Warning: Do not delete slides.
This includes extra credit slides and any problems you do not complete. All problems, including extra credit, must be assigned to a slide on Gradescope. Failure to follow this will result in a penalty

CS 6476 Project 4

Auryn Yamamura

ayamamura6@gatech.edu

ayamamura6

904154249

Parts 4 & 5: mIoU of different models

Add each of the following (keeping the changes as you move to the next row):

	Training mIoU	Validation mIoU
Simple Segmentation Net (no pretrained weights)	0.3585	0.3192
+ ImageNet-Pretrained backbone	0.4559	0.4519
+ Data augmentation	0.3867	0.4113
ImageNet-Pretrained PSPNet w/ Data Aug. without PPM	0.5962	0.6139
+ PSPNet with PPM	0.5927	0.6067
+ PSPNet with auxiliary loss	0.5961	0.6167

Parts 4 & 5: Per class IoUs

Report your model's IoU for the 11 Camvid classes (you can find the order they are listed in at dataset_lists/camvid-11/camvid-11_names.txt):

Class Index	Class name	Simple Segmentation Net Class IoU	PSPNet Class IoU
0	Building	0.8117	0.9012
1	Tree	0.7958	0.8794
2	Sky	0.7716	0.9237
3	Car	0.4678	0.8546
4	SignSymbol	0.0000	0.0000
5	Road	0.8733	0.9467
6	Pedestrian	0.0300	0.2939
7	Fence	0.3632	0.5991
8	Column_Pole	0.0000	0.0002
9	Sidewalk	0.6851	0.8326
10	Bicyclist	0.0000	0.6284

Parts 4 & 5: Most difficult classes

[Which classes have the lowest mIoU? Why might they be the most difficult? Provide an example RGB image from Camvid that illustrates your point]
Sign Symbol and Column Pole have the lowest mIoU. This is most likely because these classes tend occupy a small number of pixels, so as the images gets passed through layers it's likely that info of classes tends to get lost.

In the image to the right, we can see that the column poles/sign symbols are extremely small and overlap with classes like 'sky' and 'tree', both of which comparatively occupy many more pixels in the images. Due to this and loss of spatial information in the network, it is very likely that the column poles/sign symbols got misclassified as those larger classes.

Parts 4 & 5: ImageNet Pretraining

[Compare the performance of the network trained from scratch vs. the one with an ImageNet-pretrained backbone. Quantify the difference in final validation mIoU. Why does pretraining on a large-scale classification dataset like ImageNet typically improve performance on a different task like semantic segmentation]

$$\frac{|\text{scratch} - \text{pretrained}|}{\text{scratch}} = \frac{|0.3192 - 0.4519|}{0.3192} = \frac{0.1327}{0.3192} = 0.4157$$

Using the ImageNet pretrained network, we see a 41.57% increase in the validation mIoU.

Pretraining on a large-scale classification dataset like ImageNet allows the network to develop filters for object detection that also aid in semantic segmentation.

Parts 4 & 5: Data Augmentation

[Analyze the effect of adding data augmentation. Did it improve validation mIoU? Discuss how augmentation helps the model generalize better and avoid overfitting. Mention at least two types of augmentation you believe are most impactful for this task.]

Based on my results, it seems like data augmentation worsen the validation mIoU of the model (this may be due to the fact that I based my results on 50 epochs instead of 100). In general though, data augmentation prevents the model from learning trivial shortcuts, which may work for a large amount of images but do not generalize to the whole dataset. In particular:

- Color Jitter: prevents our network from using chromatic aberration to classify different regions of the image
- Rand Scale/Crop: forces the network to not ignore classes that tend to occupy smaller regions of the image (i.e. ‘car’, ‘pedestrian’, ‘bicyclist’, etc).

Parts 4 & 5: Dilated Convolutions

[The PSPNet backbone uses dilated convolutions. Explain the primary benefit of using dilation in the ResNet backbone for a dense prediction task like segmentation. How does it allow the network to increase its receptive field without losing spatial resolution?]

Dilated convolutions let us convolve our filters over a larger window than just the filter size, allowing us to increase the receptive fields of our layers and giving the network more global context when making its decisions.

