Learning of Visualization of Object Recognition Features and Image Reconstruction

Qiao Tan
qtan@stanford.edu

Yuanlin Wen
yuanlinw@stanford.edu

Chenyue Meng
chenyue@stanford.edu

***Abstract*— We introduce algorithms to visualize feature spaces used inin object recognition systems by inverting a visual feature back to a natural gray scale image. By investigating algorithms including binary- and multi-class SVM, pairwise dictionary and CNN, we succeed in boosting the correlation measures up to 0.846 when comparing the inverted and original images and speed up the inversion process to a few seconds on a desktop computer.**

I. INTRODUCTION

Feature visualization techniques have been inspiring many applications in the field of computer vision. Computer vision researchers have been relying on these visualizations to understand object recognition systems as well as to reveal information encoded by features [21], to find security failures in machine learning systems [4], to fix problems in convolutional neural networks [18], etc.

It makes feature visualization a even more promising tool to complement failures in the computer vision systems when Parikh and Zitnick [13] introduced a new paradigm for human debugging of object detectors through visualizations. By inverting visual features back to natural images and thus providing more perceptually intuitive representation of features, we can gain better understanding of the failures and behaviors of object detection systems and future boost their performance.

A great amount of efforts have been put in developing effective feature inversion algorithms. Weinzaepfel et al [17] were the first to reconstruct an image given its keypoint SIFT descriptors. Their approach obtains compelling reconstruction performance by using a nearest neighbor based approach on a massive database. This method analytically solves for the inverse image problem yet requires an extensive dataset. In later work, Zeiler and Fergus [19] presented a method to visualize activations from a convolutional neural network and Girshick et al [8] proposed to visualize convolutional nueral networks by finding images that activate a specific feature. Although these methods achieve to reconstruct and visualize images from their respective features, they are either tailored to a specific feature or computationally expensive.

In this project, we aim at leveraging advantages in algorithms presented in previous work and developing our own feature visualization algorithms. Namely, we will use multi-class SVM, pairwise dictionary and CNN to invert the input feature descriptors of particular image and produce a reconstructed image as the output.

In the following sections of this report, we will first disclose details of features and dataset we have used and then describe learning algorithms we have proposed to solve the inversion problem. After that, we will present the training and validation results, followed by a section of discussion with insights.

II. DATASET AND FEATURES

A. Dataset

Our training and test images are drawn from Microsoft Object Class Recognition image databases[1], which consist of 240 pixel-wise labelled images (each of size 213×320) from six object classes.

B. Feature Selection

One common way of performing object recognition is to extract HOG features from the original image and train a linear SVM classifier on a set of HOG features. Compared with other feature descriptors which could have been used for object recognition, like SIFT or SURF which compute the gradient histograms only for patches around specific interest points, HOG counts occurrences of gradient orientation in all localized portions of an image and thus capture the gradient information over the full image. Consequently, to reconstruct the whole original image, we are more interested in first trying building our feature space on HOG descriptors rather than other local descriptors.

III. METHODS

A. Binary-class Support Vector Machine

In the first method, we try to invert HOG features into black-white images and use multiple binary-class SVM classifiers to recover pixel values (either 0 or 1).

First, we divide the original image into multiple non-overlapping 8×8 cells. For each cell i, we extract its HOG feature by computing its histogram over 32 bins (which is interpreted by a 32-value vector $\vec{h}^{(i)}$) to represent the cell block. In other words, for each pixel in an 8×8 cell region, its corresponding feature vector is $\vec{h}^{(i)}$. Then to predict all the 64 values (either 0 or 1) in a cell, we need to build 64 SVMs, one for each spot in the cell region.

We regard this method as one baseline method which other methods can compare to.

B. Multi-class Support Vector Machine

Support Vector Machines (SVM) were originally designed for binary classification problem, yet many experts are engaged in effectively extending it for multi-class classification. Several methods have been proposed where typically we construct a multi-class classifier by combining several binary classifiers. The most popular methods are "one-against-all" method, which builds one SVM per class, training it to distinguish samples in a single class from the samples in the remaining classes, and "one-against-one" method, which builds one SVM for each pair of classes. As discusses in citehsu2002comparison, the second method is more efficient in computing the inversions than the first one. So we will use the "one-against-one" strategy in the multi-class SVM training method.

