# *Supplementary Material*


Figure S1: **Segmentation refinement.** Single object segmentation with our method and bilateral solver[6] as a complementary step to refine the obtained segmentation.


Figure S2: **Segmentation refinement.** Resolution manipulation using smaller size stride. ViT image input resolution is $280 \times 280$ for both images.

## 1. Resolution Manipulation

In this work, we utilize deep features extracted from ViTs. Those features represent the image patches corresponding to the ViT patch size $p \times p$. We perform our segmentation patch-wise thus, for image size $(m \times n)$ our segmentation resolution is $(\frac{m}{p} \times \frac{n}{p})$. For higher resolution segmentation, we change the ViT stride value to $\frac{p}{2}$ instead of $p$ as it doubles the number of patches the ViT uses, doubling the resolution of our segmentation to $(\frac{2m}{p} \times \frac{2n}{p})$. We found this method to yield better than changing input image resolutions as the transformer in use in this paper wasn't trained on high resolution images. Example at Fig. S2

In order to improve the segmentation resolution further, a bilateral solver[6] can be added as a complementary step to refine the boundaries of the obtained segmentation: see Figure S1. All reported results int the paper is without any post-processing methods.

## 2. Implementation details

For all experiments, we use DINO [9] trained ViT-S/8 transformer for feature extraction; specifically, we use the *keys* features from the last layer of the DINO trained "student" transformer. We use pre-trained weights provided by the DINO paper authors (trained on ImageNet[25]). **We do not conduct any training of the transformer on any of the tested datasets.** Input images resized to a resolution of $280 \times 280$, images resized using *Lanczos* interpolation. All expirements where conducted using *Tesla V100* GPU.

**Algorithm 1** DeepCut

1: $x \leftarrow Input\ image$
2: $x \leftarrow ViT(x)$ ▷ Deep features from ViT
3: $G \leftarrow Build\ graph(x)$
4: **for** Each training epoch **do**
5: $\quad s \leftarrow GCN(G)$ ▷ Single layer of GCN
6: $\quad s \leftarrow ELU(s)$
7: $\quad s \leftarrow MLP(s)$ ▷ Two layer MLP
8: $\quad s \leftarrow softmax(s)$
9: $\quad Loss \leftarrow \mathcal{L}NCut$ or $\mathcal{L}CC$
10: **end for**
11: $Output\ segmentation \leftarrow argmax(s)$

**ViT** Evaluation mode, frozen weights.

**Build graph** Create a graph from deep features as depicted at Sec. 3.1.

**GCN** Graph Convolutional Network[35]. *Input size* = Deep features size, *hidden size* = 64. *Learning rate* = 0.001.

**ELU** The Exponential Linear Unit activation function.

**MLP** Consists from 2 linear layers, **layer 1:** from GCN *hidden size* to $\frac{hidden\ size}{2}$. **layer 2:** from $\frac{hidden\ size}{2}$ to $k$ the number of desired clusters. For k-less usage with correlation clustering, the output will be set to a maximum of desired clusters. Between the layers, there is an *elu* activation function and 0.25 dropout.

**Loss** We suggest two loss function derived from classical graph theory; NCut and CC.

**Output segmentation** At step 8, in order to obtain the final segmentation, the vector $s$ is extracted. Each entry in this vector corresponds to a patch of the image and contains a probability vector that describes the likelihood of the patch belonging to a specific cluster. We chose the most likely cluster assignment for each patch and than unflatten the result to get an segmentation map.

### 2.1. Two-stage segmentation

The clustering functionals in this paper exhibit are biased towards larger clusters (e.g background-foreground), resulting in a tendency to underperform on finer details by merging them together, or in some cases, failing and introducing a significant amount of noise to the segmentation process. To address this issue, a solution is proposed by utilizing a two-stage segmentation approach, where the background and foreground segments are separately applied to avoid the aforementioned biases and limitations. Example can be seen at Fig. S5.

### 2.2. Training

To optimize object localization and object segmentation task, we perform individual optimization for each image for a duration of 10 epochs, with model weights being reset between images. For part semantic segmentation, we carry out separate optimization for each image over a span of 100 epochs, without resetting the model weights between them.

