Learning to See by Moving

Pulkit Agrawal  
UC Berkeley  
pulkitag@eecs.berkeley.edu

João Carreira  
UC Berkeley  
carreira@eecs.berkeley.edu

Jitendra Malik  
UC Berkeley  
malik@eecs.berkeley.edu

Abstract

The dominant paradigm for feature learning in computer vision relies on training neural networks for the task of object recognition using millions of hand labeled images. Is it possible to learn useful features for a diverse set of visual tasks using any other form of supervision? In biology, living organisms developed the ability of visual perception for the purpose of moving and acting in the world. Drawing inspiration from this observation, in this work we investigate if the awareness of egomotion can be used as a supervisory signal for feature learning. As opposed to the knowledge of class labels, information about egomotion is freely available to mobile agents. We show that given the same number of training images, features learnt using egomotion as supervision compare favourably to features learnt using class-label as supervision on visual tasks of scene recognition, object recognition, visual odometry and keypoint matching.

"We move in order to see and we see in order to move"  
J.J. Gibson

1. Introduction

Recent advances in computer vision have shown that visual features learnt by training neural networks for the task of object recognition using more than a million labelled images are useful for many computer vision tasks like semantic segmentation, object detection and action classification [22, 13, 17, 1, 38]. However, object recognition is one among many tasks for which vision is used. For example, humans use visual perception for recognizing objects, understanding spatial layouts of scenes and performing actions such as moving around in the world. Is there something special about the task of object recognition or is it the case that useful visual representations can be learnt through other modes of supervision? Clearly, biological agents perform complex visual tasks and it is unlikely that they require external supervision in form of millions of labelled examples. Unlabelled visual data is freely available and in theory this data can be used to learn useful visual representations. However, until now unsupervised learning approaches [6, 27, 34, 37] have not yet delivered on their promise and are nowhere to be seen in current applications on complex real world imagery.

It is believed that biological agents developed perceptual systems for obtaining sensory information about their environment that can aid them in performing actions required for accomplishing their goals [16, 12]. The motor system of both biological and robotic agents is responsible for executing these actions. The existence of close links between perceptual and motor systems in biological agents is supported by the discovery of neurons in mammalian brain that show increased neural activity not only when the animal itself performs a certain action, but also when the animal perceives another animal performing the same action (Mirror Neurons; [35]). Although motor theories of perception have a long history [7], there has been little work in formulating computational models of perception that make use of motor information. In this work we explore how agents can use information provided by their own motor system as an intrinsic source of supervision for learning useful visual features.

Useful visual features must possess two characteristics - (1) they should be adept at solving a variety of visual tasks and (2) they should enable the agent to perform novel target tasks by learning from minimal demonstrations of the same task performed by an external teacher. For example, if the agent is presented with only a few labeled examples of cars by an external teacher, the agent must be able to accurately identify cars in future images it views.

A motor signal that is readily available to mobile agents is the awareness of their egomotion (i.e. self-motion). For example, the vestibular system provides the sense of orientation in many mammals. In humans and many animals, the brain has access to information about movement of eyes, limbs and the actions that the animal performs [12]. A mobile robotic agent can estimate its egomotion either from the motor commands it issues to move or from odometry sensors like gyroscopes and accelerometers mounted on the robot itself. Stated differently, information about egomotion is available for “free” to mobile agents.
We hypothesize that agents can learn useful visual representations by performing the simple task of correlating their visual stimuli with their egomotion. As a mobile agent can be treated like a camera moving around in the world, the knowledge of egomotion is same as the knowledge of how the camera moves. Using this insight, we pose the problem of correlating visual stimuli with egomotion as the problem of predicting the camera transformation from the consequent pairs of images that the agent receives while it moves. Intuitively, the process of predicting camera transformation between two images should force the agent to learn features adept at identifying common visual elements present in both the images (i.e., visual correspondence). This means that egomotion can serve as supervision for learning features that are useful for determining visual correspondence. In the past, features such as SIFT, that were hand engineered for finding correspondences, were found to be very useful for tasks such as object recognition [28, 23]. This suggests that features learnt using egomotion may also be useful for such tasks.

In order to test our hypothesis, we trained multilayer neural networks to predict the camera transformation between pairs of images. As a proof of concept, we first demonstrate the usefulness of our approach on the MNIST dataset [26]. We show that if class-label supervision is available only for a limited number of examples, then the features learnt using our method outperform previous approaches of unsupervised feature learning (section 3.3). Next, we evaluated the efficacy of our approach on real world imagery. For this purpose, we used image and odometry data recorded from a car moving through urban scenes, made available as part of the KITTI [15] and the San Francisco (SF) city [10] datasets. This data mimics the scenario of a robotic agent moving around in the world. The quality of features learnt from this data were evaluated on four tasks - (1) Scene recognition on SUN [42] (section 5.1), (2) Visual odometry (section 5.3), (3) Keypoint matching (section 5.2) and (4) Object recognition on Imagenet [36] (section 5.4). Our results show that for the same number of labelled images, features learnt using egomotion as supervision compare favorably to features learnt using class-label as supervision. To the best of our knowledge, this work provides the first effective demonstration of learning visual representations from non-visual egomotion information in real world setting.

The rest of this paper is organized as following: In section 2 we discuss the related work, in section 3, 4, 5 we present the method, dataset details and we conclude with the discussion in section 6.

2. Related Work

Barlow [4, 3] hypothesized that good visual representations can be learnt by finding features that minimize the redundancy of visual information. This hypothesis inspired research in fully unsupervised learning from visual imagery and led to popular algorithms such as ICA [5, 20], autoencoders [24, 9], sparse coding [32], restricted boltzmann machines [27, 37] and others [6]. The aim of such approaches has been to discover compact and rich representations of images that are also sufficient to reconstruct these images. Another line of work has focused on learning features that are invariant to transformations irrelevant to the identity of objects, either from images [14, 34], or from video [41].

To the best of our knowledge, we are among the first to explore feature learning from non-visual egomotion information in the real world setting. Closest to our method is the work of transforming auto-encoders [19] that used non-visual egomotion to reconstruct the transformed image from an input source image. This work was purely conceptual in nature and the quality of learned features was not evaluated. In contrast, our method uses egomotion as supervision by predicting the transformation between two images using a siamese-like network model [11]. Another piece of work by [29] considers modeling spatial transformations using boltzmann machines but did not assess the utility learnt features for performing visual tasks.

In the past, there has been little work on using intrinsic reward signals available in robotic agents for feature learning. [18] proposed using intrinsic reward signals from the robots to learn a system for predicting traversability of paths while the robot explores its environment. [33] trained neural networks for driving vehicles directly from visual input. The recent work of [30] considered joint learning of visual and motor representations by training a system for playing Atari video games directly from image pixels, but their method made use of extrinsic rewards in form of game scores. The goal of these works has been to develop a system for performing a specific task. In contrast, the motivation of our work is to investigate how useful are the visual features learnt using egomotion for a variety of visual tasks.

3. A Simple Model of Motion-based Learning

When a mobile agent equipped with visual sensors moves from a point A to a nearby point B in its environment (i.e., performs egomotion), it receives two views of the world as its visual inputs. The visual system of the agent is modelled by a randomly initialized Convolutional Neural Network (CNN, [25]). By trying to perform the task of predicting its own egomotion using its visual inputs, the agent optimizes its own visual representations (i.e. optimizing the weights of CNN). As the problem of predicting the egomotion is equivalent to the problem of predicting camera transformation between the images, this task can be modelled as training a Siamese style Convolutional Neural Network (SCNN) [11] (illustrated in figure 2) for predicting the camera transformation between the pair of input images. The
Figure 1: Exploring the utility of egomotion as supervision for learning useful visual features. A mobile agent equipped with visual sensors receives a sequence of images as inputs while it moves in its environment. The movement of the agent is equivalent to the movement of a camera. In this work, egomotion based learning is posed as the problem of learning features that can predict the camera transformation from the consequent pairs of images. The top and bottom rows of the figure show some sample image pairs from the SF and KITTI datasets that were used for feature learning.

The process of optimizing the visual representations (i.e. feature learning) has been referred to as pretraining in rest of this paper.

3.1. SCNN, BCNN and TCNN

The SCNN is a two streamed CNN, where each stream independently computes features for one image. The two streams of the SCNN share the same architecture and the same set of weights and consequently perform the same set of operations for computing features. We call each of these streams as a Base-CNN (BCNN). Features from the two BCNNs are concatenated and passed downstream into another CNN called as the Top-CNN (TCNN). The TCNN is responsible for using the BCNN features to predict the camera transformation between the input pair of images.