The one-aginst-one method was first used in [10]. This method constructs $\frac{k(k-1)}{2}$ classifiers where each one is trained on data from two classes. For training data from the i_{th} and the j_{th} classes, we solve the following binary classification problem:

$$\begin{aligned}\underset{\omega^{ij},b^{ij},\xi^{ij}}{\operatorname{argmin}} \quad & \frac{1}{2}(\omega^{ij})^T\omega^{ij} + C\sum_t \xi_t^{ij} \\ & (\omega^{ij})^T\phi(x_t) + b^{ij} \geq 1 - \xi_t^{ij}, \text{if} y_t = i, \\ & (\omega^{ij})^T\phi(x_t) + b^{ij} \leq -1 + \xi_t^{ij}, \text{if} y_t = j, \\ & \xi_t^{ij} \geq 0.\end{aligned}$$

We use the following voting strategy suggested in [10] to perform the multi-class classification: if $\text{sign}((\omega^{ij})^T\phi(x) + b^{ij})$ says x is in the ith class, then the vote for the ith class is added by one. Otherwise, the jth is increased by one. Then we predict x is in the class with the largest vote. The voting approach described above is also called the "Max Wins" strategy. In case that two classes have identical votes, we simply select the one with the smaller index.

Practically we solve the dual of equation above where number of variables is the same as the number of data in two classes. Hence if in average each class has l/k data points, we have to solve $k(k1)/2$ quadratic programming problems where each of them has about $2l/k$ variables.

C. Pairwise Dictionary

The previous methods aim to reconstruct the images from given descriptors directly at pixel level, which means we approximate every pixel value using a batch of values from the histogram of a cell in the HOG features, regardless of how descriptors and image patterns are tightly related together. In other words, correlations between adjacent pixels are disregarded due to the pixel-level reconstruction, which should be highly preserved in the ideal case. In this section, we describe an algorithm which learns an over-complete pairwise dictionary from patches extracted from images and corresponding HOG descriptors and uses that dictionary to encode descriptors and reconstructs the original images accordingly.

Imagine we can represent an image $x^{(i)} \in \mathbb{R}^{\mathbb{P}}$ and its feature $\phi^{(i)} \in \mathbb{R}^{\mathbb{Q}}$ with a image basis $U \in \mathbb{R}^{\mathbb{P}\times\mathbb{K}}$ and a feature basis $V \in \mathbb{R}^{\mathbb{Q}\times\mathbb{K}}$. We can then estimate U and V such that images and features can be encoded in their respective bases but with shared coefficient $\alpha \in \mathbb{R}^{\mathbb{K}}$ as $x^{(i)} = U\alpha$ and $\phi = V\alpha$. Once U and V have their paired representation, we can then invert features by estimating an αthat reconstructs the feature well. By using the same α obtained from optimizing feature representation in the previous step, we are able to produce reconstructed image $y^{(i)}$ from the feature by computing $y^{(i)} = U\alpha$.

To build up the pairwise dictionary, which consists of a one-to-one mapping from a HOG descriptor basis to a image basis, images patches $x^{(i)}$ and their corresponding HOG features $\phi^{(i)}$ are extracted from PASCAL VOC 2012 [7] and we learn the dictionary by solving the optimization problem as follows using SPAMS [16][12]: (see Fig. **??** for visualization of some learned basis pairs in the dictionary.)

$$\begin{aligned}&\underset{U,V,\alpha}{\operatorname{argmin}} \sum_i ||x^{(i)} - U\alpha_i||_2^2 + ||\phi^{(i)} - V\alpha_i||_2^2 + \lambda||\alpha_i||_1 \\ &s.t.\ ||U||_2^2 \leq \varphi_1 \ \ and \ \ ||V||_2^2 \leq \varphi_2\end{aligned}$$

To encode HOG descriptors using the feature bases in the pairwise dictionary, we use the matching pursuit algorithm proposed by . Empirically, we found success producing fairly good visualization results by encoding features using only a few number of bases (see Results section for full details).

