Discriminative unsupervised feature learning with exemplar convolutional neural networks

(2015)

Alexey Dosovitskiy, Philipp Fischer, Jost Tobias Springenberg,
Martin Riedmiller, Thomas Brox

Notes

Contribution

This paper presents a method for pre-training a CNN using unlabled data, the network is trained to be robust to the set of transformations applied to a given image patch. Each selected patch of an image (or seed) and its transformed versions constitute a *surrogate* class, and the training objective is to then classify all of the instance from this set into its corresponding surrogate class.

Using such self-supervised training objective without the need of any labeled data, we can pretrain the network (called Exemplar-CNN) and reuse the representations learned by the network that are invariant to some typical transformation for various vision tasks. The authors investigated such forms of transfer learning and found that the Exemplar-CNN preforms well on object classification and descriptor matching.

Creating training dataset

The first step is to create training examples using the unlabled data. Given a set of unlabled images, we sample N patches of size 32×32 from different image and at different positions and scales, these patches are only sampled from regions with considerable gradients (e.g. after applying HOG), so that only patches containing objects are sampled. Giving us the initial set $X = \{x_1, x_2, ..., x_N\}$ (fig1 below).

Fig. 1. Exemplary patches sampled from the STL unlabeled dataset which are later augmented by various transformations to obtain surrogate data for the CNN training.

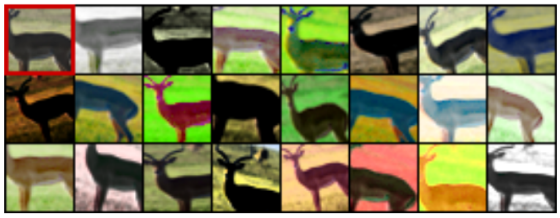

Fig. 2. Several random transformations applied to one of the patches extracted from the STL unlabeled dataset. The original ('seed') patch is in the top left corner.

We then define a set of transformations T_α parametrized by a vector $\alpha \in \mathcal{A}$ of all possible parameters $\mathcal{A}$, in this case the set of transformations are: translation (V and H by a distance of 0.2 of the patch size), scaling (factor of 0.7 to 1.4), rotation (up to 20 degrees), contract by raising the S and V values, and using the PCA of the pixels, and finally by jittering the H and V colors by a value from 0.1 to -0.1.

So for each patch x_i, we sample K transformations with their corresponding parameter vectors $\alpha_i, i \in \{1, \ldots, k\}$, and apply the corresponding K transformations $T_{\alpha i}$ for a patch x_i. So we end up

with K transformed patch for each patches x_i of the N patches (fig2 above), for a total of $N \times (K+1)$ patches, and N surrogate classes.

Training

With N sets of K transformed patches, giving us N surrogate classes, we train the CNN to classify a given transformed patch Tx_i into its correct surrogate class:

$$L(X) = \sum_{\mathbf{x}_i \in X} \sum_{T \in \mathcal{T}_i} l\left(i, T\mathbf{x}_i\right)$$

The loss is a cross entropy loss, and the CNN contains a fully connected classification layer followed by a softmax to transform the logits into probabilities, where the objective is to differentiate between different patches, and being robus to a given number of transformations at the same time.

Formal Analysis

To see why this simple training procedure might work, we need to carefully analyse the loss function, let's take the case of an infinte number of transformations:

$$\widehat{L}(X) = \sum_{\mathbf{x}_i \in X} \mathbb{E}_\alpha \left[l\left(i, T_\alpha \mathbf{x}_i\right)\right]$$

Where:

$$l\left(i, T\mathbf{x}_i\right) = M\left(\mathbf{e}_i, f\left(T\mathbf{x}_i\right)\right) = -\sum_k \mathbf{e}_i \log f\left(T\mathbf{x}_i\right))$$

$$f(\mathbf{x}) = \text{softmax}(\text{h}(\mathbf{x})) = \exp(h(\mathbf{x}))/\| \exp(h(\mathbf{x})\|_1$$

$$h(\mathbf{x}) = \mathbf{W}g(\mathbf{x})$$

So $\mathbf{g}$ is the CNN network, $\mathbf{h}(x)$ is (CNN & FC) and gives the outputs after the fully connected layer and $\mathbf{f}(x)$ is the prediction function of the whole model (CNN, FC and softmax). Given these definitions, we can rewrite the first equation in the following form:

$$\widehat{L}(X) = \sum_{\mathbf{x}_i \in X} \left[-\left\langle \mathbf{e}_i, \mathbf{W}\widehat{\mathbf{g}}_i \right\rangle + \log \left\| \exp\left(\mathbf{W}\widehat{\mathbf{g}}_i\right) \right\|_1\right]$$
$$+ \sum_{\mathbf{x}_i \in X} \left[\mathbb{E}_\alpha \left[\log \left\| \exp\left(h\left(T_\alpha \mathbf{x}_i\right)\right) \right\|_1\right] - \log \left\| \exp\left(\mathbf{W}\widehat{\mathbf{g}}_i\right) \right\|_1\right]$$

Here we endup with two terms, the first one is a multinomial logistic regression in which the objective is to minimize the error between the target $\boldsymbol{e}_i$ and the average of the transformed representations $\mathbb{E}_\alpha \left[g\left(T_\alpha \mathbf{x}_i\right)\right] = \widehat{\boldsymbol{g}}_i$, the second term is a regulizer enforcing all the output representation $h\left(T_\alpha \mathbf{x}_i\right)$ to be close to their average values.