For evaluating the quality of learnt features, the TCNN is removed and a single BCNN is used as a standard CNN for feature computation. For the target tasks of digit classification, scene recognition and keypoint matching, the pre-trained model weights were not modified and were used “as is” (i.e. no finetuning) for feature computation. For the target tasks of object recognition and visual odometry, all layers of the BCNN were finetuned for the target tasks.

3.2. Shorthand for CNN architectures

The abbreviations Ck, Fk, P, D, Op represent a convolutional(C) layer with k filters, a fully-connected(F) layer with k filters, pooling(P), dropout(D) and the output(Op) layers respectively. We used ReLU non-linearity after every convolutional/fully-connected layer, except for the output layer. The dropout layer was always used with dropout of 0.5. The output layer was a fully connected layer with number of units equal to the number of desired outputs. As an example of our notation, C96-P-F500-D refers to a network with 96 filters in the convolution layer followed by ReLU non-linearity, a pooling layer, a fully-connected layer with 500 unit, ReLU non-linearity and a dropout layer. We used [21] for training all our models.

3.3. Proof of Concept using MNIST

MNIST was used for providing a proof of concept because several methods for unsupervised feature learning have been evaluated on this dataset. In the MNIST setting, egomotion of the agent was emulated by generating synthetic data consisting of random transformation (translations and rotations) of digit images. From the training set of 60K images, digits were randomly sampled and then transformed using two different sets of random transformations to generate image pairs. CNNs were trained for predicting the transformations between these image pairs. In section 3.3.3 we present the results.

3.3.1 Data

The relative translation between the digits was constrained to be an integer value in the range [-3, 3]. The relative rotation \( \theta \) was constrained to lie within the range \([-30^\circ, 30^\circ]\). The prediction of transformation was posed as a classification task with three separate soft-max losses (one each for translation along X, Y axes and the rotation about Z-axis). SCNN was trained to minimize the sum of these three losses. Translations along X, Y were separately binned into seven uniformly spaced bins each. The rotations were
Visual features are learnt by training a Siamese style Convolutional Neural Network (SCNN, [11]) that takes as inputs two images and predicts the transformation between the images (i.e. egomotion). Each stream of the SCNN (called as Base-CNN or BCNN) computes features for one image. The outputs of two BCNNs are concatenated and passed as inputs to a second multilayer CNN called as the Top-CNN (TCNN) (shown as layers $F_1$, $F_2$). The two BCNNs have the same architecture and share weights. After feature learning, TCNN is discarded and a single BCNN stream is used as a standard CNN for extracting features for performing target tasks like scene recognition.

3.3.2 Network Architectures

The BCNN architectures used in this work are detailed in table 1. The two BCNN streams were concatenated using this TCNN architecture: $F1000-D-Op$. SCNN was pretrained for 40K iterations (i.e. 5M examples) using an initial learning rate of 0.01 which was reduced by a factor of 2 after every 10K iterations.

The following architecture was used for classification: $BCNN-F500-D-Op$. In order to evaluate the quality of BCNN features, the learning rate of all layers in the BCNN was set to 0 when training for digit classification. The network was trained for 4K iterations (which is equivalent to training for 50 epochs for 10K labelled training examples) with a constant learning rate of 0.01.

3.3.3 Results

The BCNN features were evaluated by computing error rates on the task of digit classification using 100, 300, 1K and 10K class-labelled examples for training. These sets were constructed by randomly sampling digits from the standard training set of 60K digits. For this part of the experiment, the original digit images were used (i.e. without any transformations or data augmentation). The standard test set of 10K digits was used for evaluation and error rates averaged across 3 independent runs are reported in table 1. Our best performing architecture ($C96-P-C256-P$), outperforms convolutional deep belief networks [27], a previous approach based on learning features invariant to transformations [34] and features learnt by the default autoencoder network provided with Caffe[21]. These results demonstrate that predicting relative transformation (i.e. egomotion) is a useful mode of supervision for learning visual features adept at digit classification.

### Table 1: Comparing the performance of various pretraining methods used for feature learning on MNIST. The performance is reported as the error rate. The results demonstrate that egomotion based feature learning outperforms previously proposed approaches of unsupervised feature learning on MNIST.

<table>
<thead>
<tr>
<th>Method</th>
<th>Train from Scratch</th>
<th>Finetune after Pretrain</th>
</tr>
</thead>
<tbody>
<tr>
<td>#Train</td>
<td>100 300 1000 10000</td>
<td>100 300 1000 10000</td>
</tr>
<tr>
<td>Ranzato et al.</td>
<td>23.53 13.09 7.47 2.72</td>
<td>23.59 13.10 7.93 2.58</td>
</tr>
<tr>
<td>Lee et al. [27]</td>
<td>- 7.18 3.21 0.85</td>
<td>- 2.62</td>
</tr>
<tr>
<td>C96</td>
<td>23.81 14.03 8.49 2.85</td>
<td>17.95 7.88 4.41 1.79</td>
</tr>
<tr>
<td>C96-C256</td>
<td>24.00 13.01 7.16 2.64</td>
<td>12.12 5.67 3.17 1.52</td>
</tr>
<tr>
<td>C96-C256-C256</td>
<td>27.04 14.00 8.38 36.90</td>
<td>13.56 5.90 3.39 1.41</td>
</tr>
<tr>
<td>C96-P</td>
<td>21.09 10.34 5.01 1.48</td>
<td>18.69 8.41 4.65 1.73</td>
</tr>
<tr>
<td>C96-P-C256-P</td>
<td>24.15 13.42 7.96 3.28</td>
<td>8.66 3.61 2.01 0.93</td>
</tr>
<tr>
<td>C96-P-C256-P-C256-P</td>
<td>27.04 13.00 7.16 2.64</td>
<td>16.70 7.73 4.35 3.55</td>
</tr>
</tbody>
</table>

4. Learning Visual Features From Egomotion in Natural Environments

We used two main sources of real world data for feature learning: the KITTI and SF datasets, which were collected using cameras and non-visual odometry sensors mounted on a car driving through urban scenes. Details about the data, the experimental procedure, the network architectures and the results are provided in sections 4.1, 4.2, 4.3 and 5 respectively.

4.1. KITTI Dataset

The KITTI dataset provided odometry and image data recorded during 11 short trips of variable length made by a car moving through urban landscapes. The total number of frames in the entire dataset was 23,201. Out of 11, 9 sequences were used for training and 2 for validation. The total number of images in the training set was 20,501.

The odometry data was used to compute the camera transformation between pairs of images recorded from the
car. As significant camera transformations in the KITTI data were either due to translations along the Z/X axis or rotation about the Y axis, only these three dimensions were used to express the camera transformation. The rotation was represented by euler angles. The task of predicting the transformation between pair of images was posed as a classification problem. The three dimensions of camera transformation were individually binned into 20 uniformly spaced bins each. As it was unreasonable to expect that visual features can be used to infer big camera transformations, the training image pairs were chosen from images that were almost ±7 frames apart.

The SCNN was trained to predict camera transformation from pairs of 227 × 227 pixel sized image regions. Each image was of overall size 370 × 1226 pixels and the coordinates for cropping image regions were randomly chosen for each pair of images. Figure 1 illustrates typical image crops.

4.2. SF Dataset

SF dataset provides camera transformation between ~ 136K pairs of images (constructed from a set of 17,357 unique images). The images and the camera transformation were obtained from Google StreetView data [10]. ~ 130K image pairs were used for training and ~ 6K pairs for validation.

Just like KITTI, the task of predicting camera transformation was posed as a classification problem. Unlike KITTI, significant camera transformation was found along all six dimensions of transformation (i.e. the 3 euler angles and the 3 translations). As, it was unreasonable to expect that visual features can be used to infer big camera transformations, rotations between [-30˚, 30˚] were binned into 10 uniformly spaced bins and two extra bins were used for rotations larger and smaller than 30˚ and -30˚ respectively. The three translations were individually binned into 10 uniformly spaced bins each. Images were resized to a size of 360 × 480 and randomly cropped image regions of size 227 × 227 were used for training the SCNN.

4.3. Network Architecture

The BCNN architecture closely followed the architecture of first five AlexNet layers [22]: C96-P-C256-P-C384-C384-C256-P. The TCNN architecture was: C256-C128-F500-D-Op. The spatial extent of convolutional filters in the TCNN was 3 × 3 for all the layers. The networks were trained for 60K iterations with a batch size of 128. The initial learning rate was set to 0.001 and was reduced by a factor of two after every 20K iterations.

