Image Segmentation

Computer Vision

Assignment 2

Overview

- Task 1: implement Mean-Shift to segment image (40 pts).
- Task 2: implement simplified SegNet on multi-digit MNIST dataset (60 pts).

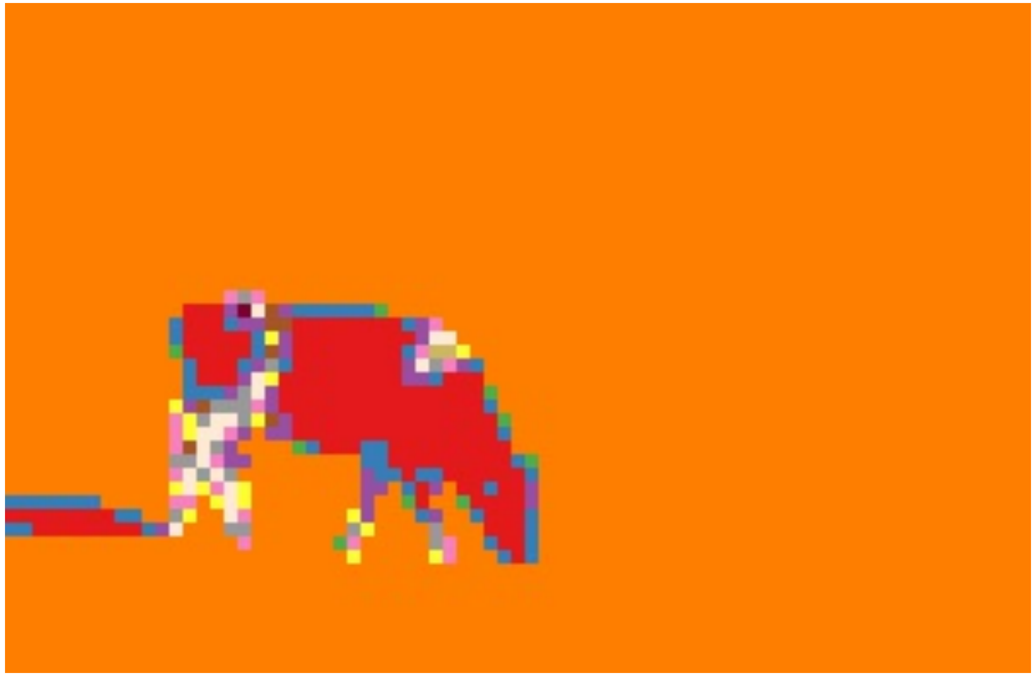
Mean-Shift

- Input image

Input Image

Mean-Shift

- Goal – segment the image in CIELAB color space (preprocessing already provided in code)



Expected Segments

Mean-Shift

- In each step, for each point:
  - Compute the distances from this point to all points (including current point) within a radius. In this assignment we set this radius to positive infinity so that you can have an easier implementation
  - Compute weight of every point as a Gaussian kernel function of distance, with bandwidth=2.5
  - Compute weighted mean of all points (including current point), then update current point with this weighted mean

Mean-Shift – Implementation Details

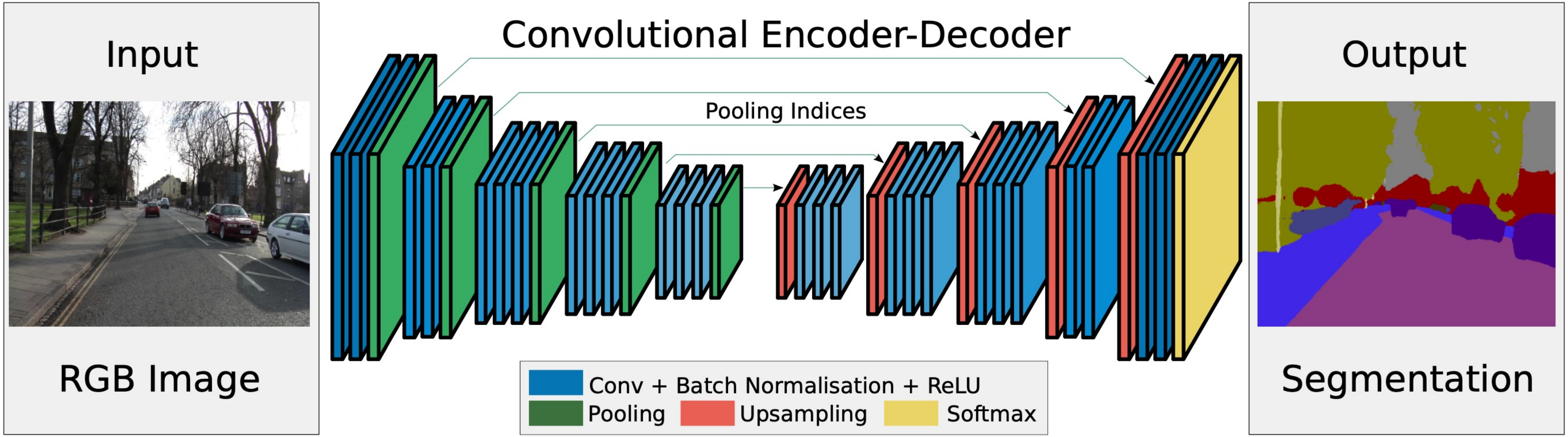
- Within each mean-shift step, you should update all points **out of place**, i.e the order of update should not affect the final output.
- Run 20 steps.
- After implementing the basic mean-shift algorithm, try to accelerate it by taking advantage of batch processing.