D. Convolutional Neural Network

Convolutional Nerual Networks (CNNs) have been proved to be a powerful method over various computer vision tasks like image classification [11][20], object recognition [5] and detection [9][15]. Typically we train supervised CNNs to output high-level abstract representations taking raw images as inputs. Inspired by [6], we reverse the standard CNN to realize deconvolutional neural network (deCNN) where each layer consists of unpooling, convolution and ReLU. Details will be demonstrated later in Sec. III-D.2. Then we use deCNN to reconstruct original images from their features, which are high-level abstractions of images.

Specifically, our deCNN takes features generated from a gray-scale image as inputs, and outputs the same gray-scale image. We train deCNN using standard backpropagation to minimize the Euclidean distance between the original and reconstructed images. And we use Theano, a Python library to build our neural network [2][3].

1) HOG feature: Based on the analysis in Sec. II and a few attempts like SVM and pairwise dictionary which have been illustrated above in Sec. III, we may have proven that HOG feature is a very good descriptor to represent the high-level abstraction of an image. Because of the plausibility of image reconstruction using HOG feature, we assume deCNN may work fine as well.

However, we failed to use deCNN to reconstruct images from HOG features. Because it is difficult to simulate the process of computing HOG feature using CNN, not to

mention reversing it. According to the illustration in Sec. II, we may assume the calculation of HOG feature requires two major steps: gradient computation and orientation-based histogram computation. Gradient magnitude is computed with one-dimensional filter kernels like $[-1, 0, 1]$ and $[-1, 0, 1]^T$ or 3×3 Sobel masks, which can be realized perfectly with CNN.

The major problem is about orientation-based histogram. When counting the occurrences of different gradient orientations in each cell, it is inevitable to lose specific position information of each gradient orientation. However, position information of gradient is extremely important for unpooling operation in deCNN [20], especially in our case where the region for computing HOG feature is as large as 8×8, which leads to massive position data loss.

We tried using a three-layer deCNN to recover image from its HOG features but only outputs an image full of noise, which further strengthens our previous assumption that HOG feature may not be a proper representation for image reconstruction.

2) CNN feature: To overcome the weakness of HOG feature whose local position information is lost, we extract features of images directly using CNN, which will be much easier and more plausible for deCNN to invert, and those CNN features can also be used for object recognition [14] and classification [11].

Generally we build an auto-encoder which is composed of two layers of convolution operations and two layers of deconvolution operations, shown in Fig. 1.

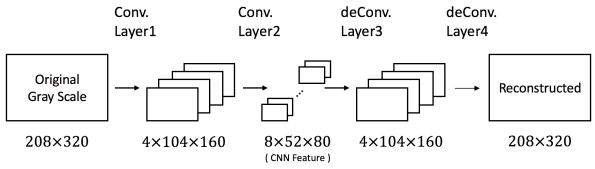

Fig. 1. Architecture of the auto-encoder (CNN + deCNN)

The convolution part, which basically plays a role as feature extraction, takes image as input and outputs its CNN feature. Then the deconvolution part aims to reconstruct the original image, whose input is the extracted CNN feature and output is the reconstructed image. More details of our auto-encoder network, such as unpooling operations and kernel sizes will be illustrated in Sec. IV.

IV. Results

In this section, we show our results of all algorithms we proposed in III and using both quantitative and qualitative measures to evaluate the performance.

A. Binary-class SVM

Training HOG features using binary SVM algorithm, we get the reconstructed image in Fig.2.

In reconstructed image of Fig.2, we can only roughly see the contour of the hat and the face with the mean normalized cross correlation being 0.0127. This result provides us a baseline in evaluating later inversion algorithms.

Fig. 2. We show results for image reconstruction using binary SVM algorithm. left column: original image of Lena. right column: HOG feature inversion.

B. Multi-class SVM

After trying to reconstruct images in the training set, we found that the mean normalized correlation is unacceptably high. However, when we test the model, test error is unacceptably high where the classifiers tend to randomly pick a pixel value to output.

This result can also be shown from correlation value between original and predicted vectors, which are 0.9321, 0.9248, 0.8945 corresponding to the three images in the training set.

