Homework 4

ECE 253
Digital Image Processing

November 23, 2020

Make sure you follow these instructions carefully during submission :

• Homework 4 is due by 11:59 PM, December 17, 2020.
• You should avoid using loops in your MATLAB/Python code unless you are explicitly per-

mitted to do so.
• Submit your homework electronically by following the two steps listed below:

1. Upload a pdf file with your write-up on Gradescope. This should include your answers to each question and relevant code snippet. Make sure the report mentions your full name and PID. Finally, carefully read and include the following sentences at the top of your report:

Academic Integrity Policy: Integrity of scholarship is essential for an academic commu- nity. The University expects that both faculty and students will honor this principle and in so doing protect the validity of University intellectual work. For students, this means that all academic work will be done by the individual to whom it is assigned, without unauthorized aid of any kind.

By including this in my report, I agree to abide by the Academic Integrity Policy men- tioned above.

2. Upload a zip file with all your scripts and files on Gradescope. Name this file:
ECE 253 hw4 lastname studentid.zip. This should include all files necessary to run your code out of the box.

1

Problem 1. Hough Transform (10 points)
(The first two parts of this problem are borrowed from Professor Belongie's Image Processing

course.)

1. (i) Implement the Hough Transform (HT) using the (ρ, θ) parameterization as described in GW Third Edition p. 733-738 (see 'HoughTransform.pdf' provided in the data folder). Use accumulator cells with a resolution of 1 degree in θ and 1 pixel in ρ.
2. (ii) Produce a simple 11 × 11 test image made up of zeros with 5 ones in it, arranged like the 5 points in GW Third Edition Figure 10.33(a). Compute and display its HT; the result should look like GW Third Edition Figure 10.33(b). Threshold the HT by looking for any (ρ, θ) cells that contains more than 2 votes then plot the corresponding lines in (x,y)-space on top of the original image.
3. (iii) Load in the image 'lane.png'. Compute and display its edges using the Sobel operator with default threshold settings, i.e.,

   E = edge(I,'sobel')

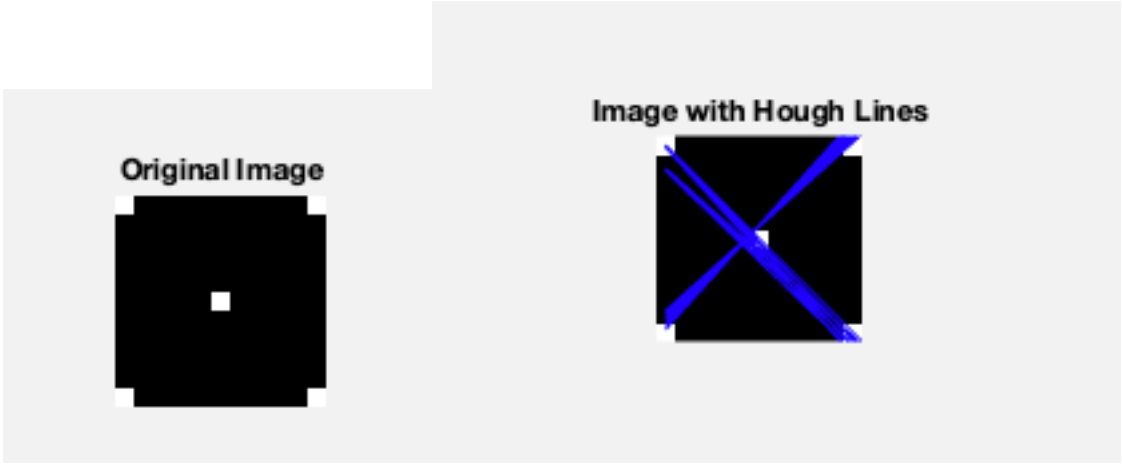
   Now compute and display the HT of the binary edge image E. As before, threshold the HT and plot the corresponding lines atop the original image; this time, use a threshold of 75% maximum accumulator count over the entire HT, i.e. 0.75*max(HT(:)).