In the remainder of the paper, the networks pretrained on KITTI and SF datasets are called KITTI-Net and SF-Net respectively. Figure 3 shows the layer-1 filters of KITTI-Net and SF-Net. A large majority of layer-1 filters are color detectors, while some of them are edge detectors. As color is a useful cue for determining correspondences between closely frames of a video sequence, learning of color detectors as layer-1 filters is not surprising. The fraction of filters that detect edges is higher for the SF-Net as compared to the KITTI-Net. One possible explanation for this is the fact that a higher fraction of images in the SF dataset contain structured objects like buildings and cars.

5. Evaluating Egomotion Based Learning

For evaluating the merits of the proposed approach, features learned using egomotion based supervision were compared against features learned using class-label based supervision on the challenging tasks of scene recognition, intra-class keypoint matching and visual odometry. In addition, the usefulness of egomotion based pretraining for the task of object recognition was also evaluated. The ultimate goal of feature learning is to find features that can generalize from only a few supervised examples on a new task that the agent needs to perform. Therefore it makes sense to evaluate the quality of features when only a few labelled examples for the target task are provided. Consequently, the scene and object recognition experiments were performed in the setting when only 1-50 labelled examples per class were available for training.

The KITTI-Net and SF-Net (examples of models trained using egomotion based supervision) were trained using only ~ 20K unique images. To make a fair comparison with class-label based supervision, a model with AlexNet architecture was trained using only 20K images taken from the training set of ILSVRC12 challenge (i.e. 20 examples per class). This model has been referred to as AlexNet-20K. In addition, some experiments presented in this work also make comparison with AlexNet models trained with 100K and 1M images that have been named as AlexNet-100K and AlexNet-1M respectively.
5.1. Scene Recognition

We used the SUN dataset for evaluating scene recognition performance. This dataset consists of a total of 397 indoor and outdoor scene categories and provides 10 standard training splits constructed using 1, 5, 10, 20 and 50 training images per class and associated test set splits containing 50 images per class. Due to computational constraints, recognition accuracy was evaluated only for the settings of 5 and 20 labelled examples per class and 3 train/test splits.

For evaluating the utility of CNN features produced by different layers, separate linear (SoftMax) classifiers were trained on features produced by individual CNN layers (i.e. BCNN layers for KITTI-Net and SF-Net). Table 2 reports recognition accuracy (averaged over 3 train/test splits) for various networks considered in this study. The performance of KITTI-Net is superior to SF-Net and comparable to AlexNet-20K. This indicates that given a fixed budget of pretraining images, egomotion based supervision learns features that are almost as good as the features using class-based supervision on the task of scene recognition.

The KITTI-Net outperforms hand-engineered GIST [31] features developed for scene classification, but is outperformed by Dense SIFT with spatial pyramid matching (SPM) kernel [23]. The KITTI-Net was trained using limited visual data (≈ 20K frames) containing visual imagery of limited diversity. The KITTI data majorly contains images of roads, buildings, cars, few pedestrians, trees and some vegetation. It is in fact surprising that a network trained on data with such little diversity is competitive on classifying indoor and outdoor scenes with the AlexNet-20K that was trained on a much more diverse set of images. We hope that with more diverse training data for egomotion based learning, the performance of learnt features will be superior than currently reported numbers.

The KITTI-Net outperforms the SF-Net except when the performance of first layer (L1, conv-1 layer of AlexNet architecture) is compared. From figure 3 notice that layer 1 of SF-Net learns a greater number of edge filters. This may explain why the performance of L1 for SF-Net is higher. The KITTI-Net performed better than SF-Net possibly because a greater number of 227 × 227 image region pairs were available for egomotion based training from the KITTI dataset (see section 4.1, 4.2). As KITTI-Net was found to be superior to the SF-Net in this experiment, the KITTI-Net was used for all other experiments described in this paper.

5.2. Intra-Class Keypoint Matching

Identifying the same part (or keypoint) of an object across different instances of the same object class (for instance, identifying “eyes” among different instances of a dog) is an important visual task. Visual features learned by motion based and class-label based supervision were evaluated for this task using the keypoint annotations on the PASCAL VOC 2012 dataset [8].

The keypoint matching was calculated in the following way: First, ground-truth object bounding boxes (GT-BBOX) from PASCAL-VOC2012 dataset were extracted and resized (while preserving the aspect ratio) to ensure that the smaller side of the boxes was of length 227 pixels. Next, feature maps from layers 2-5 of AlexNet-10K, AlexNet-100K, AlexNet-1M and KITTI-Net were computed for every GT-BBOX. The keypoint matching score was computed between all pairs of GT-BBOX belonging to the same object class. For given pair of GT-BBOX, the features associated with keypoints in the first image were used to predict the location of the same keypoints in the second image. The normalized pixel distance between the actual and predicted keypoint locations has been referred to as error in the keypoint matching. More details about this procedure have been provided in appendix A.

Intuitively, matching keypoints across object instances that are related by a large transformation (i.e. viewpoint distance) should be harder than matching keypoints across instances with a small viewpoint distance. In order to obtain a holistic understanding of the quality of features learnt using different modes of supervision on this task, matching error was computed as a function of viewpoint distance [40]. Figure 4 reports the matching error averaged across all keypoints, all pairs of GT-BBOX and all classes using features extracted from layers conv-3 and conv-4. Matching error for layers 2 and 5 have been provided in appendix A.4

The KITTI-Net, trained only with 20K unique frames was superior to AlexNet-20K and AlexNet-100K and inferior only to AlexNet-1M. A network with AlexNet architecture initialized with random weights (Alexnet-Rand), surprisingly performed better than AlexNet-20K. One possible explanation for this observation is that with only 20K examples, features learnt by AlexNet-20K only capture coarse global appearance of objects and are therefore poor at keypoint matching. SIFT has been hand engineered for finding correspondences across images and performs as well as the best Alexnet-1M features for this task (i.e. conv-4 features). See figure 5 for a visualization of the keypoint matching results.

The results of this experiment indicate that egomotion based supervision is superior to class-label based supervision for learning features adept at keypoint matching. These results further support the intuition that different kinds of supervision can provide features that are useful for different tasks.

5.3. Visual Odometry

Visual odometry is the task of predicting the egomotion (or camera transformation) between pairs of input images. The performance of SCNNs formed by using KITTI-Net,
Table 2: Comparing the accuracy of neural networks pre-trained using motion-based and class-label based supervision for the task of scene recognition on the SUN dataset. The performance of layers 1-6 (labelled as L1-L6) of these networks was evaluated after finetuning the network using 5/20 images per class from the SUN dataset. The performance of the KITTI-Net (i.e. motion-based pretraining) fares favorably with a network pretrained on Imagenet (i.e. class-based pretraining) with the same number of pretraining images (i.e. 20K).

<table>
<thead>
<tr>
<th>Method</th>
<th>Pretrain Supervision</th>
<th>#Pretrain</th>
<th>#Finetune</th>
<th>L1</th>
<th>L2</th>
<th>L3</th>
<th>L4</th>
<th>L5</th>
<th>L6</th>
<th>L1</th>
<th>L2</th>
<th>L3</th>
<th>L4</th>
<th>L5</th>
<th>L6</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlexNet-1M</td>
<td>Class-Label</td>
<td>1M</td>
<td>5</td>
<td>5.3</td>
<td>10.5</td>
<td>12.1</td>
<td>12.5</td>
<td>18.0</td>
<td>23.6</td>
<td>20</td>
<td>11.8</td>
<td>22.2</td>
<td>25.0</td>
<td>26.8</td>
<td>33.3</td>
</tr>
<tr>
<td>AlexNet-20K</td>
<td></td>
<td>20K</td>
<td>5</td>
<td>4.9</td>
<td>6.3</td>
<td>6.6</td>
<td>6.3</td>
<td>6.6</td>
<td>6.7</td>
<td>20</td>
<td>8.7</td>
<td>12.6</td>
<td>12.4</td>
<td>11.9</td>
<td>12.5</td>
</tr>
<tr>
<td>SF-Net</td>
<td>Motion</td>
<td>18K</td>
<td>5</td>
<td>4.4</td>
<td>5.2</td>
<td>4.9</td>
<td>5.1</td>
<td>4.7</td>
<td>-</td>
<td>-</td>
<td>8.6</td>
<td>11.6</td>
<td>10.9</td>
<td>10.4</td>
<td>9.1</td>
</tr>
<tr>
<td>KITTI-Net</td>
<td>Motion</td>
<td>20.5K</td>
<td>5</td>
<td>4.3</td>
<td>6.0</td>
<td>5.9</td>
<td>5.8</td>
<td>6.4</td>
<td>-</td>
<td>-</td>
<td>7.9</td>
<td>12.2</td>
<td>12.1</td>
<td>11.7</td>
<td>12.4</td>
</tr>
<tr>
<td>GIST [42]</td>
<td>Human</td>
<td>-</td>
<td>5</td>
<td>6.2</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>20</td>
<td>11.6</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>SPM [42]</td>
<td>Human</td>
<td>-</td>
<td>5</td>
<td>8.4</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>20</td>
<td>16.0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 3: Comparing the accuracy of motion-based pretraining with class-label based pretraining on the task of visual odometry.