In an unsupervised setting, the objective is to model the input distribution $p(x)$, and based on the assumption that a good model of $p(x)$ contains information about the category distribution $p(y|x)$, and that the learned representations, that can reconstruct the example perfectly, encode some infromation abour the category of the sample that can be used for various vision tasks. In our case, we don't model the input distribution, but we learn a representation that discriminates between the input patches but is still invariant to a set of transofmations, by forcing the representation of the transformed patches to be predictive of the surrogate classes.

Experiments

Here we compare a model pretrained using examplar CNN and fine tuned on a given vision task and its corresponding dataset, we also have some ablation studies for the number of surrogate classes N and the number of transformations K per class, and the types of transformations used:

Algorithm	STL-10	CIFAR-10(400)	CIFAR-10	Caltech-101	Caltech-256(30)	#features
Convolutional K-means Network [32]	60.1 ± 1	70.7 ± 0.7	82.0	—	—	8000
Multi-way local pooling [33]	—	—	—	77.3 ± 0.6	41.7	1024×64
Slowness on videos [14]	61.0	—	—	74.6	—	556
Hierarchical Matching Pursuit (HMP) [34]	64.5 ± 1	—	—	—	—	1000
Multipath HMP [35]	—	—	—	82.5 ± 0.5	50.7	5000
View-Invariant K-means [16]	63.7	72.6 ± 0.7	81.9	—	—	6400
Exemplar-CNN (64c5-64c5-128f)	67.1 ± 0.2	69.7 ± 0.3	76.5	$79.8 \pm 0.5^{*}$	42.4 ± 0.3	256
Exemplar-CNN (64c5-128c5-256c5-512f)	72.8 ± 0.4	75.4 ± 0.2	82.2	$86.1 \pm 0.5^{\dagger}$	51.2 ± 0.2	960
Exemplar-CNN (92c5-256c5-512c5-1024f)	$\mathbf{74.2 \pm 0.4}$	$\mathbf{76.6 \pm 0.2}$	**84.3**	$\mathbf{87.1 \pm 0.7}^{\ddagger}$	$\mathbf{53.6 \pm 0.2}$	1884
Supervised state of the art	70.1[36]	—	92.0 [37]	91.44 [38]	70.6 [2]	—

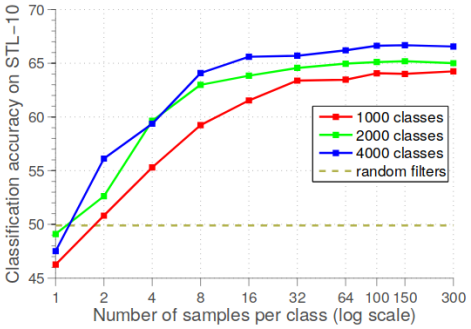

Fig. 4. Classification performance on STL for different numbers of samples per class. Random filters can be seen as '0 samples per class'.

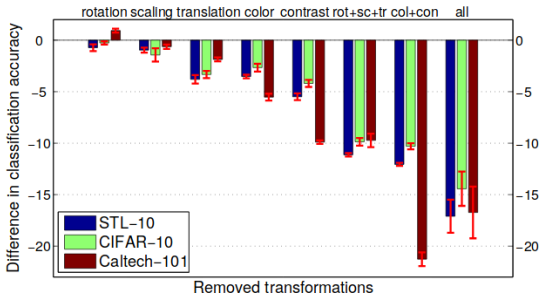

Fig. 5. Influence of removing groups of transformations during generation of the surrogate training data. Baseline ('0' value) is applying all transformations. Each group of three bars corresponds to removing some of the transformations.

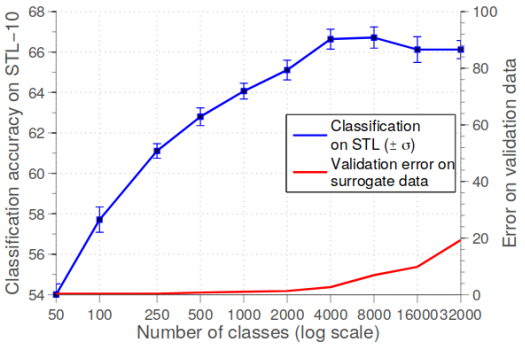

Fig. 3. Influence of the number of surrogate training classes. The validation error on the surrogate data is shown in red. Note the different y-axes for the two curves.