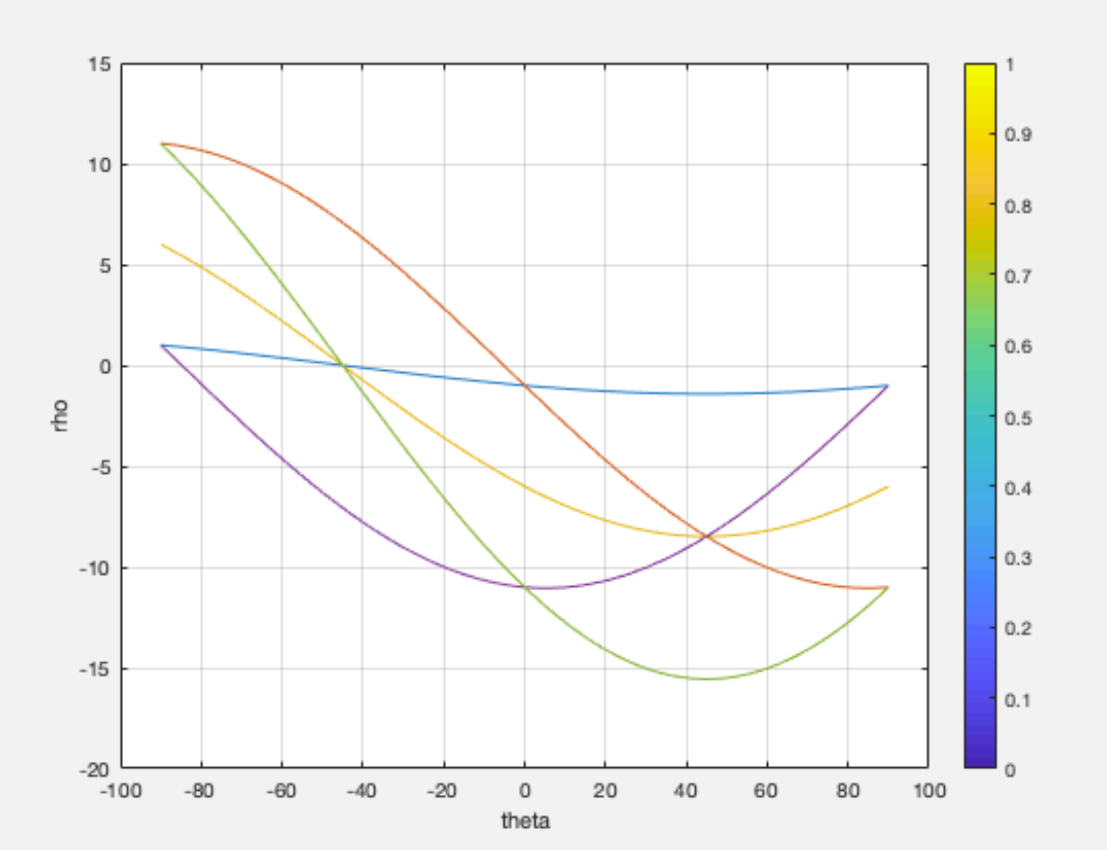
4. (iv) We would like to only show line detections in the driver's lane and ignore any other line detections such as the lines resulting from the neighboring lane closest to the bus, light pole, and sidewalks. Using the thresholded HT from the 'lanes.png' image in the previous part, show only the lines corresponding to the line detections from the driver's lane by thresholding the HT again using a specified range of θ this time. What are the approximate θ values for the two lines in the driver's lane?

Things to include in your report:

- HT images should have colorbars next to them
- Line overlays should be clearly visible (adjust line width if needed)
- HT image axes should be properly labeled with name and values (see Figure 10.33(b) for example)
- 3 images from 2(ii): original image, HT, original image with lines
- 4 images from 2(iii): original image, binary edge image, HT, original image with lines
- 1 image from 2(iv): original image with lines
- θ values from 2(iv)
- Code for 2(i), 2(ii), 2(iii), 2(iv)