<table>
<thead>
<tr>
<th>Method</th>
<th>Translation Acc.</th>
<th>Rotation Acc.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>δX</td>
<td>δY</td>
</tr>
<tr>
<td>Train from Scratch</td>
<td>40.2</td>
<td>58.2</td>
</tr>
<tr>
<td>KITTI-Net</td>
<td>43.4</td>
<td>57.9</td>
</tr>
<tr>
<td>AlexNet-1M</td>
<td>41.8</td>
<td>58.0</td>
</tr>
</tbody>
</table>

AlexNet-1M and a network with AlexNet architecture initialized with random weights as BCNNs were compared on the task of predicting camera transformations between image pairs taken from the validation set of the SF dataset. All layers of the SCNNs were finetuned for 25K iterations using image pairs from the training set of SF dataset.

Table 3 reports the performance of different networks. The KITTI-Net was found to be either superior or as good as the AlexNet-1M on the task of visual odometry prediction. This indicates that on this task, the features learnt using motion-based supervision using only 20K images are as good as features learnt using class-label based supervision using 1M images.

5.4. Object Recognition

If it is the case that motion-based pretraining learns useful features for object recognition, a network initialized with weights from KITTI-Net should outperform a network initialized with random weights on the task of object recognition. For testing this, four different networks (initialized with random weights) were trained using 1, 5, 10 and 20 images per class from the ILSVRC-2012 challenge. As this dataset contains 1000 classes, the total number of training examples available for training for these networks were 1K, 5K, 10K and 20K respectively. Four more networks, each initialized from KITTI-Net weights were also finetuned using the same number of examples. The results of this experiment are presented in table 4. KITTI-Net clearly outperforms the networks trained from scratch. As expected, the improvement offered by motion-based pretraining is larger when the number of examples provided for the target task are fewer. This result shows that motion-based pretraining learns features that are useful for object recognition.

6. Discussion

In this work, we have shown that egomotion is a useful source of intrinsic supervision for visual feature learning in mobile agents. In contrast to class labels, knowledge of egomotion is "freely" available. On MNIST, egomotion-based feature learning outperforms previous unsupervised methods of feature learning. Given the same budget of pretraining images, on task of scene recognition, egomotion-based learning performs almost as well as class-label-based learning. Further, egomotion based features outperform features learnt by a CNN trained using class-label supervision on two orders of magnitude more data (AlexNet-1M) on the task of visual odometry and one order of magnitude more data on the task of intra-class keypoint matching. In addition to demonstrating the utility of egomotion based supervision, these results also suggest that features learnt by
AlexNet trained for object recognition may not be generic for a wide variety of visual tasks. This means that future work should look at what kinds of supervision are useful for feature learning.

One potential criticism of our work is that we have trained and evaluated high capacity deep models on relatively little data (e.g. only 20K unique images available on the KITTI dataset). In theory, we could have learnt better features by downsizing these networks. For example, in our experiments with MNIST we found that pretraining a 2-layer network instead of 3-layer results in better performance (table 1). In this work, we have made a conscious choice of using standard deep models because the main goal of this work was not to explore novel feature extraction architectures but to investigate the value of egomotion for learning visual representations on architectures known to perform well on practical applications and previous work [1] has shown that such networks outperform conventional visual features when trained using only about \( \sim 10K \) images. While egomotion is freely available to mobile agents, there are currently no publicly available datasets as large as ImageNet. Consequently, we were unable to evaluate the utility of motion-based supervision across the full spectrum of training set sizes. Future research focused on exploring architectures that are better suited for egomotion based learning and more nuanced learning formulations for using egomotion as supervision can only make a stronger case for this line of work.

In this work, we chose to first pretrain our models using a base task (i.e. egomotion) and then finetune these models for target tasks. A more interesting setting is that of online learning where the agent has continuous access to intrinsic supervision (such as egomotion) and occasional explicit access to extrinsic teacher signal (such as the class labels). We believe that such a training procedure is likely to result in learning of better features. Our intuition behind this is that seeing different views of the same instance of an object (say) car, may not be sufficient to learn that different instances of the car class should be grouped together. The occasional extrinsic signal about class labels may prove useful for the agent to learn such concepts.

Another worthwhile direction of future research is to revisit active vision [2]. Our current work makes use of passively collected egomotion data and it would be interesting to investigate if the agent is able to construct better visual representations if it is allowed to decide on how it explores its environment.

**Acknowledgements**

This work was supported in part by ONR MURI-N00014-14-1-0671. Pulkit Agrawal was partially supported by Fulbright Science and Technology Fellowship. João Carreira was supported by the Portuguese Science Foundation, FCT, under grant SFRH/BPD/84194/2012. We gratefully acknowledge NVIDIA corporation for the donation of Tesla GPUs for this research.

**Appendix**

**A. Keypoint Matching Score**

Consider images of two instances of the same object class (for example airplane images as shown in first row of figure 5) for which keypoint matching score needs to be computed.

The images are pre-processed in the following way:

- Crop the groundtruth bounding box from the image.
• Pad the images by 30 pixels along each dimension.
• Resize each image so that the smallest side is 227 pixels. The aspect ratio of the image is preserved.

A.1. Keypoint Matching using CNN

Assume that the $l^{th}$ layer of the CNN is used for feature computation. The feature map produced by the $l^{th}$ layer is of dimensionality $I \times J \times M$, where $(I,J)$ are the spatial dimensions and $M$ is the number of filters in the $l^{th}$ layer. Thus, the $l^{th}$ layer produces a $M$ dimensional feature vector for each of the $I \times J$ grid position in the feature map.

The coordinates of the keypoints are provided in the image coordinate system [8]. For the keypoints in the first image, we first determine their grid position in the $I \times J$ feature map. Each grid position has an associated receptive field in the image. The keypoints are assigned to the grid positions for which the center of receptive field is closest to the keypoints. This means that each keypoint is assigned one location in the feature map.

Let the $M$ dimensional feature vector associated with the $k^{th}$ keypoint in the first image be $F^k_{ij}$. Let the $M$ dimensional feature vector at grid location $C_{ij}$ for the second image be $F_2(C_{ij})$. The location of matching keypoint in the second image is determined by solving:

$$C^*_s = \underset{C_{ij}}{\text{argmin}} \left\| F^k_{ij} - F_2(C_{ij}) \right\|_2$$

(1)

$C^*_s$ is transformed into the image coordinate system by computing the center of receptive field (in the image) associated with this grid position. Let this transformed coordinates be $C^*_{im}$ and the coordinates of the corresponding keypoint (in the second image) be $C^*_{gtm}$. The matching error for the $k^{th}$ keypoint ($E_k$) is defined as:

$$E_k = \left\| C^*_{im} - C^*_{gtm} \right\|_2$$

(2)

where, $L^2_{ij}$ is the length of diagonal (in pixels) of the second image. As different images have different sizes, dividing by $L^2_{ij}$ normalizes for the difference in sizes. The matching error for a pair of images of instances belonging to the same class is calculated as:

$$E_{instance} = \frac{\sum_{k=1}^{K} E_k}{K}$$

(3)

The average matching error across all pairs of the instance of the same class is given by $E_{class}$:

$$E_{class} = \frac{\sum_{\text{instances}} E_{instance}}{\#\text{pairs}}$$

(4)

where, $\#\text{pairs}$ is the number of pairs of object instances belonging to the same class. In Figure 4 of the main paper we report the matching error averaged across all the 20 classes.

A.2. Keypoint Matching using SIFT

SIFT features are extracted using a square window of size 72 pixels and a stride of 8 pixels using the open source code from [39]. The stride of 8 pixels was chosen to have a fair comparison with the ConvNet features. The ConvNet features were computed with a stride of 8 for layer conv-2 and stride of 16 for layers conv-3, conv-4 and conv-5 respectively. The matching error using SIFT was calculated in the same way as for the CNNs.

A.3. Effect of Viewpoint on Keypoint Matching

Intuitively, matching instances of the same object that are related by a large transformation (i.e. viewpoint distance) should be harder than matching instances with a small viewpoint distance. Therefore, in order to obtain a holistic understanding of the accuracy of features in performing keypoint matching it is instructive to study the accuracy of matching as a function of viewpoint distance.

