**COT 5930 Digital Image Processing - Spring 2021**
**Assignment 4**
**Christian Garbin**

This report has two parts:

1. Answers to the questions: answers to the questions in the assignment.
2. Evaluating different solutions: a summary of the optional part of the assignment, attempting a different solution. The details of the attempted solution are in the live script.

## Answers to the questions

**QUESTION 1: What is the difference between an imageDatastore and a pixelLabelDatastore in MATLAB?**

imageDatastore handles a collection of regular image files, while pixelLabelDatastore handles the semantic segmentation of images. The output of an imageDatastore instance is an image file. The output of a pixelLabelDatastore is the label assigned to the pixels of an image.

**QUESTION 2: What are some of the main characteristics of the CamVid dataset? Think about size, historical context, images, labels, etc.**

*Description*

CamVid stands for "Cambridge-driving Labeled Video Database (CamVid)". It claims to be "the first collection of video object class semantic labels" (from its website).

The first citation on the website is from 2008, thus the dataset is about 13 years old.

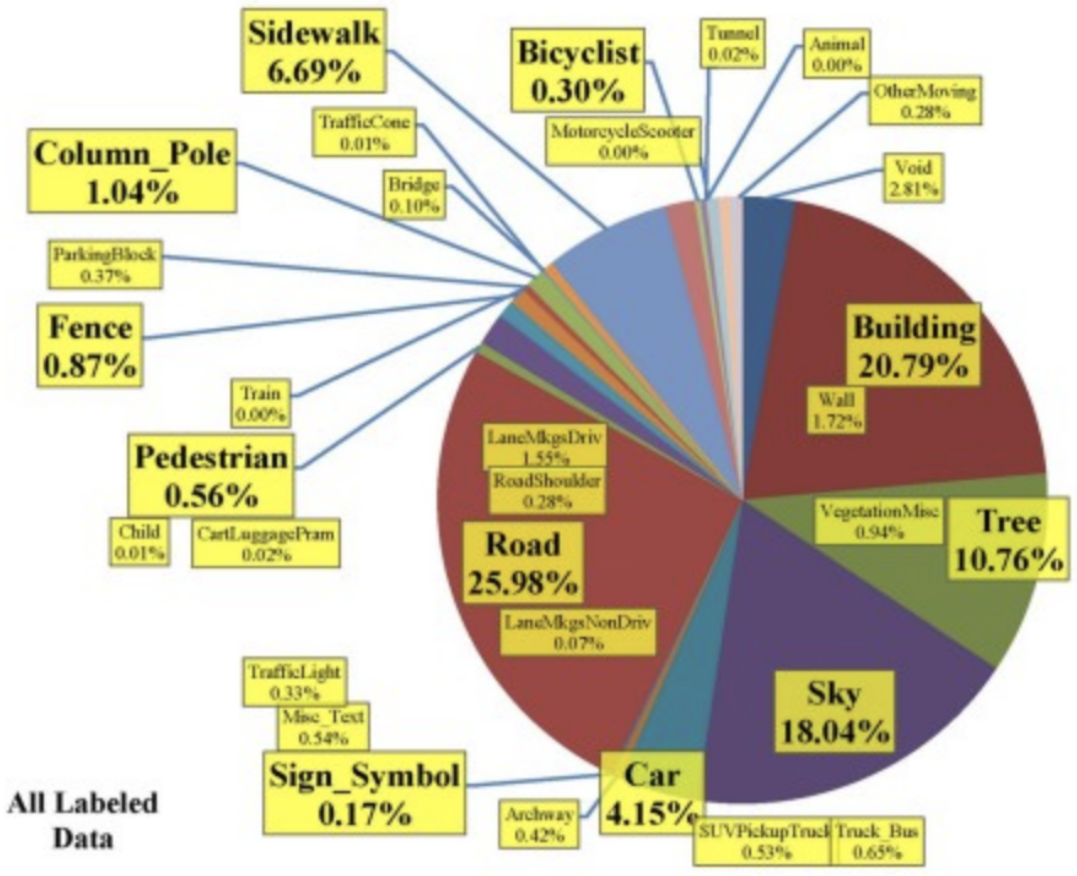
*Contents*

The images are captured from the point of view of a driver of a car. There are ten minutes of video, split into five sequences. Out of these videos, 700 images are semantically segmented at the pixel level (segmented by one person, verified by another).