Mean-Shift

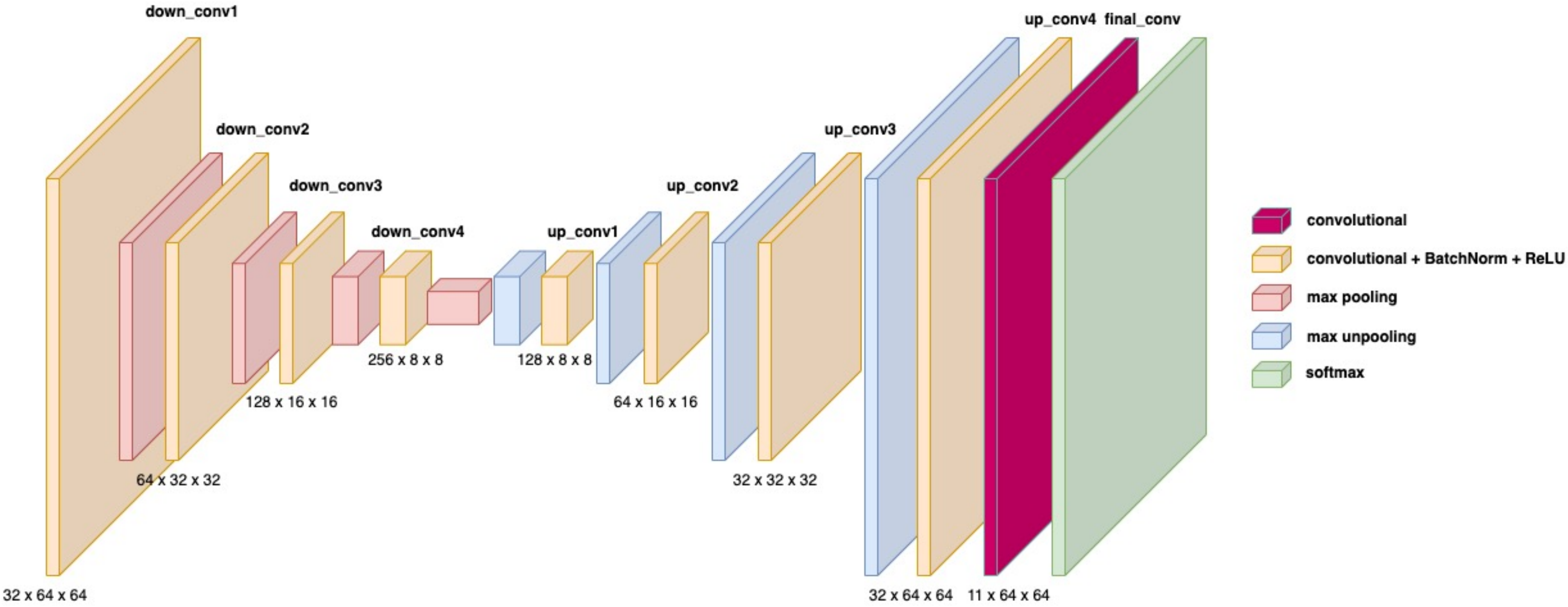
- Pseudo code

```
while not_converged():
    for i, point in enumerate(points):
        # distance for the given point to all points
        distances = distance(point, points)

        # turn distance into weights using a gaussian
        weights = gaussian(dist, bandwidth=2.5)

        # update the point by calculating weighted mean of all points
        points[i] = update_point(weight, X)

return points
```