### 2.3. Performance

All of the results presented in the paper demonstrate DeepCut without the utilization of any post-processing (e.g. bilateral solver). All experiments were conducted using the same hardware: Tesla V100 GPU and an Intel Xeon 32 core CPU.

| Model | DUTS [mIoU] | ECSSD [mIoU] | Throughput [img/sec] |
|---|---|---|---|
| TokenCut + Bilateral Solver | 62.4 | 77.2 | 0.5 |
| TokenCut w/o Bilateral Sol. | 57.6 | 71.2 | 1 |
| Ours | 59.5 | 74.6 | 5 |

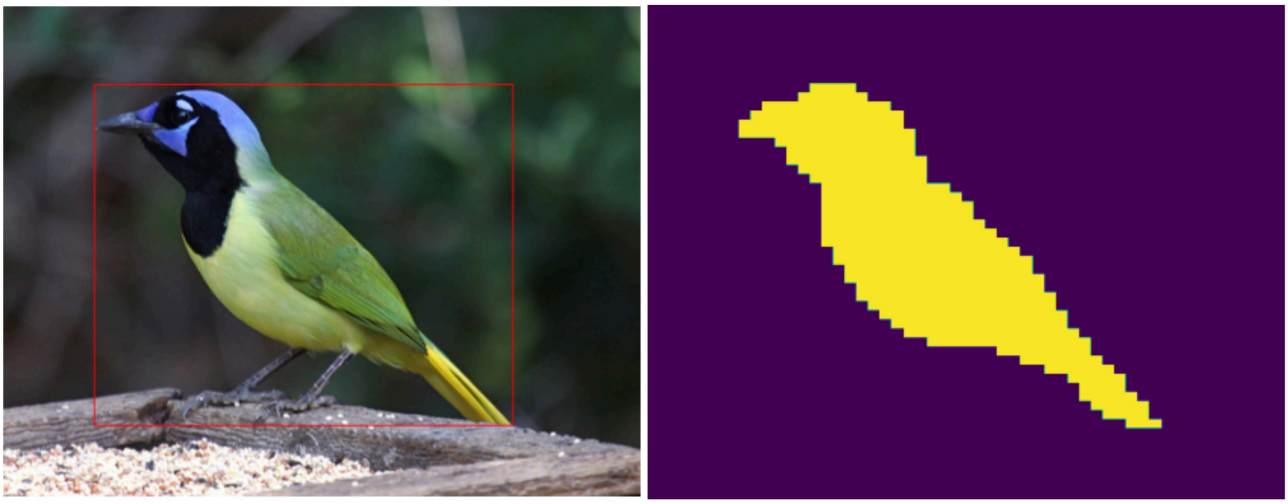

Figure S3: **Method example:** Object localization using DeepCut(NCut).