[40] aligned instances of the same class (from PASCAL-VOC-2012) in a global coordinate system and provide a rotation matrix ($R$) for each instance in the class. To measure the viewpoint distance, we computed the riemannian metric on the manifold of rotation matrices $||\log(R_i,R_j^T)||_F$, where $\log$ is the matrix logarithm, $||||_F$ is the Frobenius norm of the matrix and $R_i,R_j$ are the rotation matrices for the $i^{th},j^{th}$ instances respectively. We binned the distances into 10 uniform bins (of 18°each). In Figure 4 of the main paper we show the mean error in keypoint matching in each of these viewpoints bin. The matching error in the $k^{th}$ bin is calculated by considering all the instances with a viewpoint distance $\leq k \times 18^\circ$, for $k \in [1,10]$. As expected we find that keypoint matching is worse for larger viewpoint distances.

A.4. Matching Error for layers 2 and 5

The matching errors using features computed from layers 2 and 5 of different networks are shown in figure 6.

References

Figure 5: Example matchings between pairs of objects (randomly chosen) with viewpoints within 60 degrees of each other, for classes “aeroplane”, “bottle”, “dog”, “person” and “tvmonitor” from PASCAL VOC. The matchings have been shown for features from layer conv-4 of AlexNet-20K, AlexNet-100K, AlexNet-1M, KittiNet-20K and SIFT. The left image shows the ground truth keypoints that were matched with the keypoints in the right image. Right images shows the location of the ground truth keypoint (shown by solid dot) and lines joining the predicted keypoint location (tip of the line) with the ground keypoint location. Please see section A for details of keypoint matching procedure and figure 4 in the main paper for numerical results. This figure is best seen in color and with zoom.
Figure 6: Intra-class keypoint matching error as a function of viewpoint distance averaged over 20 PASCAL objects using features extracted from layers pool-2 (left) and conv-5 (right) of various networks used in this work. Please see section 5.2 for more details.


Abstract

Is strong supervision necessary for learning a good visual representation? Do we really need millions of semantically-labeled images to train a ConvNet? In this paper, we present a simple yet surprisingly powerful approach for unsupervised learning of ConvNets. Specifically, we use hundreds of thousands of unlabeled videos from the web to learn visual representations. Our key idea is that we track millions of patches in these videos. Visual tracking provides the key supervision. That is, two patches connected by a track should have similar visual representation in deep feature space since they probably belong to same object or object part. We design a Siamese-triplet network with a ranking loss function to train this ConvNet representation. Without using a single image from ImageNet, just using 100K unlabeled videos and the VOC 2012 dataset, we train an ensemble of unsupervised networks that achieves 52% mAP (no bounding box regression). This performance comes tantalizingly close to its ImageNet-supervised counterpart, an ensemble which achieves a mAP of 54.4%. We also show that our unsupervised network can perform competitive in other tasks such as surface-normal estimation.

Figure 1. Overview of our approach. (a) Given unlabeled videos, we perform unsupervised tracking on the patches in them. (b) Triplets of patches including query patch in the initial frame of tracking, tracked patch in the last frame, and random patch from other videos are fed into our siamese-triplet network for training. (c) The learning objective: Distance between the query and tracked patch in feature space should be smaller than the distance between query and random patches.

1. Introduction

What is a good visual representation and how can we learn it? At the start of this decade, most computer vision research focused on “what” and used hand-defined features such as SIFT [29] and HOG [5] as the underlying visual representation. Learning was often the last step where these low-level feature representations were mapped to semantic/3D/functional categories. However, the last three years have seen the resurgence of learning visual representations directly from pixels themselves using the deep learning and ConvNets [25, 21, 20]. At the heart of ConvNets is a completely supervised learning paradigm. Often millions of examples are first labeled using Mechanical Turk followed by data augmentation to create tens of millions of training instances. ConvNets are then trained using gradient descent and back propagation. But one question still remains: is strong-supervision necessary for training these ConvNets? Do we really need millions of semantically-labeled images to learn a good visual representation? It seems humans can learn visual representations using little or no semantic supervision but our current learning approaches still remain completely supervised.

In this paper, we explore the alternative: how can we exploit the unlabeled visual data on the web to train ConvNets (e.g. AlexNet [21])? In the past, there have been several attempts at unsupervised learning using millions of static images [23, 41] or frames extracted from videos [50, 44, 31]. The most common architecture used is an auto-encoder which learns representations based on its ability to recon-
struct the input images [32, 3, 45, 33]. While these approaches have been able to automatically learn V1-like filters given unlabeled data, they are still far away from supervised approaches on tasks such as object detection. So, what is the missing link? We argue that static images themselves might not have enough information to learn a good visual representation. But what about videos? Do they have enough information to learn visual representations? In fact, humans also learn their visual representations not from millions of static images but years of dynamic sensory inputs. Can we have similar learning capabilities for ConvNets?

We present a simple yet surprisingly powerful approach for unsupervised learning of ConvNets using hundreds and thousands of unlabeled videos from the web. Visual tracking is one of the first capabilities that develops in infants and often before semantic representations are learned\(^1\). Taking a leaf from this observation, we propose to exploit visual tracking for learning ConvNets in an unsupervised manner. Specifically, we track millions of “moving” patches in hundreds of thousands of videos. Our key idea is that two patches connected by a track should have similar visual representation in deep feature space since they probably belong to the same object. We design a Siamese-triplet network with ranking loss function to train the ConvNet representation. This ranking loss function enforces that in the final deep feature space the first frame patch should be much closer to the tracked patch than any other randomly sampled patch. We demonstrate the strength of our learning algorithm using extensive experimental evaluation. Without using a single image from ImageNet [34], just using 100K unlabeled videos and VOC 2012 dataset, we train an ensemble of AlexNet networks that achieves 52% mAP (no bounding box regression). This performance is similar to its ImageNet-supervised counterpart, an ensemble which achieves a mAP of 54.4%. We also show that our network trained using unlabeled videos also achieves similar performance to its completely supervised counterpart on other tasks such as surface normal estimation. To the best of our knowledge, the results reported in this paper come closest to standard supervised ConvNets approaches (which use millions of semantically-labeled images) among unsupervised approaches.

2. Related Work

Unsupervised learning of visual representations has a rich and diverse history starting from original auto-encoders work of Olhausen and Field [32] and early generative models. Most of the work in the area of unsupervised learning can be broadly divided into three categories. The first class of unsupervised learning algorithms focus on learning generative models with strong priors [17, 49, 42]. These algorithms essentially capture co-occurrence statistics of features. The second class of algorithms use manually defined features such as SIFT or HOG and perform clustering over training data to discover semantic classes [39, 35]. Some of these recent algorithms also focus on learning mid-level representations rather than discovering semantic classes themselves [38, 6, 7].

The third class of algorithms and more related to our paper is unsupervised learning of visual representations from the pixels themselves using deep learning approaches [18, 23, 41, 36, 26, 43, 8, 30, 2, 45]. Starting from the seminal work of Olhausen and Field [32], the goal is to learn visual representations which are (a) sparse and (b) reconstructive. Olhausen and Field [32] showed that using this criteria they can learn V1-like filters directly from the data. However, this work only focused on learning a single layer. This idea was extended by Hinton and Salakhutdinov [18] to train a deep belief network in an unsupervised manner via stacking layer-by-layer RBMs. Similar to this, Bengio et al. [3] investigated stacking of both RBMs and autoencoders. As a next step, Le et al. [23] scaled up the learning of multi-layer autoencoder on large-scale unlabeled data. They demonstrated that although the network is trained in an unsupervised manner, the neurons in high layers can still have high responses on semantic objects such as human heads and cat faces. Sermanet et al. [36] applied convolutional sparse coding to pre-train the model layer-by-layer in unsupervised manner. The model is then fine-tuned for the pedestrian detection task on the labeled datasets.

However, it is not clear if static images is the right way to learn visual representations. Therefore, researchers have started focusing on learning feature representations using videos [24, 40, 50, 14, 44, 31]. Early work such as [50] focused on inclusion of constraints via video to autoencoder framework. The most common constraint is the smoothing constraints which enforces learned representations to be temporally smooth. Similar to this, Goroshin et al. [14] proposed to learn auto-encoders based on the slowness prior. Other approaches such as Taylor et al. [44] trained convolutional gated RBMs to learn latent representations from pairs of successive images. This was extended in a recent work by Srivastava et al. [40] where they proposed to learn a LSTM model in an unsupervised manner. Given a few consecutive frames, their optimization goal for LSTM model includes reconstructing the given frames and predicting the future frames. Our work differs from this body of work in two aspects: (a) We train our model with patches obtained from tracking; (b) Instead of training auto-encoders, we train a deep ConvNet which can be transferred to different challenging vision tasks.