Parts 4 & 5: Auxiliary Loss

[Analyze the contribution of the auxiliary loss. How does adding a secondary loss function partway through the network assist in the training of such a deep model? Did you observe it leading to faster convergence or a higher final mIoU]

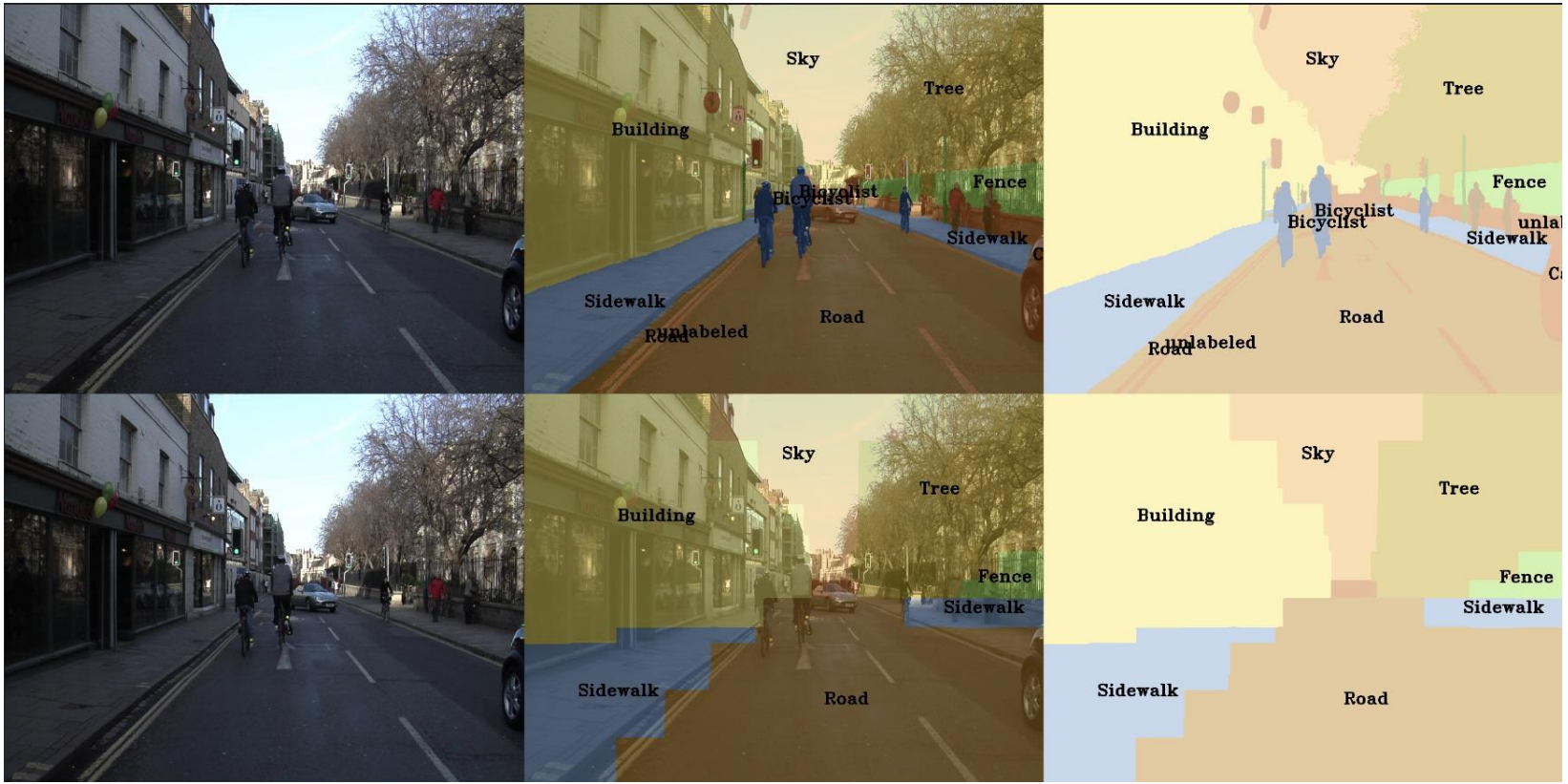
We're progressively losing more information as our network gets deeper due to downsampling. This causes vanishing gradients when backpropagating leading to the early/intermediate layers not updating much during training. Auxiliary loss remedies the vanishing gradients since it operates on shallower "window" of our model, allowing the early/intermediate layers to get more meaningful updates to their weights.