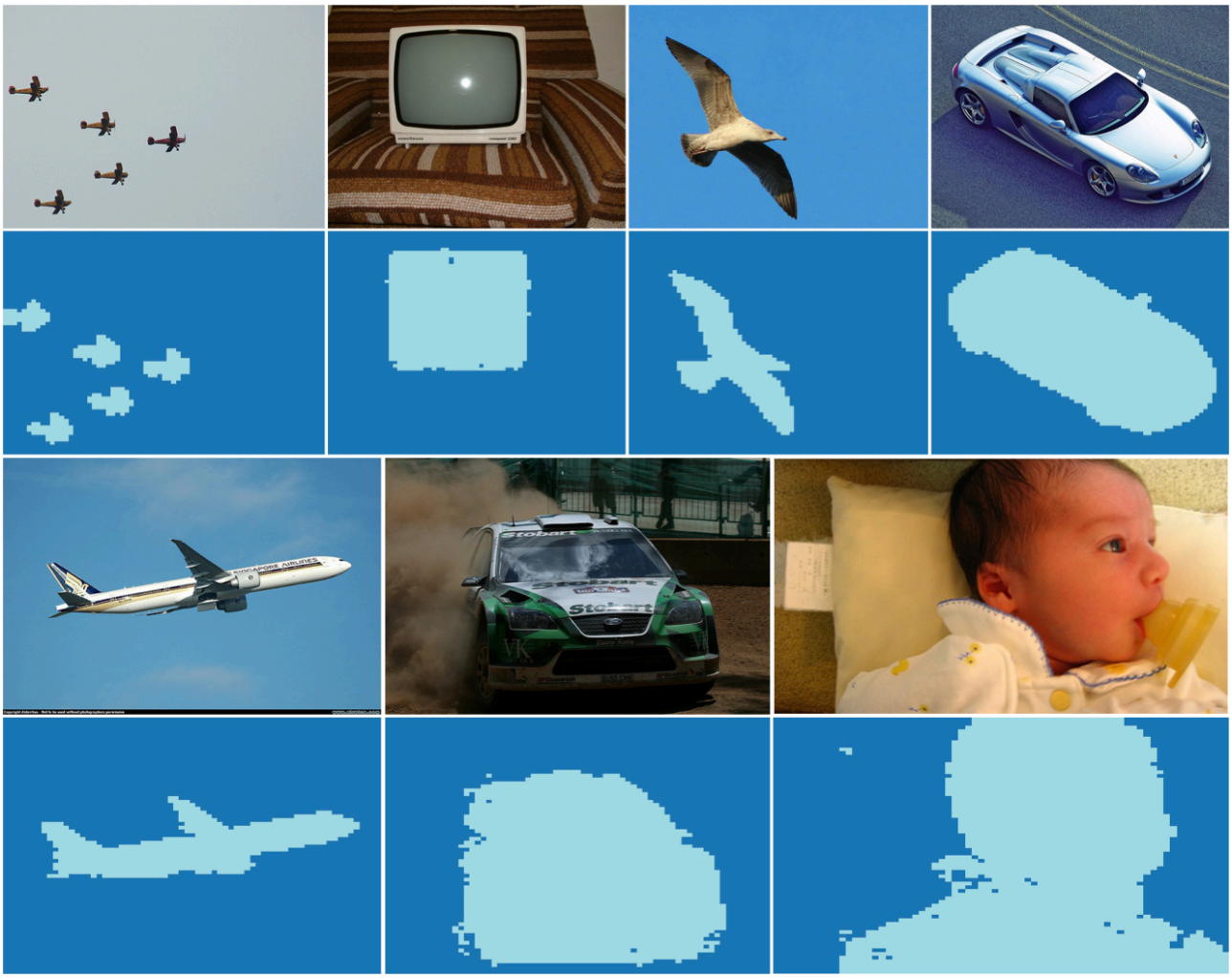

Figure S4: **Method example:** Random foreground-background segmentation samples using DeepCut(NCut) on VOC07.