Finally, our work is also related to metric learning via deep networks [47, 28, 4, 15, 13, 19]. For example, Chopra et al. [4] proposed to learn convolutional networks in a

\(^1\)http://www.aoa.org/patients-and-public/good-vision-throughout-life/childrens-vision/infant-vision-birth-to-24-months-of-age
Our goal is to train convolutional neural networks using hundreds and thousands of unlabeled videos from the Internet. We follow the AlexNet architecture \cite{21} to design our base network. However, since we do not have labels, it is not clear what should be the loss function and how we should optimize it. But in case of videos, we have another supervisory information: time. For example, we all know that the scene does not change drastically within a short time in a video and same object instances appear in multiple frames of the video. So, how do we exploit this information to train a ConvNet-based representation?

We sample millions of patches in these videos and track them over time. Since we are tracking these patches, we know that the first and last tracked frames correspond to the same instance of the moving object or object part. Therefore, any visual representation that we learn should keep these two data points close in the feature space. But just using this constraint is not sufficient: all points can be mapped to a single point in feature space. Therefore, for training our ConvNet, we sample a third patch which creates a triplet. For training, we use a loss function \cite{47} that enforces that the first two patches connected by tracking are closer in feature space than the first one and a random one.

But training a network with such triplets converges fast since the task is easy to overfit to. One way is to increase the number of training triplets. However, after initial convergence most triplets satisfy the loss function and therefore back-propagating gradients using such triplets is inefficient. Instead, analogous to hard-negative mining, we select the third patch from multiple patches that violates the constraint (loss is maximum). Selecting this patch leads to more meaningful gradients for faster learning.

4. Patch Mining in Videos

The first step in our learning procedure is to extract training instances from videos. In our case, every training instance for learning the deep network consists of three patches. The loss function enforces the pairs of patches connected by tracks to have more similar representations as compared to any other two randomly selected patches. But what do these patches that are tracked correspond to? Since our videos are unlabeled, the location and the extent of the objects in the frame are unknown. Therefore, instead of trying to extract patches corresponding to semantic objects, we focus on using motion information in the video to extract moving image patches. Note that these patches might contain objects or part of an object as shown in Figure 2. These patches are then tracked over time to obtain a second patch. The initial and tracked patches are then grouped together. The details are explained below.

Given a video, we want to extract patches of interest (patches with motion in our case) and track these patches to create training instances. One obvious way to find patches of interest is to compute optical flow and use the high magnitude flow regions. However, since YouTube videos are...
5. Learning Via Videos

In the previous section, we discussed how we can use tracking to generate pairs of patches where the first patch (query) is initialized based on motion and the second patch is obtained after tracking the query patch for 30 frames. We use this procedure to generate millions of such pairs (See Figure 3 for examples of pairs of patches mined). We now describe how we use these as training instances for our visual representation learning.

5.1. Siamese Triplet Network

Our goal is to learn a feature space such that the query patch is closer to the tracked patch as compared to any other randomly sampled patch. To learn this feature space we design a Siamese-triplet network. A Siamese-triplet network consist of three base networks which share the same parameters (see Figure 4). For our experiments, we take the image with size $227 \times 227$ as input. The base network is based on the AlexNet architecture [21] for the convolutional layers. Then we stack two full connection layers on the pool5 outputs, whose neuron numbers are 4096 and 1024 respectively. Thus the final output of each single network is 1024 dimensional feature space $f(\cdot)$. We define the loss function on this feature space.

5.2. Ranking Loss Function

Given the set of patch pairs $S$ sampled from the video, we propose to learn an image similarity model in the form of ConvNet. Specifically, given an image $X$ as an input for the network, we can obtain its feature in the final layer as $f(X)$. Then, we define the distance of two image patches $X_1, X_2$ based on the cosine distance in the feature space as,

$$D(X_1, X_2) = 1 - \frac{f(X_1) \cdot f(X_2)}{\|f(X_1)\| \|f(X_2)\|}.$$  \hspace{1cm} (1)

We want to train a ConvNet to obtain feature representation $f(\cdot)$, so that the distance between query image patch and the tracked patch is small and the distance between query patch and other random patches is encouraged to be larger. Formally, given the patch set $S$, where $X_i$ is the original query patch (first patch in tracked frames), $X_i^+$ is the tracked patch and $X_i^-$ is a random patch, we want to enforce $D(X_i, X_i^+) > D(X_i, X_i^-)$.

Given a triplet of image patches $X_i, X_i^+, X_i^-$ as input, where $X_i, X_i^+$ is a tracked pair and $X_i^-$ is obtained from a different video, the loss of our ranking model is defined by hinge loss as,

$$L(X_i, X_i^+, X_i^-) = \max(0, D(X_i, X_i^+) - D(X_i, X_i^-) + M),$$  \hspace{1cm} (2)

where $M$ represents the gap parameters between two distances. We set $M = 0.5$ in the experiment. Then our objective function for training can be represented as,

$$\min_{W} \frac{\lambda}{2} \|W\|_2^2 + \sum_{i=1}^{N} \max\{0, D(X_i, X_i^+) - D(X_i, X_i^-) + M\},$$  \hspace{1cm} (3)

where $W$ is the parameter weights of the network, i.e., parameters for function $f(\cdot)$. $N$ is the number of the triplets of samples. $\lambda$ is a constant representing weight decay, which is set to $\lambda = 0.0005$. 

Figure 3. Examples of patch pairs we obtain via patch mining in the videos.
Figure 4. Siamese-triplet network. Each base network in the Siamese-triplet network share the same architecture and parameter weights. The architecture is rectified from AlexNet by using only two full connection layers. Given a triplet of training samples, we obtain their features from the last layer by forward propagation and compute the ranking loss.

5.3. Hard Negative Mining for Triplet Sampling

One non-trivial part for learning to rank is the process of selecting negative samples. Given a pair of similar images $X_i, X_i^+$, how can we select the patch $X_i^-$, which is a negative match to $X_i$, from the large pool of patches? Here we first select the negative patches randomly, and then find hard examples (in a process analogous to hard negative mining).

Random Selection: During learning, we perform mini-batch Stochastic Gradient Descent (SGD). For each $X_i, X_i^+$, we randomly sample $K$ negative matches in the same batch $B$, thus we have $K$ sets of triplet of samples. For every triplet of samples, we calculate the gradients over three of them respectively and perform back propagation. Note that we shuffle all the images randomly after each epoch of training, thus the pair of patches $X_i, X_i^+$ can look at different negative matches each time.

Hard Negative Mining: While one can continue to sample random patches for creating the triplets, it is more efficient to search the negative patches smartly. After 10 epochs of training using negative data selected randomly, we want to make the problem harder to get more robust feature representations. Analogous to hard-negative mining procedure in SVM, where gradient descent learning is only performed on hard-negatives (not all possible negative), we search for the negative patch such that the loss is maximum and use that patch to compute and back propagate gradients.

Specifically, the sampling of negative matches is similar as random selection before, except that this time we select according to the loss (Eq. 2). For each pair $X_i, X_i^+$, we calculate the loss of all other negative matches in batch $B$, and select the top $K$ ones with highest losses. We apply the loss on these $K$ negative matches as our final loss and calculate the gradients over them. Notice that since the feature of each sample is already computed after the forward propagation, we only need to calculate the loss over these features, thus the extra computation for hard negative mining is very small. For the experiments in this paper, we use $K = 4$.

5.4. Adapting for Supervised Tasks

Given the ConvNet learned by using unsupervised data, we want to transfer the learned representations to the tasks with supervised data. In our experiments, we apply our model to two different tasks including object detection and surface normal estimation. In both tasks we take the base network from our Siamese-triplet network (which is based on AlexNet architecture) and adjust the full connection layers and outputs accordingly. We introduce two ways to fine-tune and transfer the information obtained from unsupervised data to supervised learning.

One straight forward approach is directly applying our ranking model as a pre-trained network for the target task. More specifically, we use the parameters of the convolutional layers in the base network of our triplet architecture as initialization for the target task. For the full connection layers, we initialize them randomly. This method of transferring feature representation is very similar to the approach applied in RCNN [12]. However, RCNN uses the network pre-trained with ImageNet Classification data. In our case, the unsupervised ranking task is quite different from object detection and surface normal estimation. Thus, we need to adapt the learning rate to the fine-tuning procedure introduced in RCNN. We start with the learning rate with $\epsilon = 0.01$ instead of 0.001 and set the same learning rate for convolutional layers and full connection layers. This setting is crucial since we want the pre-trained features to be used as initialization of supervised learning, and adapting the features to the new task.