At first we set our parameters as: $cellsize = 8, blocksize = 1, number_of_directions = 256$ and $overlap = 0$. From results above, we deduce that the bad performance of testing sets is due to overfitting, so we adjust $number_of_directons = 16$, which means we use 16 features to estimate one pixel point. From experiments and other experiences, we think the reason this method fails should be more than overfitting.

We know that HOG features counts occurrences of gradient orientation in localized portions of an image. Specifically, a group of features describe gradient or edge information of one block. (In our model, we set one block to be a cell of 8*8 pixels.) In order to estimate values of pixels by HOG features, we down-scale our image by 8 so that one group of HOG features can be used to estimate value of the corresponding pixel. It seems to be reasonable but when we dig deeper into the principle of estimation, we find that we are using gradient or edge features to estimate RGB value of one pixel, which is equal to use velocity to estimate distance without knowing an origin. In other words, this method makes no sense even theoretically, not to mention in reality.

After figuring out the problems of visualizing HOG features by multi-class SVM algorithm, we decide to use a different method, Pairwise Dictionary, to fix it.

C. Pairwise Dictionary

We have trained our pairwise dictionary over the training set PASCAL VOC 2012 [7] using SPAM [12] and obtained 1024 pairwise bases, which are used to decompose HOG features in the validation set. We show our inversion results in Fig.4.

Qualitatively, by learning a dictionary over the correlation between HOG features and natural images, pairwise dictionary method tends to produce good visualizations of HOG

features with higher correlation values in the validation set, compared with the baseline method, the binary SVM classifiers. Notice that during the inversion, pairwise dictionary method recovers high frequencies, which are invisible in the original images, without introducing significant noise. As HOG is invariant to illustration changes and amplifies gradient, inverting HOG features provides us with extra surrounding information about the original images and helps us obtain a better understanding about the difference between HOG and human visions in an object recognition system.

We have also discovered that the learnt dictionary is generalized enough to produce fairly good visualization with only few bases. If we only use top n bases with largest coefficients in the matching pursuit algorithm to reconstruct the original images, we end up with producing visualizations with good quality by using few bases (see Fig.3). This observation is significant in reducing computational costs in a sense that we are able to perform fast inversions in a few seconds over HOG features on a desktop computer without losing high level content of the inverse .

Fig. 3. Inverting HOG feature using different number of bases in the dictionary. As we increase number of bases used, edges are captured better in the results.

D. CNN

The convolution layer in Fig. 1 is standard, which consists of three operations: convolution, maxpooling and ReLU. As for the deConvolution layer, we change the order of those three operations to realize the inversion of convolution. The operations are unpooling, convolution and ReLU [6]. The diagram of unpooling operation is shown in Fig. 5, where we replace each entry of a feature map by a 2×2 block with the entry value in the top left corner and zeros elsewhere.

TABLE I

CONVOLUTION AND DECONVOLUTION FILTER AND POOLING SIZE

	layer 1	layer 2	layer 3	layer 4
Conv. kernel	9×9	5×5	5×5	9×9
Max/Un-pool size	2×2	2×2	2×2	2×2

The details about the sizes of convolution filters and pooling can be found in Tab. I. For the first and second layers, we operate maxpooling. And for the third and forth layers, unpooling is operated.

Another critical parameter of our network is the number of channels at different layers. To research on the influence of this parameter, we vary the numbers of the channels in the 3 middle stages (shown as 4-8-4 in Fig. 1) and see how it influences CNN features, shown in Fig. 6.

When channel number set is 4-8-4, we see that the CNN features extracted from 2 convolution layers are basically down-sampled images. And 2 out of 8 basically store no

