. Fast R-CNN. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 2015.

[9] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014.

[10] J.R.R. Uijlings, K.E.A. van de Sande, T. Gevers, and A.W.M. Smeulders. Selective search for object recognition. *International Journal of Computer Vision*, 2013.

[11] C. Lawrence Zitnick and Piotr Dollár. Edge boxes: Locating object proposals from edges. In *ECCV*. European Conference on Computer Vision, September 2014.

[12] Justin Johnson, Andrej Karpathy, and Fei-Fei Li. Densecap: Fully convolutional localization networks for dense captioning. abs/1511.07571, 2015.