The dataset has 32 labels:

| **Moving object** | **Road** | **Ceiling** | **Fixed object** |
|---|---|---|---|
| Animal | Road == drivable surface | Sky | Building |
| Pedestrian | Shoulder | Tunnel | Wall |
| Child | Lane markings drivable | Archway | Tree |
| Rolling cart/luggage/pram | Non-Drivable | | Vegetation misc. |
| Bicyclist | | | Fence |
| Motorcycle/scooter | | | Sidewalk |
| Car (sedan/wagon) | | | Parking block |
| SUV / pickup truck | | | Column/pole |
| Truck / bus | | | Traffic cone |
| Train | | | Bridge |
| Misc | | | Sign / symbol |
| | | | Misc text |
| | | | Traffic light |
| | | | Other |

The distribution of the labels is shown in the picture below, taken for the CamVid website.



*Possible limitations*

- *Class imbalance*: The picture shows that the dataset is imbalanced, with a few classes dominating the number of labeled pixels. This reflects the physical world but may be a problem when training a model on this data.
- *Geographical bias*: The road scenes seem to have been filmed in the UK, as we may guess from the names. Some specific traffic features, such as road marking for traffic-calming zones, may apply only to British roads (specifically the classes "Lane markings drivable" and "Non-Drivable"). Other classes seem to be generalizable, at least for western countries.
- *Urban bias*: Most of the scenes are in a city, with generally good light conditions. This may affect the usability of the dataset for rural roads and other light conditions, e.g. dusk, dawn, night, rain, or snow.

**QUESTION 3: How are undefined or void labels handled in the example code? Would you suggest handling it differently, perhaps? Why (not)?**

They are defined as [0 0 0]. It seems to be a sensible choice because it is easy to visualize (maps to the color black) and makes the code to identify them slightly simpler and perhaps even a bit faster. For example, because 0 (zero) is "false", we can write "if ! ..." instead of "if <some value>" to create a mask with all void and undefined pixels.

**QUESTION 4: What are the main insights provided by plotting the histogram of the distribution of pixel counts by class (step 2)?**

It shows that the dataset is imbalanced. Some classes, like roads, buildings, and sky, have a large number of pixels, while others, like pole, signs, pedestrian, and bicyclist, have a small number of pixels. Training a self-driving car on this dataset is probably not a good idea. It will likely stay on the road but hit pedestrians and bicyclists.

**QUESTION 5: If you could refactor/improve the auxiliary function partitionCamVidData, what would you do differently?**

I would pass the distribution of training, validation, and test sets as parameters. The size of the test (20%) seems a bit larger, reducing the number of images we can use for training. An option to stratify the split, so classes are equally represented in all sets.

**QUESTION 6: How is the issue of class imbalance handled in the example code? Would you suggest handling it differently, perhaps? Why (not)?**

It is handled using class weighting, with median frequency balancing. Other methods could be used in addition to that:

1. Oversampling of the pictures with the less-common classes.
2. Favor images with less-common classes when generating augmented images (generate more of those) - a variation of oversampling.
3. Use a loss function specifically designed to deal with class imbalance (some references here).
4. Use a network architecture specifically designed for segmentation (some examples here).

**QUESTION 7: How is image data augmentation implemented in the example code? Would you suggest doing it differently, perhaps? If so, how exactly?**

It is implemented with random reflection and X/Y translations. We could use other transformations, for example, rotation and scaling, to try to improve the training process.

More sophisticated approaches use GANs to generate additional (realistic) examples (reference).

**QUESTION 8: During quantitative evaluation of the segmentation results, which classes show the best performance across all metrics? Do these match your expectations? Why (not)?**

The top three classes by mean BF score are sky, road, and pavement. Sky and road match the expectation, but I was expecting to see buildings in the top three because it is more represented in the dataset than pavement.

**QUESTION 9: During quantitative evaluation of the segmentation results, which classes show the greatest discrepancy between accuracy and IoU? How do you explain that?**

The greatest discrepancies between accuracy and IoU are for the classes pedestrian, fence, sign symbol, and pole. These are the classes with the fewer instances (pixels) in the dataset, thus it is easy to get their "true negative" value right, driving up accuracy. IoU, on the other hand, measures the overlap of the ground truth, i.e. only the expected segment of the image that contains the class, not the whole image. Because these classes are less represented in the dataset, it is reasonable to expect that a measure that considers only their pixels would be smaller.