SegNet – From the Lecture



SegNet – Simplified Version



Basic Modules

- Conv2d, BatchNorm2d, MaxPool2d and MaxUnpool2d

Basic Modules – BatchNorm2d

https://pytorch.org/docs/stable/generated/torch.nn.BatchNorm2d.html

CLASS `torch.nn.BatchNorm2d(`*`num_features, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True, device=None, dtype=None`*`)` [SOURCE]

Applies Batch Normalization over a 4D input (a mini-batch of 2D inputs with additional channel dimension) as described in the paper Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift .

$$y = \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x] + \epsilon}} * \gamma + \beta$$

The mean and standard-deviation are calculated per-dimension over the mini-batches and γ and β are learnable parameter vectors of size C (where C is the input size). By default, the elements of γ are set to 1 and the elements of β are set to 0. The standard-deviation is calculated via the biased estimator, equivalent to *torch.var*(*input, unbiased=False*).

Also by default, during training this layer keeps running estimates of its computed mean and variance, which are then used for normalization during evaluation. The running estimates are kept with a default `momentum` of 0.1.

If `track_running_stats` is set to `False`, this layer then does not keep running estimates, and batch statistics are instead used during evaluation time as well.

Basic Modules – MaxUnpool2d

https://pytorch.org/docs/stable/generated/torch.nn.MaxUnpool2d.html

CLASS `torch.nn.MaxUnpool2d(kernel_size, stride=None, padding=0)` [SOURCE]

Computes a partial inverse of `MaxPool2d`.

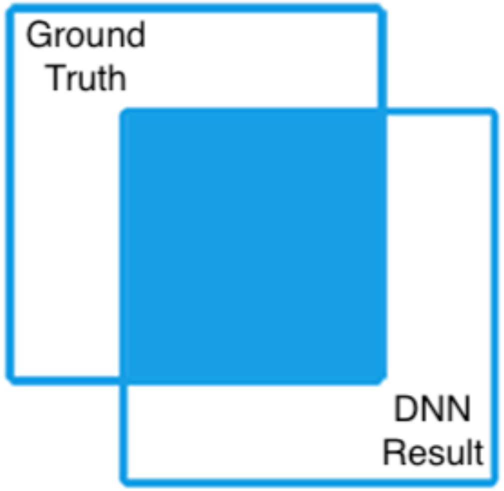
`MaxPool2d` is not fully invertible, since the non-maximal values are lost.

`MaxUnpool2d` takes in as input the output of `MaxPool2d` including the indices of the maximal values and computes a partial inverse in which all non-maximal values are set to zero.

Loss – Per-pixel Cross-Entropy Loss

```python
class CrossEntropy2D(nn.Module):
    def __init__(self, ignore_index, reduction='mean', weight=None):
        """Initialize the module

        Args:
            ignore_index: specify which the label index to ignore.
            reduction (str): reduction method. See torch.nn.functional.cross_entropy for details.
            output_dir (str): output directory to save the checkpoint
            weight: weight for samples. See torch.nn.functional.cross_entropy for details.
        """
        super(CrossEntropy2D, self).__init__()
        self.weight = weight
        self.ignore_index = ignore_index
        self.reduction = reduction

    def forward(self, output, target, resize_scores=True):
        """Forward pass of the loss function

        Args:
            output (torch.nn.Tensor): output logits, i.e. network predictions w.o. softmax activation.
            target (torch.nn.Tensor): ground truth labels.
            resize_scores (bool): if set to True, when target and output have different widths or heights,
                                  upsample output bilinearly to match target resolution. Otherwise, downsample
                                  target using nearest neighbor to match input.
        Returns:
            loss (torch.nn.Tensor): loss between output and target.
        """
        _assert_no_grad(target)

        b, c, h, w = output.size()
        tb, th, tw = target.size()

        assert(b == tb)

        # Handle inconsistent size between input and target
        if resize_scores:
            if h != th or w != tw:  # upsample logits
                output = nn.functional.interpolate(output, size=(th, tw), mode="bilinear", align_corners=False)
        else:
            if h != th or w != tw:  # downsample labels
                target = nn.functional.interpolate(target.view(b, 1, th, tw).float(), size=(h, w), mode="nearest").view(b, h, w).long()


        loss = nn.functional.cross_entropy(
            output, target, weight=self.weight, ignore_index=self.ignore_index, reduction=self.reduction
        )

        return loss
```

Evaluation Metric – Intersection Over Union



$$\mathrm{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$

SegNet

- Dataset – multi-digit MNIST

Input Image (64x64)

GT Labels

Summary

- Materials will be available later today.
- Assignment 2 is due 05.11.2021 at midnight (11:59pm)