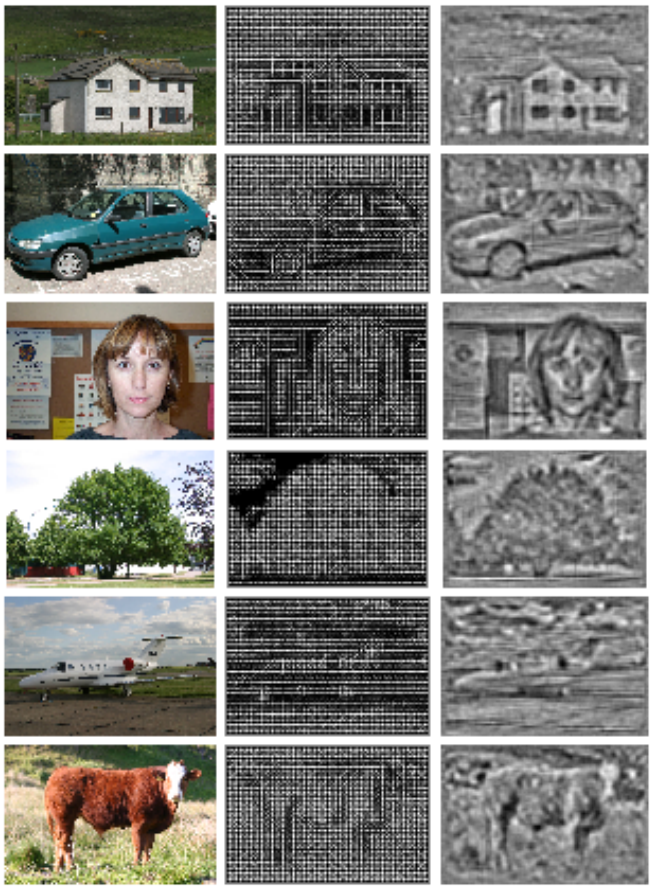

Fig. 4. We show results for inversions in all six categories using pairwise dictionary algorithm. left column: original images. middle column: HOG features. right column: HOG feature inversions.

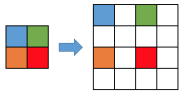

Fig. 5. Illustration of 2×2 unpooling operation used in the network

information of our original image. If we reduce the channel numbers to 4-4-4, we can see the results are quite similar. Hence we assume that the previous layer (layer 1) passes in too much information which is not needed for deCNN to realize image reconstruction.

However, if we reduce all the channel numbers by half to 2-4-2, then the extracted features become edge information rather than down-sampled image. And this situation will remain the same even if we increase the middle channel numbers to 8. Hence the number of channels in our network has a significant impact on the feature extraction.

The learning rate also affects the training speed and performance of our netowork. So we had a few attempts and the results are illustrated in Fig. 7.

After building our network, we trained the auto-encoder with different sets of the numbers of channels (4-8-4 and 2-4-2) and reconstruct images shown as follows in Fig. 8.

The reconstructed image from 4-8-4 is recovered using the

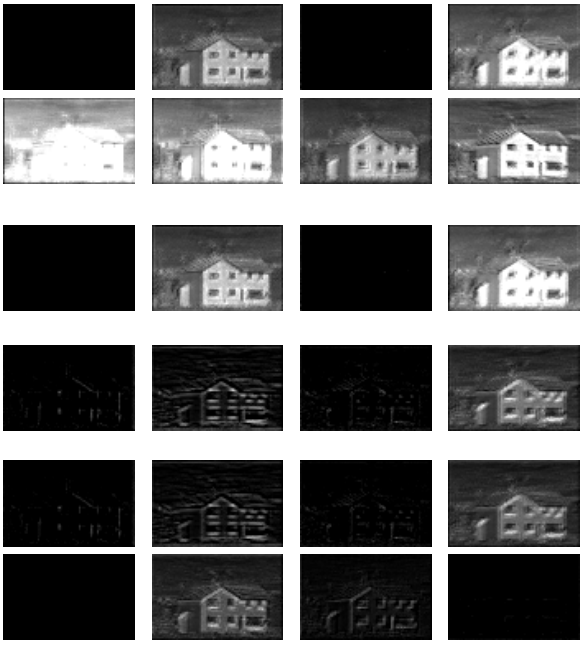

Fig. 6. CNN feature visualization (of different number of channels), from top to bottom: 1) 4-8-4, 2) 4-4-4, 3) 2-4-2, 4) 2-8-2

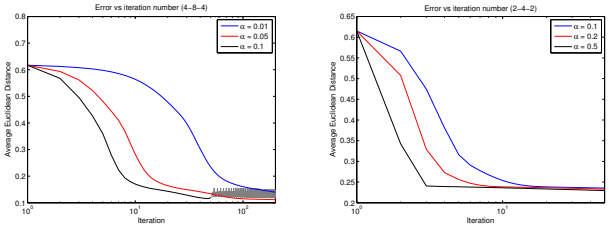

Fig. 7. Convergence rate versus iteration numbers over different learning rates

down-sampled feature while the image from 2-4-2 is built using edge feature. These results verify our former analysis of the CNN features.