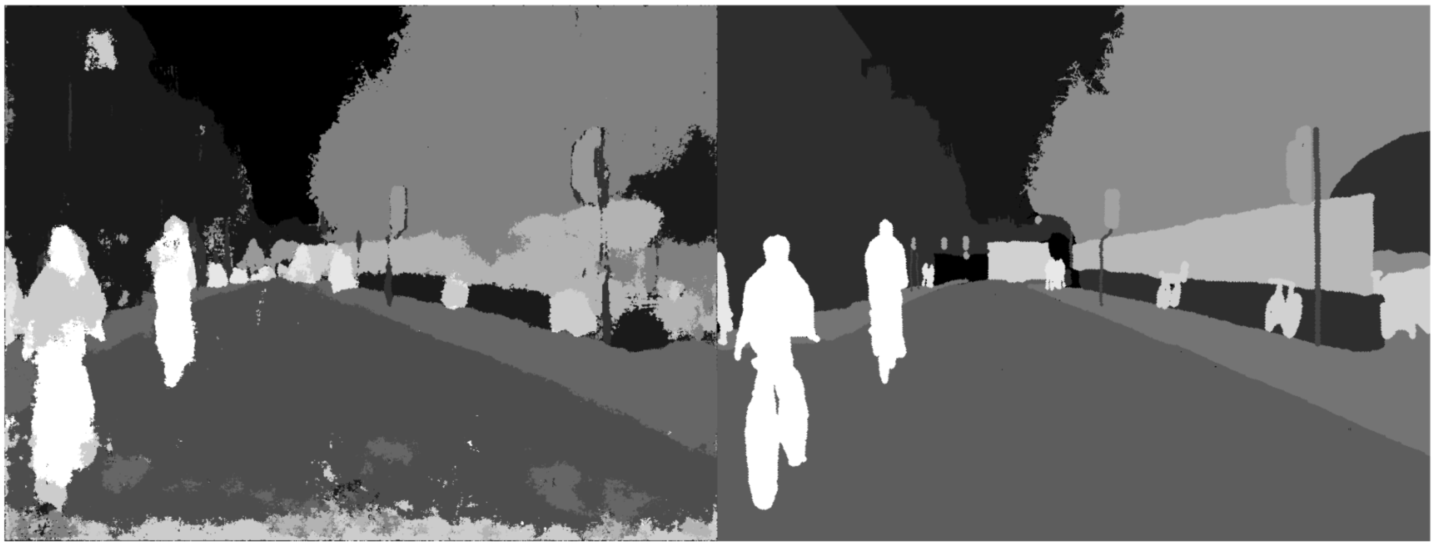
## Evaluating different solutions

In this section, we evaluate segmentation with a VGG16 pretrained network.

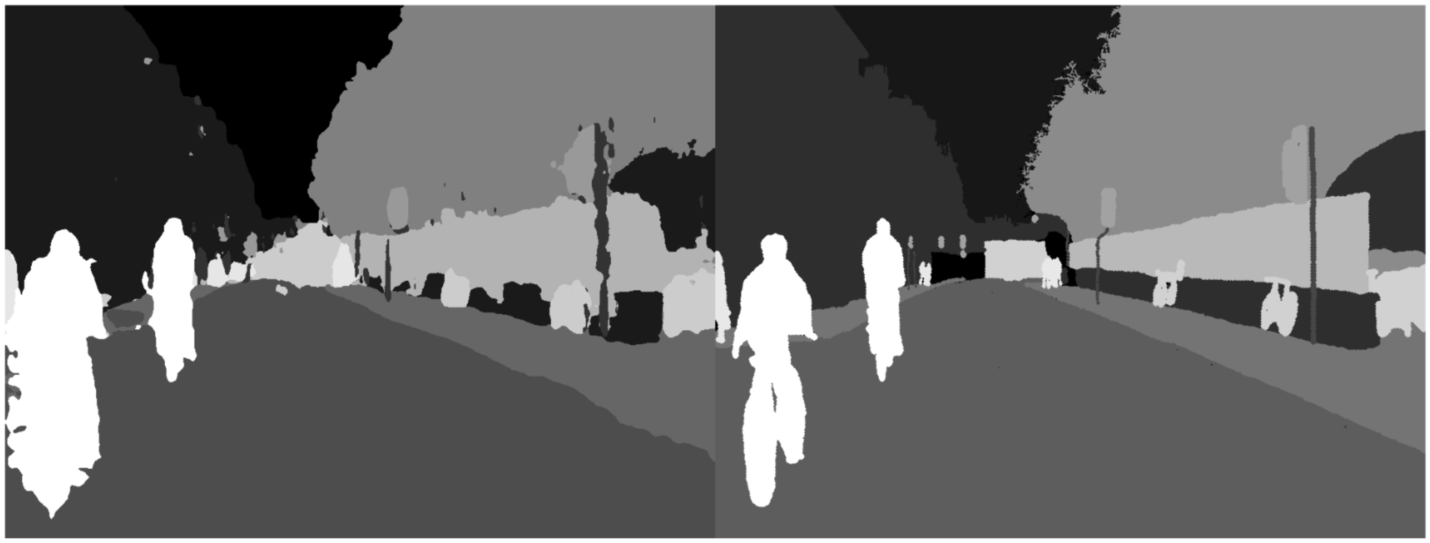
```
% From
https://blogs.mathworks.com/deep-learning/2018/06/08/semantic-segmentation-using-deep-learn
ing/
pretrainedURL = 'https://www.mathworks.com/supportfiles/vision/data/segnetVGG16CamVid.mat';
...
```