In this paper, we explore one more approach to transfer/fine-tune the network. Specifically, we note that there might be more juice left in the millions of unsupervised training data (which could not be captured in the initial learning stage). Therefore, we use an iterative fine-tuning scheme. Given the initial unsupervised network, we first fine-tune using the PASCAL VOC data. Given the new fine-tuned network, we use this network to re-adapt to ranking triplet task. Here we again transfer convolutional parameters for re-adapting. Finally, this re-adapted network is fine-tuned on the VOC data yielding a better trained model. We show in the experiment that this circular approach gives improvement in performance. We also notice that after two iterations of this approach the network converges.

5.5. Model Ensemble

We proposed an approach to learn ConvNets using unlabeled videos. However, there is absolutely no limit to generating training instances and pairs of tracked patches (YouTube has more than billions of videos). This opens...
up the possibility of training multiple ConvNets using different sets of data. Once we have trained these ConvNets, we append the fc7 features from each of these ConvNets to train the final SVM. Note that the ImageNet trained models also provide initial boost for adding more networks (See Table 1).

5.6. Implementation Details

We apply mini-batch SGD in training. As the 3 networks share the same parameters, instead of inputting 3 samples to the triplet network each time, we perform the forward propagation for the whole batch by a single network and calculate the loss based on the output feature. Given a pair of patches $X_i, X_i^+$, we randomly select another patch $X_i^- \in B$ which is extracted in a different video from $X_i, X_i^+$. Given their features from forward propagation $f(X_i), f(X_i^+), f(X_i^-)$, we can compute the loss according to Eq. 2.

For learning, we download 100K videos from YouTube using the URLs provided by [27]. By performing our patch mining method on the videos, we obtain 8 million image patches. We train three different networks separately using 1.5M, 1.5M and 5M training instances. Therefore, we report number based on these three networks. To train our siamese-triplet networks, we set the batch size as $|B| = 100$, the learning rate starting with $\epsilon_0 = 0.001$. For the dataset with 1.5M and 5M patches, we first trained our network with random negative samples with this learning rate for 150K iterations, and then we apply hard negative mining based on it. For training on 1.5M patches, we reduce the learning rate by a factor of 10 at every 80K iterations and train for 240K iterations. For training on 5M patches, we reduce the learning rate by a factor of 10 at every 120K iterations and train for 350K iterations.

6. Experiments

We demonstrate the quality of our learned visual representations with qualitative and quantitative experiments. Qualitatively, we show the convolutional filters learned in layer 1 (See Figure 6). Our learned filters are similar to V1 though not as strong. However, after fine-tuning on PASCAL VOC 2012, these filters become quite strong. We also show that the underlying representation is reasonable by showing what the neurons in Pool5 layers represent (See Figure 5). We use the red bounding boxes to represent the receptive field with top responses for five different neurons (one neuron each line). We can see the clusters represented by these neurons are quite reasonable and correspond to semantic parts of objects. For example, the first neuron represents animal heads, second represents potted plant, etc.

For quantitative evaluations, we evaluate our approach by transferring the feature representation learned in unsupervised manner to the tasks with labeled data. We focus on two challenging problems: object detection and surface normal estimation.

6.1. Object Detection

For object detection, we perform our experiments on PASCAL VOC 2012 dataset [9]. We follow the detection pipeline introduced in RCNN [12], which borrowed the
Table 1. mean Average Precision (mAP) on VOC 2012. The second column “external” represents the number of patches used to pre-train the model in the unsupervised manner.

<table>
<thead>
<tr>
<th>VOC 2012 test</th>
<th>external</th>
<th>aero</th>
<th>bike</th>
<th>bird</th>
<th>boat</th>
<th>bottle</th>
<th>bus</th>
<th>car</th>
<th>chair</th>
<th>cow</th>
<th>table</th>
<th>dog</th>
<th>horse</th>
<th>mbike</th>
<th>person</th>
<th>plant</th>
<th>sheep</th>
<th>sofa</th>
<th>train</th>
<th>tv</th>
<th>mAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>scratch</td>
<td>0</td>
<td>68.9</td>
<td>54.5</td>
<td>34.7</td>
<td>25.3</td>
<td>57.7</td>
<td>56.3</td>
<td>22.0</td>
<td>51.5</td>
<td>51.2</td>
<td>50.3</td>
<td>51.5</td>
<td>53.2</td>
<td>52.3</td>
<td>50.1</td>
<td>56.5</td>
<td>26.4</td>
<td>51.1</td>
<td>34.3</td>
<td>52.3</td>
<td>52.3</td>
</tr>
<tr>
<td>unsup + ft</td>
<td>1.5M</td>
<td>68.8</td>
<td>62.1</td>
<td>34.7</td>
<td>25.3</td>
<td>57.7</td>
<td>56.3</td>
<td>22.0</td>
<td>51.5</td>
<td>51.2</td>
<td>50.3</td>
<td>51.5</td>
<td>53.2</td>
<td>52.3</td>
<td>50.1</td>
<td>56.5</td>
<td>26.4</td>
<td>51.1</td>
<td>34.3</td>
<td>52.3</td>
<td>52.3</td>
</tr>
<tr>
<td>unsup + ft</td>
<td>5M</td>
<td>69.0</td>
<td>64.0</td>
<td>37.1</td>
<td>23.6</td>
<td>24.6</td>
<td>58.7</td>
<td>59.6</td>
<td>22.3</td>
<td>46.0</td>
<td>53.1</td>
<td>53.3</td>
<td>53.7</td>
<td>66.9</td>
<td>54.1</td>
<td>54.1</td>
<td>54.1</td>
<td>54.1</td>
<td>54.1</td>
<td>54.1</td>
<td>54.1</td>
</tr>
<tr>
<td>unsup + ft (2 ensemble)</td>
<td>6.5M</td>
<td>72.4</td>
<td>66.2</td>
<td>41.3</td>
<td>26.4</td>
<td>26.8</td>
<td>61.0</td>
<td>61.9</td>
<td>63.1</td>
<td>52.3</td>
<td>51.0</td>
<td>58.7</td>
<td>58.3</td>
<td>70.0</td>
<td>56.2</td>
<td>56.2</td>
<td>56.2</td>
<td>56.2</td>
<td>56.2</td>
<td>56.2</td>
<td>56.2</td>
</tr>
<tr>
<td>unsup + ft (3 ensemble)</td>
<td>8M</td>
<td>73.5</td>
<td>67.8</td>
<td>43.5</td>
<td>28.9</td>
<td>27.9</td>
<td>62.3</td>
<td>62.6</td>
<td>64.9</td>
<td>72.3</td>
<td>51.5</td>
<td>51.5</td>
<td>51.5</td>
<td>60.0</td>
<td>71.8</td>
<td>58.3</td>
<td>56.1</td>
<td>51.6</td>
<td>51.6</td>
<td>51.6</td>
<td>51.6</td>
</tr>
<tr>
<td>unsup + iterative ft</td>
<td>5M</td>
<td>67.7</td>
<td>64.0</td>
<td>41.3</td>
<td>25.3</td>
<td>27.3</td>
<td>58.6</td>
<td>60.3</td>
<td>60.2</td>
<td>24.3</td>
<td>46.7</td>
<td>54.4</td>
<td>53.8</td>
<td>68.2</td>
<td>55.7</td>
<td>55.7</td>
<td>55.7</td>
<td>55.7</td>
<td>55.7</td>
<td>55.7</td>
<td>55.7</td>
</tr>
<tr>
<td>RCNN 70K</td>
<td></td>
<td>72.7</td>
<td>62.9</td>
<td>49.3</td>
<td>31.1</td>
<td>25.9</td>
<td>56.2</td>
<td>53.0</td>
<td>70.0</td>
<td>23.3</td>
<td>49.0</td>
<td>38.0</td>
<td>69.5</td>
<td>60.1</td>
<td>68.2</td>
<td>46.4</td>
<td>51.7</td>
<td>46.2</td>
<td>50.8</td>
<td>54.1</td>
<td>50.1</td>
</tr>
<tr>
<td>RCNN 70K (2 ensemble)</td>
<td></td>
<td>75.3</td>
<td>68.3</td>
<td>53.1</td>
<td>35.2</td>
<td>27.7</td>
<td>59.6</td>
<td>54.7</td>
<td>73.4</td>
<td>26.5</td>
<td>53.0</td>
<td>42.2</td>
<td>73.1</td>
<td>66.1</td>
<td>71.0</td>
<td>48.5</td>
<td>21.7</td>
<td>50.8</td>
<td>55.2</td>
<td>58.0</td>
<td>53.6</td>
</tr>
<tr>
<td>RCNN 70K (3 ensemble)</td>
<td></td>
<td>74.6</td>
<td>68.7</td>
<td>54.9</td>
<td>35.7</td>
<td>29.4</td>
<td>61.0</td>
<td>54.4</td>
<td>74.0</td>
<td>28.4</td>
<td>53.6</td>
<td>43.0</td>
<td>74.0</td>
<td>66.1</td>
<td>72.8</td>
<td>50.3</td>
<td>20.5</td>
<td>60.0</td>
<td>51.2</td>
<td>57.9</td>
<td>58.0</td>
</tr>
<tr>
<td>RCNN 200K (big stepsize)</td>
<td>9M</td>
<td>73.3</td>
<td>67.1</td>
<td>46.3</td>
<td>31.7</td>
<td>30.6</td>
<td>59.4</td>
<td>61.0</td>
<td>67.9</td>
<td>27.3</td>
<td>53.1</td>
<td>39.1</td>
<td>64.1</td>
<td>60.5</td>
<td>70.9</td>
<td>57.2</td>
<td>26.1</td>
<td>59.0</td>
<td>40.1</td>
<td>54.9</td>
<td>52.3</td>
</tr>
</tbody>
</table>