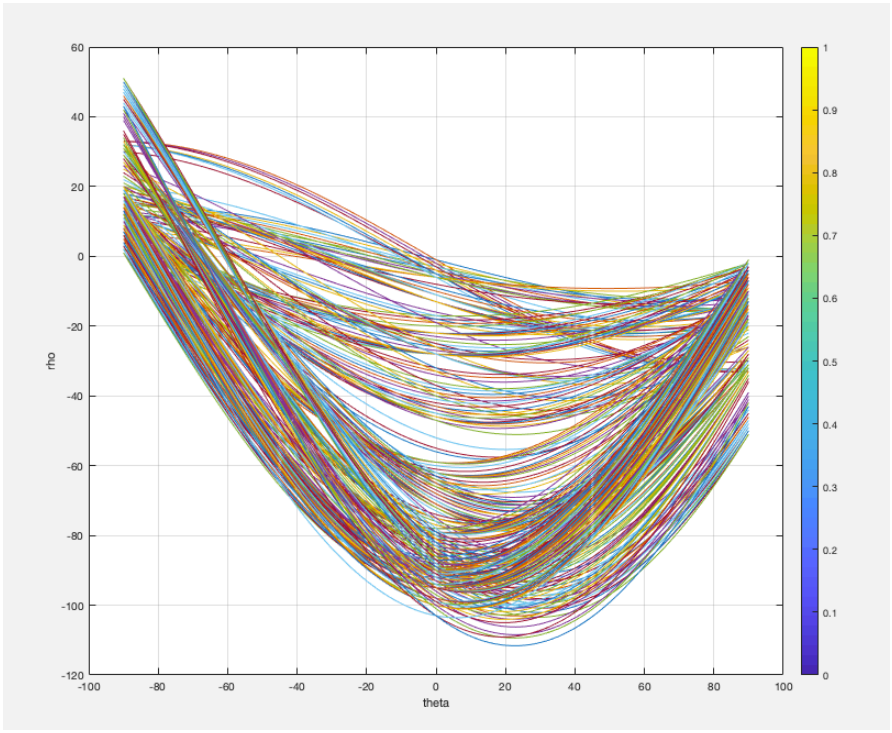
Part 2:

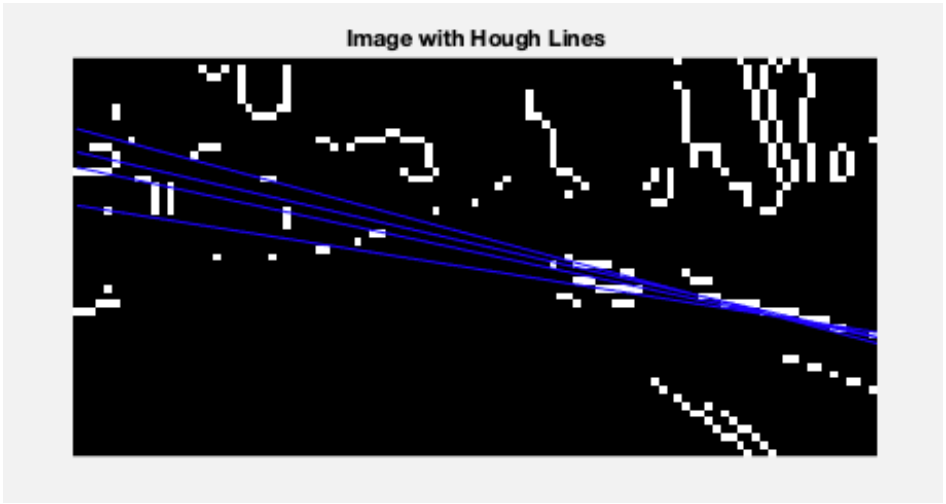




Part 3:

Unfortunately the image was so large that it crashed my laptop when I ran it on anything more than the image scaled down a lot. I understand that these may not be the ideal results, but they're the best I can do with what I have.





Part 4:

Values of Theta: (-60 to 60)

Problem 2. K-Means Segmentation (15 points)
In this problem, we shall implement a K-Means based segmentation algorithm from scratch. To

do this, you are required to implement the following three functions -

- features = createDataset(im) : This function takes in an RGB image as input, and returns a dataset of features which are to be clustered. The output features is an N × M matrix where N is the number of pixels in the image im, and M = 3 (to store the RGB value of each pixel). You may not use a loop for this part.
- [idx, centers] = kMeansCluster(features, centers) : This function is intended to perform K-Means based clustering on the dataset features (of size N × M). Each row in features represents a data point, and each column represents a feature. centers is a k × M matrix, where each row is the initial value of a cluster center. The output idx is an N × 1 vector that stores the final cluster membership ($\in 1, 2, \cdots, k$) of each data point. The output centers are the final cluster centers after K-Means. Note that you may need to set a maximum iteration count to exit K-Means in case the algorithm fails to converge. You may use loops in this function.

  Functions you may find useful : pdist2(), isequal().