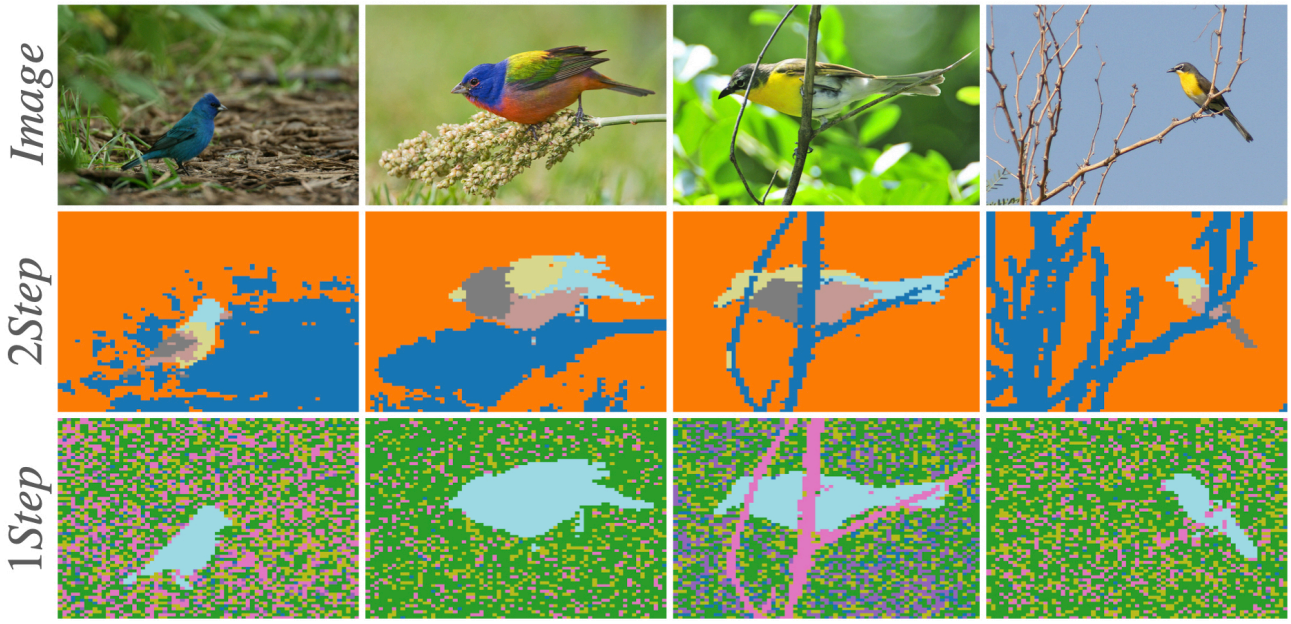


Figure S5: **Method example:** Two-stage segmentation using DeepCut(NCut).

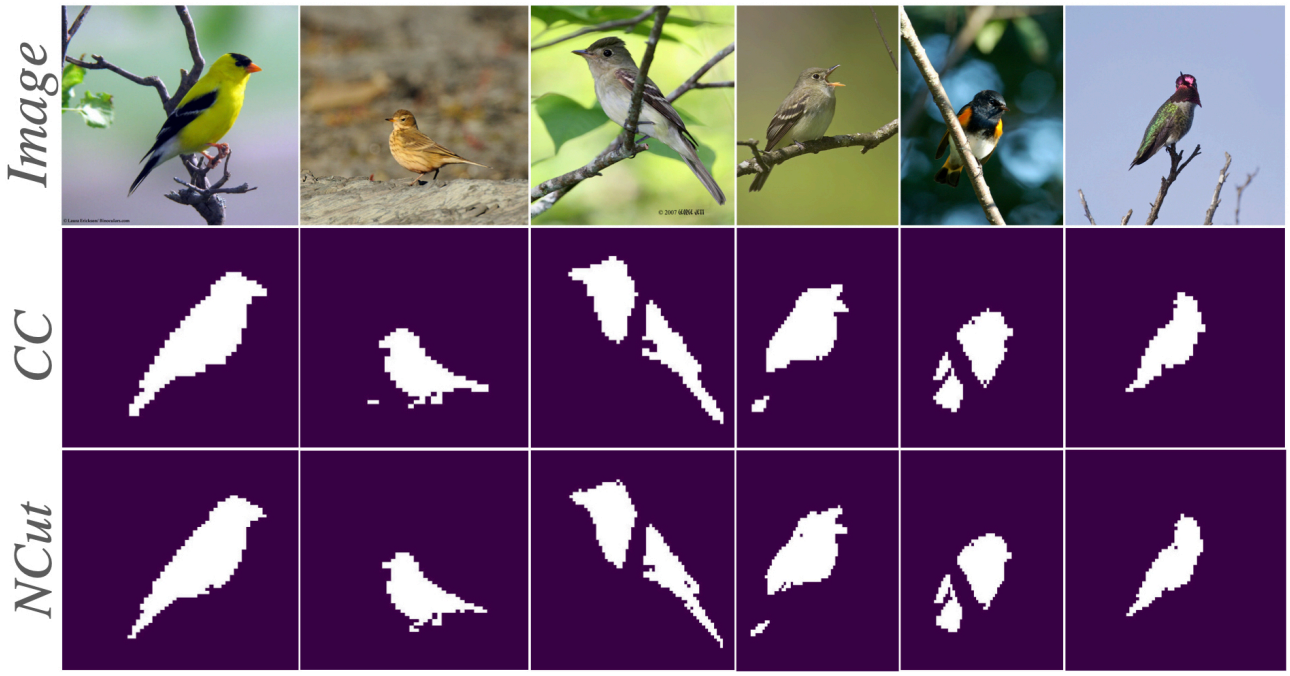


Figure S6: **Method example:** Random foreground-background segmentation samples using DeepCut(NCut/CC) on CUB-200. DeepCut segments the birds accurately without including other objects such as branches and leaves (which is a common failure point of previous methods).