Learning Hierarchical Features for Scene Labeling

1 Citation

Farabet, Clement, et al. "Learning hierarchical features for scene labeling." IEEE transactions on pattern analysis and machine intelligence 35.8 (2013): 1915-1929.

https://hal-upec-upem.archives-ouvertes.fr/file/index/docid/742077/filename/farabet-pami-13.pdf

2 Abstract

We use a CNN feature extractor to build a system that can label each image in a pixel with its category. We try many postprocessing steps, one of which is a way to take a pool of segmentation components and select the best components to describe the image. We set state-of-the-art on the 33-class Sift Flow Dataset, the 170-class Barcelona dataset, and 8-class Stanford Background Dataset (near state-of-the-art). Our system processes a 320×240 image in under a second.

3 Introduction

Full-scene labeling (also called scene parsing) is when we tag every pixel with a category. There are two challenges in this problem: (1) making good feature vectors, (2) making sure that predictions are consistent with each other. We use a multi-scale convolutional neural network to label each pixel. The sophistication of the CNN allows our post-processing steps (which ensure consistency) to be relatively simple. Our two main components are:

1. CNN applied on different scales (from Laplacian pyramid) and trained to predict label for each pixel. It doesn't predict object boundaries, which is why we need post-processing.
2. We oversegment the image (i.e break it into regions), group feature vectors in each segment, and pass the result to labeling. Our main oversegmentation techniques are:
   (a) Superpixel: These are small regions that are roughly uniform within themselves. Each labeled pixel in the superpixel votes to label the superpixel. Superpixels are fixed-size, which is limiting.

   (b) Conditional Random Field over Superpixels: This graph-based approach smoothes the superpixels (avoids aberrations like person in the sky).

   (c) Multilevel Cut with Purity Criterion: Make family of segmentations at each level (e.g. segmentation tree, super-pixels with different algorithm parameters). For each pixel in a segment, get feature vector, and do component-wise max-pooling of feature vectors. Classify each segment and get probability distribution of classes. Select segments to minimize average impurity (i.e. entropy over class distribution)

All operations are nearly linear, with the CNN being the most compute-intensive. There are no parameters in the system.

4 Related Work

Previous work uses Markov Random Fields or Conditional Random Fields to smooth superpixels.

Other work uses deep learning on hand-engineered features.

Other work uses graph cuts to aggregate candidate segments (our purity criterion is easier).

5 Multiscale Feature Extraction for Scene Parsing

We want a feature extractor $f : \mathbb{R}^P \to \mathbb{R}^Q$ that turns an image $\mathbb{R}^P$ into a linearly-classifiable feature vector $\mathbb{R}^Q$. We do feature extraction with a multi-scale CNN. Given an image I, we create scaled versions X_s for $s \in \{1...N\}$ with a Laplacian pyramid (neighborhoods have zero-mean unit-variance) and process each densely with a CNN. Our CNN has three layers. The first two have convolution, *tanh* nonlinearity, and pooling, while the last only has convolution. That is:

$$f(X, \theta) = W_L H_{L-1}$$

$$H_l = \text{pool}(\tanh(W_l H_{l-1} + b_l))$$

where $L = 3$, $l \in \{1...L\}$, $H_0 = X$ and the W matrices apply convolution.

The output is a set of N feature maps $(\mathbf{f_1}, ..., \mathbf{f_N}))$, which we upsample (denote as u) to have the same size. Then we concatenate them together to get the feature vector:

$$\mathbf{F} = [\mathbf{f_1}, u(\mathbf{f_2}), ..., u(\mathbf{f_N})]$$

To train the CNN, we strap on a softmax layer to predict the class distribution for pixel i as:

$$\hat{\mathbf{c}}_{i,a} = \frac{e^{\mathbf{w}_a^T \mathbf{F}_i}}{\sum_{b \in classes} e^{\mathbf{w}_b^T \mathbf{F}_i}}$$

and we optimize cross-entropy loss:

$$L_{cat} = \sum_{i \in pixels} \sum_{a \in classes} \mathbf{c}_{i,a} \ln \hat{\mathbf{c}}_{i,a}$$

6 Scene Labeling Strategies

One simple way to label a pixel is to take the argmax, but this is noisy.

$$l_i = \text{argmax}_{a \in classes} \hat{\mathbf{c}}_{i,a}$$

Another alternative is to oversegment with superpixels and predict a label for each pixel using a 2-layer neural network (class distribution for superpixel k is $\hat{\mathbf{d}}_k$) and take an argmax over the distribution for the superpixel.

$$\mathbf{y}_i = W_2 \tanh(W_1 F_i + b_1)$$

$$\hat{\mathbf{d}}_{i,a} = \frac{e^{\mathbf{y}_{i,a}}}{\sum_{b \in classes} e^{\mathbf{y}_{i,b}}}$$

$$L_{cat} = \sum_{i \in pixels} \sum_{a \in classes} \mathbf{d}_{i,a} \ln \hat{\mathbf{d}}_{i,a}$$

$$\hat{\mathbf{d}}_{k,a} = \frac{1}{surface(k)} \sum_{i \in k} \hat{\mathbf{d}}_{i,a}$$

The problem with this approach is that it doesn't incorporate global information. To mitigate this, we can use a Conditional Random Field (CRF). Each pixel is a node and we draw an edge between neighboring pixels. Then we pick the labeling, l, that minimizes this energy function with alpha-expansions:

$$E(l) = \sum_{i \in V} \Phi(\hat{\mathbf{d}}_i, l_i) + \gamma \sum_{e_{ij} \in E} \Psi(l_i, l_j)$$

where $\Phi(\hat{\mathbf{d}}_{i.a}, l_i) = \exp(-\alpha \hat{\mathbf{d}}_{i,a}) \mathbb{I}(l_i \neq a)$, $\Psi(l_i, l_j) = \exp(-\beta ||\nabla I||_i) \mathbb{I}(l_i \neq l_j)$, and $||\nabla I||_i$ is the L2 norm of the gradient of the image at pixel i.