- im seg = mapValues(im, idx) : This function takes in the cluster membership vector idx (N × 1), and returns the segmented image im seg as the output. Each pixel in the segmented image must have the RGB value of the cluster center to which it belongs. You may use loops for this part.

  Functions that you may find useful : mean(), cat().

  With the above functions set up, perform image segmentation on the image white-tower.png, with the number of clusters, nclusters = 7. To maintain uniformity in the output image, please initialize clusters centers for K-Means as follows -

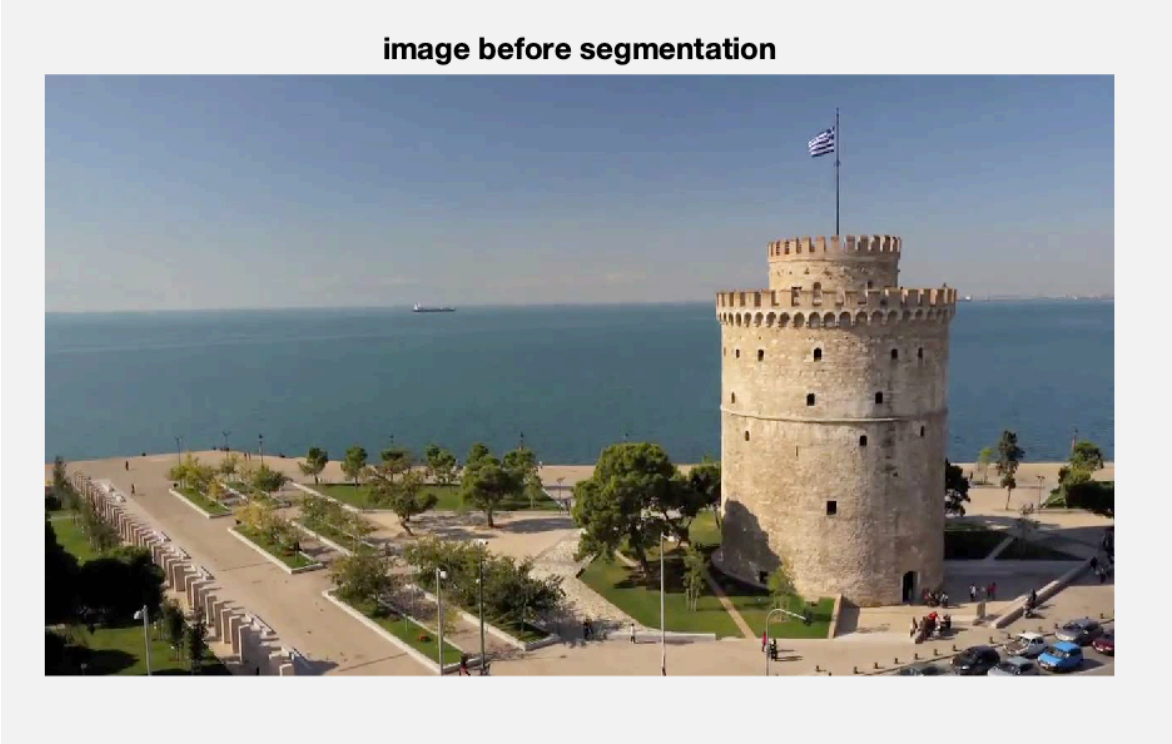
  ```
  rng(5);
  id = randi(size(features, 1), 1, nclusters);
  centers = features(id, :);
  ```

  Things to include in your report:

  • The input image, and the image after segmentation.
  • The final cluster centers that you obtain after K-Means.

  • All your code for this problem.

//image



//image

//image

```
centers =

  138.7021  152.9537  165.8813
   73.0599   99.2435  109.6092
   86.7154   77.1771   57.4341
   33.3953   30.1177   22.2486
  101.1376  126.1952  154.4881
  159.9740  134.1533  110.5623
  204.4274  172.7763  142.9593
```

//image

Matlab code for this problem is in the code appendix to make the report formatting easier on the reader.

Problem 3. Semantic Segmentation (20 points)

In this problem, we will train a fully convolutional network [1] to do semantic segmentation. Most of the code is provided, but after a long day of Digital Image Processing, someone forgot to hit 'save', so part of the network is missing! Your task is to complete and train the network using the CityScape [2] dataset, and to answer the following questions. Please check the README.md for training and testing commands. (And please, help each other out on Piazza if you get stuck!)