V. Conclusion

In this project, we have implemented different algorithms, namely binary- and multi-class SVM, pairwise dictionary and CNN, to visualize object recognition features and provide more intuitive understandings of what happens in an object recognition system. It turns out that both of pairwise dictionary and CNN methods produce promising results in terms of visualization quality. While CNN is able to invert features with less errors, pairwise dictionary method tends to perform fast inversions without losing high level contents of the original image. For future work, it would be interesting to investigate methods to recover a color image from HOG features by possibly overlaying multiple layers of inversions over different color channels.

ACKNOWLEDGMENT

We sincerely acknowledge our classmates Shutong, Zhang and Yixin, Wang for their helps on setting up the CNN framework and thank teaching staffs of CS229 for bring us such a rewarding course.

Fig. 8. We show results for inversion in all six categories using CNN and deCNN. Left column: original images, middle column: channels 4-8-4, right images: channels 2-4-2.

TABLE II

Mean normalized cross correlation

category	Pairwise Dict	CNN	
		8-4-8	4-2-4
house	0.593	0.841	0.459
face	0.636	0.756	0.261
car	0.639	0.775	0.249
tree	0.618	0.734	0.150
cow	0.632	0.796	0.233
plane	0.634	0.878	0.260

References

[1] Microsoft Object Class Recognition image databases kernel description, 2005.

[2] F. Bastien, P. Lamblin, R. Pascanu, J. Bergstra, I. J. Goodfellow, A. Bergeron, N. Bouchard, and Y. Bengio. Theano: new features and speed improvements. Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop, 2012.

[3] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio. Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*, June 2010. Oral Presentation.

[4] B. Biggio, B. Nelson, and P. Laskov. Poisoning attacks against support vector machines. In *International Conference on Machine Learning, 2012*.

[5] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. *arXiv preprint arXiv:1310.1531*, 2013.

[6] A. Dosovitskiy, J. T. Springenberg, and T. Brox. Learning to generate chairs with convolutional neural networks. *arXiv preprint arXiv:1411.5928*, 2014.

[7] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results. http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html.

[8] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *arXiv:13112524*, 2013.

[9] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pages 580–587. IEEE, 2014.

[10] C.-W. Hsu and C.-J. Lin. A comparison of methods for multiclass support vector machines. *Neural Networks, IEEE Transactions on*, 13(2):415–425, 2002.

[11] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[12] J. Mairal, F. Bach, J. Ponce, and G. Sapiro. Online dictionary learning for sparse coding. In *International Conference on Machine Learning*, pages 689–696. IEEE, 2009.

[13] D. Parikh and C. L. Zitnick. The role of features, algorithms and data in visual recognition. In *Computer Vision and Pattern Recognition, 2010*, pages 2328 – 2335. IEEE.

[14] A. S. Razavian, H. Azizpour, J. Sullivan, and S. Carlsson. Cnn features off-the-shelf: an astounding baseline for recognition. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2014 IEEE Conference on*, pages 512–519. IEEE, 2014.

[15] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*, 2013.

[16] C. Vondrick, A. Khosla, T. Malisiewicz, and A. Torralba. Hoggles: Visualizing object detection features. In *International Conference on Computer Vision*, pages 1550–5499. IEEE.

[17] P. Weinzaepfel, H. Jegou, and P. Perez. Reconstructing an image from its local descriptors. In *Computer Vision and Pattern Recognition, 2011*, pages 337 – 344. IEEE.

[18] D. M. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *Computer Vision and Pattern Recognition, 2013*.

[19] M. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *arXiv:1311.2901*, 2013.

[20] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *Computer Vision–ECCV 2014*, pages 818–833. Springer, 2014.

[21] L. Zhang, H. Dibeklioglu, and L. van der Maaten. Speeding up tracking by ignoring features. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on 23-28 June 2014*, pages 1266 – 1273. IEEE, 2014.