Despite being twice as large as the ResNet18 network used in the starter code (107 MB vs. 48 MB), the VGG 16 network did not perform as well. A possible reason may be overfitting, caused by the much larger network and the relatively small dataset. This needs to be investigated further.

On the left side, we see the segmentation it predicted. On the right side, we see the ground truth.

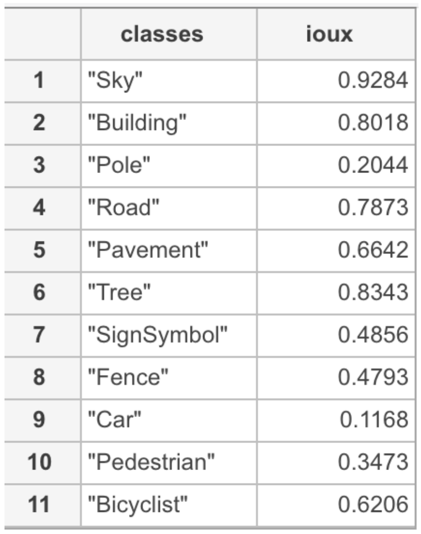
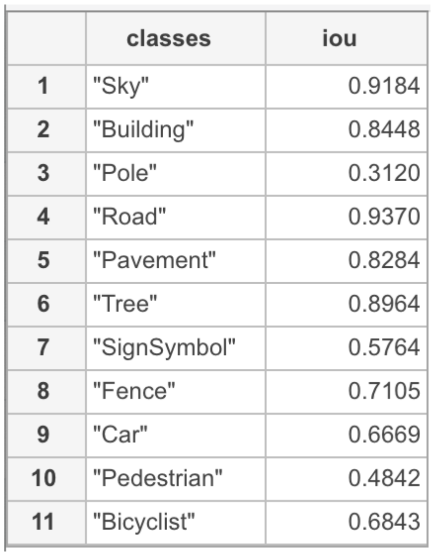


For comparison, this the same image with the result from the starter code, using ResNet18. We can see that it does a better job overall.



However, for some specific cases, it seems to perform better. More specifically, it seems to be better at detecting boundaries. For example, the boundaries of the cyclists are better defined.

The following table shows the ioU for this image, compared with the ResNet18 metrics.

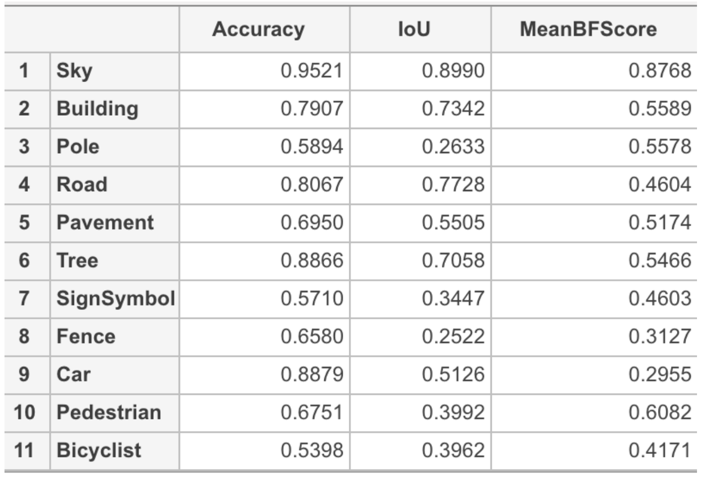
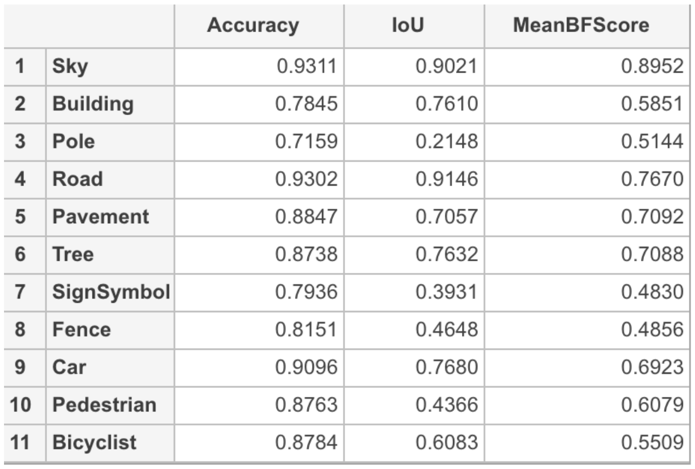
| VGG16 | ResNet18 |
|---|---|
|  |  |