1. Please complete the FCN network, the fcn8s in ptsemseg/models/fcn.py. And briefly describe the model structure.
   The model uses 5 sequential blocks with the first 2 having a 2D convolution layer followed by a ReLU layer twice and then ending in a 2D maxPooling layer. The last 3 have a 2D convolution layer followed by a ReLU layer three times and then end in a 2D maxPooling layer. With each sequential block, the number of output channels increases in the first layer.

   With the first sequential block starting with 3 input channels from a 3 channel RGB image and having c1 output channels in the first 2D convolution layer (all over layers of this block have c1 channels as both the input and the output). The first 2D convolution layer in the second sequential block takes c1 input channels and outputs c2 input channels (all over layers of this block have c2 channels as both the input and the output). This continues through sequential block 5. The 6th sequential block is a classifier block with a 2D convolutional layer, followed by a ReLU layer, followed by a 2D dropout layer, likely to prevent overfitting, and follows this order twice. The first time the 2D convolution layer takes c5 input channels and outputs c6, and the second time the 2D convolution layer takes c6 input channels and outputs c7. The final 2D convolution layer of this sequential block takes c7 input channels and outputs n_classes output channels.

2. Do we use weights from a pre-trained model, or do we train the model from scratch?

   ~~We train the model from scratch as no pretrained weights are inputted. VG16 is defined but not called by the model. The training of all the weights takes a long while too (over 6 hours, even when my device running train.py was the barra-Cuda), hinting that there is no pre-trained weights which would cut down on training time. The paper also mentions training the FCN end-to-end.~~

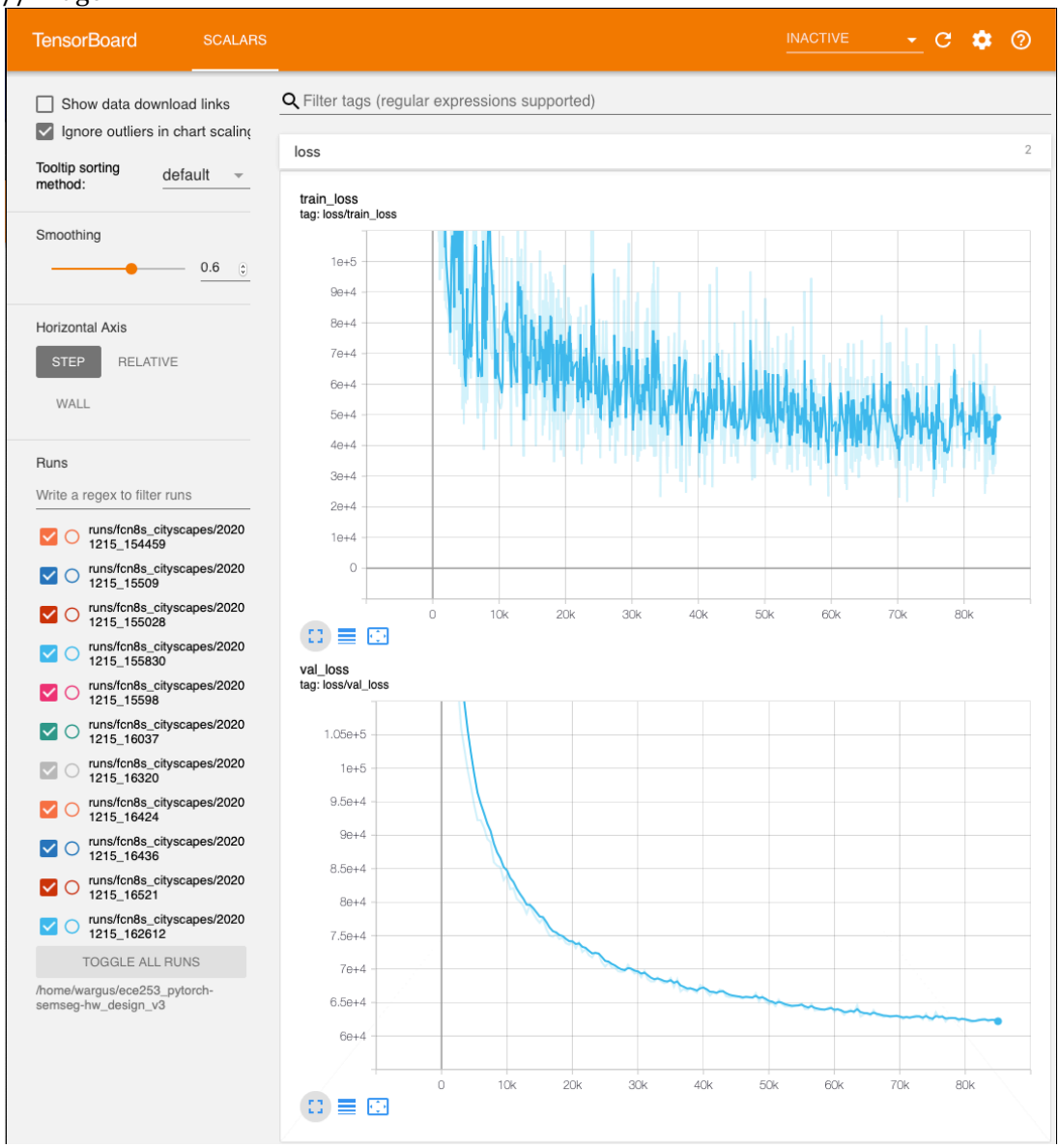
   We use pretained weights from vvg16. The parameters are initialized on each of the convolution blocks in the model file fcn.py. The paper also mentions training the FCN end-to-end but doing so using super-vised pre-training. So they use pre-trained weights, but then we train even more end to end to fine-tune the