There's still one problem. How do we handle multiple image scales? Well, suppose we have a set of segmentations (called components) of the image (they may overlap and be different sizes): $C_k \forall k \in \{1...K\}$ (C_k has score S_k). We pick the best k for a pixel i as follows:

$$k^*(i) = \text{argmin}_{k | i \in C_k} S_k$$

In practice, we arrange components in a hierarchy (i.e. a segmentation tree), where a parent component contains its children - this is how you can structure components taken at different scales. Then you can do a depth-first search of thre tree and pick the best k for each pixel.

How do you pick the score for a component? We just say it's the entropy of the component's class distribution. How do we compute the class distribution? First, consider the feature vectors inside the component in question: $\mathbf{F} \in C_k$. Do a max-pooling to create a $G \times G$ grid with some number of channels (we set $G = 3$), call it O_k. Feed this into a 2-layer neural network that predicts a class distribution. Optimize the neural network with cross entropy loss.

$$\mathbf{y}_k = W_2 \tanh(W_1 O_k + b_1)$$

$$\hat{\mathbf{d}}_{k,a} = \frac{e^{\mathbf{y}_{k,a}}}{\sum_{b \in classes} e^{\mathbf{y}_{k,b}}}$$

$$S_k = -\sum_{a \in classes} \mathbf{d}_{k,a} \ln \hat{\mathbf{d}}_{k,a}$$

Now that we can predict the class distribution for each component, we can find the entropy of the distribution and use that as S_k for the component (this is different from the cross entropy function, which depends on the ground truth - normalized histogram of ground truth pixel distributions in component). Once we select the optimal components, we get their labels with an arg max.

$$l_k = \text{argmax}_{a \in classes} \hat{\mathbf{d}}_{k,a}$$

7 Experiments

The Stanford Background dataset (outdoor scenes) has 700 images and 8 classes. SIFT flow has 2700 images and 33 classes. Barcelona (streets of Barcelona) has 15000 images and 170 classes.

We evaluate (1) CNN (2) Multiscale CNN (3) Superpixel (4) CRF (5) Cover predictions.

As desired, Cover sets the state of the art on Barcelona and SIFT flow. We are near state of the art on Stanford, and the models that do better than us take minutes to run on an image (we take < 1 second).

We tried two sampling strategies: balanced sampling and class frequency sampling. Balanced lets us do better at finding small objects, but hurts overall accuracy. Technically it's the better thing to do because it doesn't assume priors over classes. It also does better on Barcelona, which has few examples for many classes and thus frquency sampling leads to overfitting.

Our CNN uses 7×7 filters, 2×2 max-pooling, YUV (not RGB) images, a Laplacian pyramid with three scales and 15×15 neighborhood zero-mean/unit-variance. For the CNN, to produce the next layer, we randomly sample feature maps from the previous layer. Each feature vector $\mathbf{F}$ is 768 dimensions. We train with stochastic gradient with a batch size of 1 and a 10% validation set. We augmented the data with flips, small rotations, and rescaling. This outperforms the non-multi-scale CNN, but struggles to delineate object boundaries.

The Superpixel approach works well and quickly. The 2-layer neural network has 768 input units, 1024 hidden units, and the number of output units matches the number of classes. If you use multiple superpixels at different levels and then use the optimal cover technique, it does even better.

If you also include the CRF, you get state of the art. We selected $\alpha, \beta, and \gamma$ with grid search.

To see how our features generalize, we filmed and segmented a 360 movie in our workplace and did segmentation on each frame with the multi-scale CNN + Superpixel approach.

8 Discussion

Here's what we learned.

Learning features is better than hand-engineering them. The features need to consider a large window to perform well. Even the simple multiscale CNN + Superpixel approach does really well. Using complex post-processing schemes does not help more, which means we could be limited by the labeling quality of the ground truth. The CNN and feedforward neural networks are much faster than approaches that rely on graphical models.

9 Conclusion and Future Work

A multi-scale CNN is a great feature extractor and does surprisingly well on scene parsing. Combining with post-processing steps like Superpixel, CRF, and Cover gives some gains.

Our system doesn't do well when there are a ton of classes (no existing system does well in this case). Maybe the scene parsing problem needs to be refined. Using pixelwise accuracy is a bad idea because a model that accurately finds the boundary for the sky but completely misses a small object would still do well on that metric. In practice, it would be better to detect the small object and get a rough boundary for the sky, but the evaluation metrics do not incentivize this. Datasets need better labels. Maybe we could use hierarchical labeling (e.g. building + window contains building and window) in the form of "is-part-of" relationships.

We train the CNN separately from post-processing. Maybe we could backprop the gradient through postprocessing as well, but we couldn't get it to work. This would also allow us to use weakly labeled data because the CNN would not need every pixel in the training image labeled.