Figure 6. Conv1 filters visualization. (a) The filters of the first convolutional layer of the siamese-triplet network trained in unsupervised manner. (b) By fine-tuning the unsupervised pre-trained network on PASCAL VOC 2012, we obtain sharper filters.

ConvNets pre-trained on other datasets and fine-tuned on it using the VOC data. The fine-tuned ConvNet was then used to extract features followed by training SVMs for each object class. However, instead of using ImageNet pre-trained network as initialization in RCNN, we use our ConvNets trained in the unsupervised manner. Note that the network architecture is based on AlexNet. We fine-tune our network with the trainval set (11540 images) and train SVMs with the same images. Evaluation is performed in the standard test set (10991 images).

At the fine-tuning stage, we change the output to 21 and initialize the convolutional layers with our unsupervised pre-trained network. To fine-tune the network, we start with learning rate as $\epsilon = 0.01$ and reduce the learning rate by a factor of 10 at every 80K iterations. The network is fine-tuned for 200K iterations. Note that for all the experiments, no bounding box regression is performed.

We compare our method with the model trained from scratch as well as using ImageNet pre-trained network. Not-tice that the results for VOC 2012 reported in RCNN [12] are obtained by only fine-tuning on the train set without using the val set. The mAP for VOC 2012 reported in [12] is 49.6%. For fair comparison, we fine-tuned the ImageNet pre-trained network with VOC 2012 trainval set. Moreover, as the step size of reducing learning rate in RCNN [12] is set to 20K and iterations for fine-tuning is 70K, we also try to enlarge the step size to 50K and fine-tune the network for 200K iterations. We report the results for both of these settings.

**Single Model.** We show the results in Table 1. As a baseline, we train the network from scratch on VOC 2012 dataset and obtain 44% mAP. Using our unsupervised network pre-trained with 1.5M pair of patches and then fine-tuned on VOC 2012, we obtain mAP of 46.2% (unsup+ft, external data = 1.5M). By looking into more data, using 5M patches in pre-training and then fine-tune, we can achieve 47% mAP (unsup+ft, external data = 5M). These results indicate that our unsupervised network provides a significant boost as compared to the scratch network. More importantly, when more unlabeled data is applied, we can get better performance (3% boost compared to training from scratch).

**Model Ensemble.** As looking at more external data in unsupervised pre-training gives the boost in performance, we also try combining different models using different unlabeled data in pre-training. By ensembling two fine-tuned networks which are pre-trained using 1.5M and 5M patches, we obtained a boost of 3.5% comparing to the single model, which is 50.5%(unsup+ft (2 ensemble)). By moving one step forward, we ensemble all three different networks pre-trained with different sets of data, whose size are 1.5M, 1.5M, and 5M respectively. We get another boost of 1.5% and reach 52% mAP (unsup+ft (3 ensemble)).

**Baselines.** We also compare our approach with RCNN [12] which uses ImageNet pre-trained models. Following the procedure in [12], we obtain 50.1% mAP (RCNN 70K) by setting the step size to 20K and fine-tuning for 70K iterations. To generate a model ensemble, three ConvNets are first trained on the ImageNet dataset sepa-
Table 2. Results on NYU v2 for per-pixel surface normal estimation, evaluated over valid pixels.

<table>
<thead>
<tr>
<th></th>
<th>(Lower Better)</th>
<th>(Higher Better)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mean</td>
<td>Median</td>
</tr>
<tr>
<td>scratch</td>
<td>38.6</td>
<td>26.5</td>
</tr>
<tr>
<td>unsup + ft</td>
<td>35.2</td>
<td>23.2</td>
</tr>
<tr>
<td>ImageNet + ft</td>
<td>33.3</td>
<td>20.8</td>
</tr>
<tr>
<td>UNFOLD [11]</td>
<td>35.1</td>
<td>19.2</td>
</tr>
<tr>
<td>Discr. [22]</td>
<td>32.5</td>
<td>22.4</td>
</tr>
<tr>
<td>3DP (MW) [10]</td>
<td>36.0</td>
<td>20.5</td>
</tr>
</tbody>
</table>

Rate, and then they are fine-tuned with the VOC 2012 dataset. The result of ensembling two of these networks is 53.6% mAP (RCNN 70K (2 ensemble )), and ensembling three of these networks gives 0.8% improvement, leading to 54.4% mAP (RCNN 70K (3 ensemble )). For fair of comparison, we also fine-tuned the ImageNet pre-trained model with larger step size(50K) and more iterations(200K). The result is 52.3% mAP (RCNN 200K (big stepsize)). Note that while ImageNet network shows diminishing returns with ensembling since the training data remains similar, in our case since every network in the ensemble looks at different sets of data, we get huge performance boosts.

**Exploring a better way to transfer learned representation.** Given our fine-tuned model using 5M patches in pre-training (unsup+ft, external = 5M), we use it to re-learn and re-adapt to the unsupervised triplet task. After that, the network is re-applied to fine-tune on VOC 2012. We repeat this iterative approach twice in our experiment and find it converges very quickly. The final result for this single model is 48% mAP (unsup + iterative ft), which is 1% better than the initial fine-tuned network.

6.2. **Surface Normal Estimation**

To illustrate that our unsupervised representation can be generalized to different tasks, we adapt the unsupervised ConvNet to the task of surface normal estimation from a RGB image. In this task, we want to estimate the orientation of the pixels. We perform our experiments on the NYUv2 dataset [37], which includes 795 images for training and 654 images for testing. Each image is has corresponding depth information which can be used to generate groundtruth surface normals. For evaluation and generating the groundtruth, we adopt the protocols introduced in [10] which is used by different methods [10, 22, 11] on this task.

To apply deep learning to this task, we followed the same form of outputs and loss function as the coarse network mentioned in [48]. Specifically, we first learn a codebook by performing k-means on surface normals and generate 20 codewords. Each codeword represents one class and thus we transform the problem to 20-class classification for each pixel. Given a $227 \times 227$ image as input, our network generates surface normals for the whole scene. The output of our network is $20 \times 20$ pixels, each of which is represented by a distribution over 20 codewords. Thus the dimension of output is $20 \times 20 \times 20 = 8000$.

The network architecture for this task is also based on the AlexNet. To relieve over-fitting, we only stack two full connection layers with 4096 and 8000 neurons on the pool5 layer. During training, we initialize the network with the unsupervised pre-trained network. We use the same learning rate $1.0 \times 10^{-6}$ as mentioned in [48] and fine-tune the network with 10K iterations given the small number of training data. Note that unlike [48], we do not utilize any data from the videos in NYU dataset for training.

For comparisons, we also trained networks from scratch as well as using ImageNet pre-trained. We show our results in Table 2. Compared to the recent results which do not use external data [10, 22, 11], we show that we can get reasonable results even using this small amounts of training data. Our approach(unsup + ft) is generally $2 \sim 3\%$ better than network trained from scratch in 5 different metrics. We show a few qualitative results in Figure 7.

**Acknowledgement:** This work was partially supported by ONR MURI N00014141010934, NSF IIS 1320083, and gifts from Google and Yahoo!. AG was partially supported by Bosch Young Faculty Fellowship. The authors would like to thank Yahoo! and Nvidia for the compute cluster and GPU donations respectively.

**References**