As for this project, while auxiliary loss did not increase mIoU by a significant amount, it did result in our model converging much more quickly: w/o aux_loss, the model converges within ~45 epochs; w/ aux_loss, the model converges within ~35 epochs. (Loss records from model shown below)

PSPNet (no aux_loss): [1.052, 0.651, 0.54, 0.487, 0.458, 0.412, 0.408, 0.378, 0.37, 0.321, 0.334, 0.317, 0.312, 0.324, 0.304, 0.288, 0.295, 0.284, 0.289, 0.277, 0.27, 0.275, 0.267, 0.264, 0.268, 0.253, 0.258, 0.25, 0.247, 0.236, 0.238, 0.244, 0.254, 0.238, 0.237, 0.237, 0.233, 0.227, 0.231, 0.229, 0.226, 0.235, 0.229, 0.234, 0.221, 0.224, 0.22, 0.224, 0.231, 0.227]

PSPNet (aux_loss): 1.02, 0.623, 0.519, 0.459, 0.45, 0.413, 0.367, 0.374, 0.357, 0.334, 0.316, 0.308, 0.308, 0.3, 0.307, 0.292, 0.282, 0.27, 0.268, 0.275, 0.271, 0.269, 0.253, 0.251, 0.248, 0.254, 0.236, 0.241, 0.236, 0.235, 0.219, 0.231, 0.237, 0.233, 0.224, 0.22, 0.231, 0.225, 0.22, 0.219, 0.223, 0.211, 0.216, 0.216, 0.207, 0.221, 0.211, 0.218, 0.209, 0.22]

Part 4: Simple segmentation net qualitative results



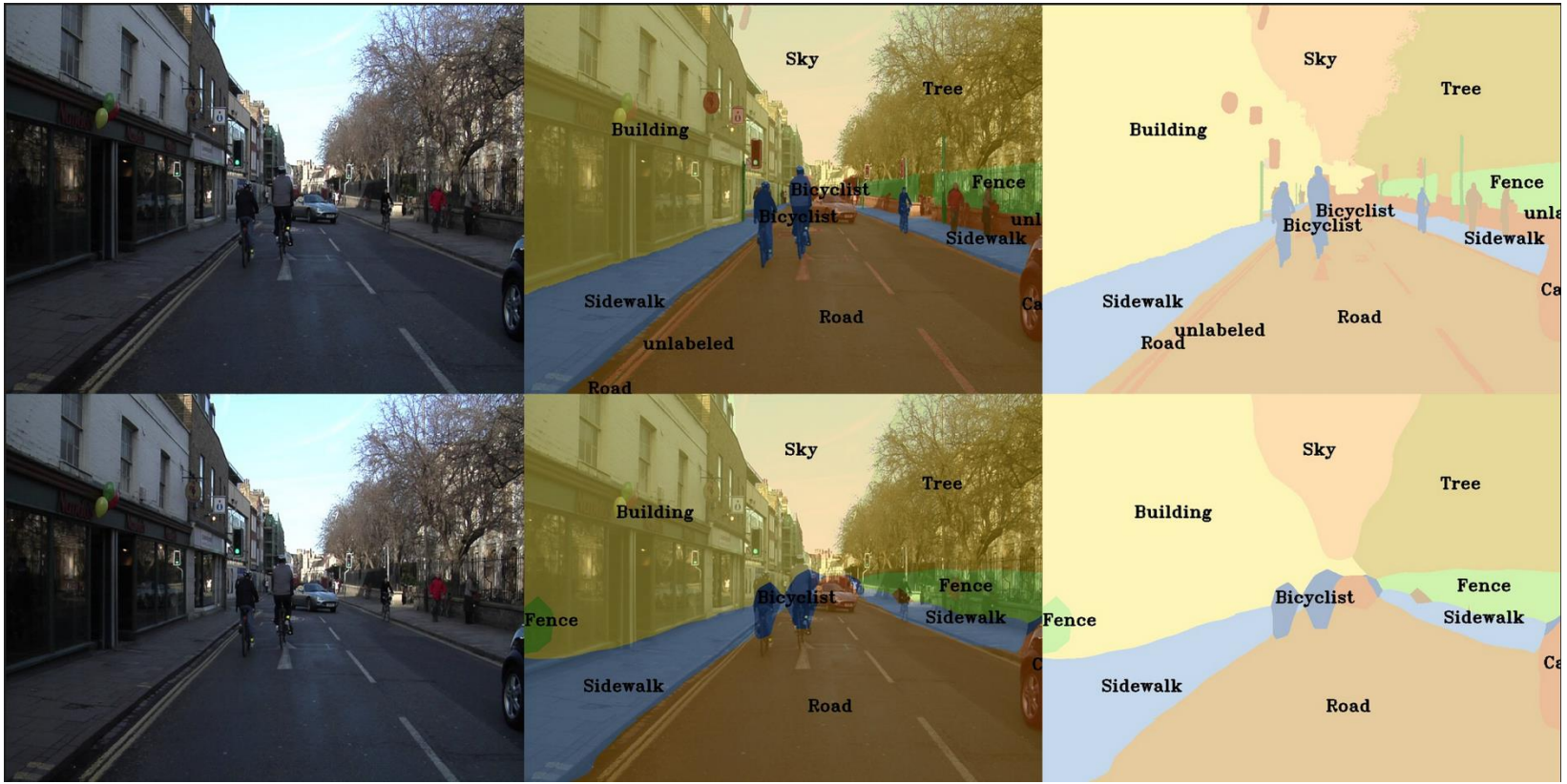
Part 4: Qualitative Analysis of Segmentation Errors