| | classes | ioux |
|---|---|---|
| 1 | "Sky" | 0.9284 |
| 2 | "Building" | 0.8018 |
| 3 | "Pole" | 0.2044 |
| 4 | "Road" | 0.7873 |
| 5 | "Pavement" | 0.6642 |
| 6 | "Tree" | 0.8343 |
| 7 | "SignSymbol" | 0.4856 |
| 8 | "Fence" | 0.4793 |
| 9 | "Car" | 0.1168 |
| 10 | "Pedestrian" | 0.3473 |
| 11 | "Bicyclist" | 0.6206 |

| | classes | iou |
|---|---|---|
| 1 | "Sky" | 0.9184 |
| 2 | "Building" | 0.8448 |
| 3 | "Pole" | 0.3120 |
| 4 | "Road" | 0.9370 |
| 5 | "Pavement" | 0.8284 |
| 6 | "Tree" | 0.8964 |
| 7 | "SignSymbol" | 0.5764 |
| 8 | "Fence" | 0.7105 |
| 9 | "Car" | 0.6669 |
| 10 | "Pedestrian" | 0.4842 |
| 11 | "Bicyclist" | 0.6843 |

The following table shows overall class metrics with the test set. Overall, the scores are lower for VGG16.

This may be another case where it is more important to improve the dataset, not the model.

| VGG16 | ResNet18 |
|---|---|
|  |  |

| | | Accuracy | IoU | MeanBFScore |
|---|---|---|---|---|
| 1 | Sky | 0.9521 | 0.8990 | 0.8768 |
| 2 | Building | 0.7907 | 0.7342 | 0.5589 |
| 3 | Pole | 0.5894 | 0.2633 | 0.5578 |
| 4 | Road | 0.8067 | 0.7728 | 0.4604 |
| 5 | Pavement | 0.6950 | 0.5505 | 0.5174 |
| 6 | Tree | 0.8866 | 0.7058 | 0.5466 |
| 7 | SignSymbol | 0.5710 | 0.3447 | 0.4603 |
| 8 | Fence | 0.6580 | 0.2522 | 0.3127 |
| 9 | Car | 0.8879 | 0.5126 | 0.2955 |
| 10 | Pedestrian | 0.6751 | 0.3992 | 0.6082 |
| 11 | Bicyclist | 0.5398 | 0.3962 | 0.4171 |

| | | Accuracy | IoU | MeanBFScore |
|---|---|---|---|---|
| 1 | Sky | 0.9311 | 0.9021 | 0.8952 |
| 2 | Building | 0.7845 | 0.7610 | 0.5851 |
| 3 | Pole | 0.7159 | 0.2148 | 0.5144 |
| 4 | Road | 0.9302 | 0.9146 | 0.7670 |
| 5 | Pavement | 0.8847 | 0.7057 | 0.7092 |
| 6 | Tree | 0.8738 | 0.7632 | 0.7088 |
| 7 | SignSymbol | 0.7936 | 0.3931 | 0.4830 |
| 8 | Fence | 0.8151 | 0.4648 | 0.4856 |
| 9 | Car | 0.9096 | 0.7680 | 0.6923 |
| 10 | Pedestrian | 0.8763 | 0.4366 | 0.6079 |
| 11 | Bicyclist | 0.8784 | 0.6083 | 0.5509 |