[Identify a specific object or region that is imperfectly segmented and describe the primary type of error.]

Most of the regions in our SimpleSeg prediction end up block-y which we can describe as a sort of “bleeding” error. Take the boundary between the sidewalk and road in the lower left region of our image: we can see that there are multiple of sections of the road are improperly labeled as ‘sidewalk’.

This is most likely due to the loss of spatial information as our input progresses through the network. In other words, it is possible that sections of the road got ”overwritten” as ’sidewalk’ (or ‘sidewalk’ information “bled” into ‘road’ sections) since our receptive fields are shrinking as we progress through the network. Additionally sidewalks and roads do look similar, so without the global context, it would be difficult to parse instance of the two apart.

Part 5: PSPNet qualitative results



Part 5: Revisiting Segmentation Errors

[Directly compare the new result to the previous one. Was the specific error you identified earlier corrected, partially improved, or did it remain? Explain why you think the full PSPNet architecture succeeded or failed to fix this particular error.]

The specific error I mentioned in my previous slide is corrected by PSPNet. I suspect that by adding the PPM module and dilated convolutions, we forcibly increased our receptive fields which gave the network the global context it needed to make a better distinct between the sidewalk and road.

Part 6: Transfer Learning (Extra Credit)

Report your model's IoU for the Kitti Dataset.

	mIoU	mAcc/	allAcc
Train result	0.9395	0.9677	0.9809
Val result	0.9222	0.9563	0.9759

Class Index	Class name	iou	accuracy
0	Road	0.8734	0.9257
1	Not_Road	0.9710	0.9869

Part 6: Transfer Learning (Extra Credit)

Compare the training loss generated when training on Kitti dataset and Camvid dataset. Which decreases at a faster rate? If Camvid or Kitti training loss decreases at a faster rate than the other, why do you think this happened? Or, if the loss decreases at a similar rate, why do you think that is so?

Camvid ('loss_train'): [1.02, 0.623, 0.519, 0.459, 0.45, 0.413, 0.367, 0.374, 0.357, 0.334, 0.316, 0.308, 0.308, 0.3, 0.307, 0.292, 0.282, 0.27, 0.268, 0.275, 0.271, 0.269, 0.253, 0.251, 0.248, 0.254, 0.236, 0.241, 0.236, 0.235, 0.219, 0.231, 0.237, 0.233, 0.224, 0.22, 0.231, 0.225, 0.22, 0.219, 0.223, 0.211, 0.216, 0.216, 0.207, 0.221, 0.211, 0.218, 0.209, 0.22]

Kitti ('loss_train'): [0.229, 0.126, 0.1, 0.076, 0.063, 0.062, 0.059, 0.06, 0.052, 0.055, 0.048, 0.048, 0.047, 0.05, 0.046, 0.046, 0.048, 0.048, 0.048, 0.047]

Above is my model's training loss from both Camvid and Kitti. If we're evaluating absolute loss across epochs (rather than percent/relative loss), then the Camvid training loss decreases at a faster rate than Kitti. Our training loss on the Camvid dataset is initially large but rapidly decreases because we're training parts of our model from scratch (i.e. ppm, cls, and aux layers).

Conversely, the loss decreases at a much slower rate for Kitti since we're using our model pre-trained on Camvid and the tasks for Camvid and Kitti datasets are very similar. In short, our Kitti training loss decreases more slowly since there's not much more we need to optimize for.