model.

3. Please train the network with CityScape dataset. Visualize the training curves (suggested option: use Tensorboard). Include pictures of the training and validation curve. (config file: configs/fcn8s cityscapes.yml)

   Training done.

   //image



   //image

   Both training loss and validation loss drop as the iterations increase, showing the training accomplished something positive on the model.

4. What are the metrics used by the original paper? Do inference (validate.py) on the validation set. Which classes work well? Which classes do not?

   The original paper uses 4 metrics. They were pixel accuracy, mean accuracy, mean IU, and frequency weighted IU. The equations are as follows, with n_{ij} being the number of pixels, that actually belong to class

i, and are predicted to belong to class j, n_{cl} is the number of classes, and $t_i = \sum_j n_{ij}$ is the total number of pixels of class i.
Pixel accuracy: $\sum_i n_{ii} / \sum_i t_i$
Mean accuracy: $(\frac{1}{n_d}) \sum_i n_{ii}/t_i$
Mean IU: $\left(\frac{1}{n_d}\right) \sum_i n_{ii} /(t_i + \sum_i n_{ji} - n_{ii}$
Frequency weighted IU: $(\sum_k t_k)^{-1} \sum_i t_i n_{ii} /(t_i + \sum_j n_{ji} - n_{ii})$

//image

```
Inference time                  (iter   249): 14.22806 fps
Inference time                  (iter   250): 13.85592 fps
Overall Acc:     0.9138714694555968
Mean Acc :       0.6296436822014362
FreqW Acc :      0.8500962541040423
Mean IoU :       0.5402731778298582
0 0.9559419819384405
1 0.6897451181983139
2 0.8475712579967133
3 0.4514492624085583
4 0.36303027535503907
5 0.24706036133828152
6 0.30510968042648645
7 0.3997868148942861
8 0.8592325900045735
9 0.5152351393072165
10 0.8866929928841735
11 0.5455285259732457
12 0.23265178837757952
13 0.8529927676996593
14 0.5286657705569516
15 0.46708003091245665
16 0.3427567367612327
17 0.21999116501345328
18 0.554668118720644
(base) wargus@dsmlp-jupyter-wargus:~/ece253_pytorch-semseg-hw_design_v3$
```

```
self.valid_classes = [
    7,
    8,
    11,
    12,
    13,
    17,
    19,
    20,
    21,
    22,
    23,
    24,
    25,
    26,
    27,
    28,
    31,
    32,
    33,
]
self.class_names = [
    "unlabelled",
    "road",
    "sidewalk",
    "building",
    "wall",
    "fence",
    "pole",
    "traffic_light",
    "traffic_sign",
    "vegetation",
    "terrain",
    "sky",
    "person",
    "rider",
    "car",
    "truck",
    "bus",
    "train",
    "motorcycle",
    "bicycle",
]
```

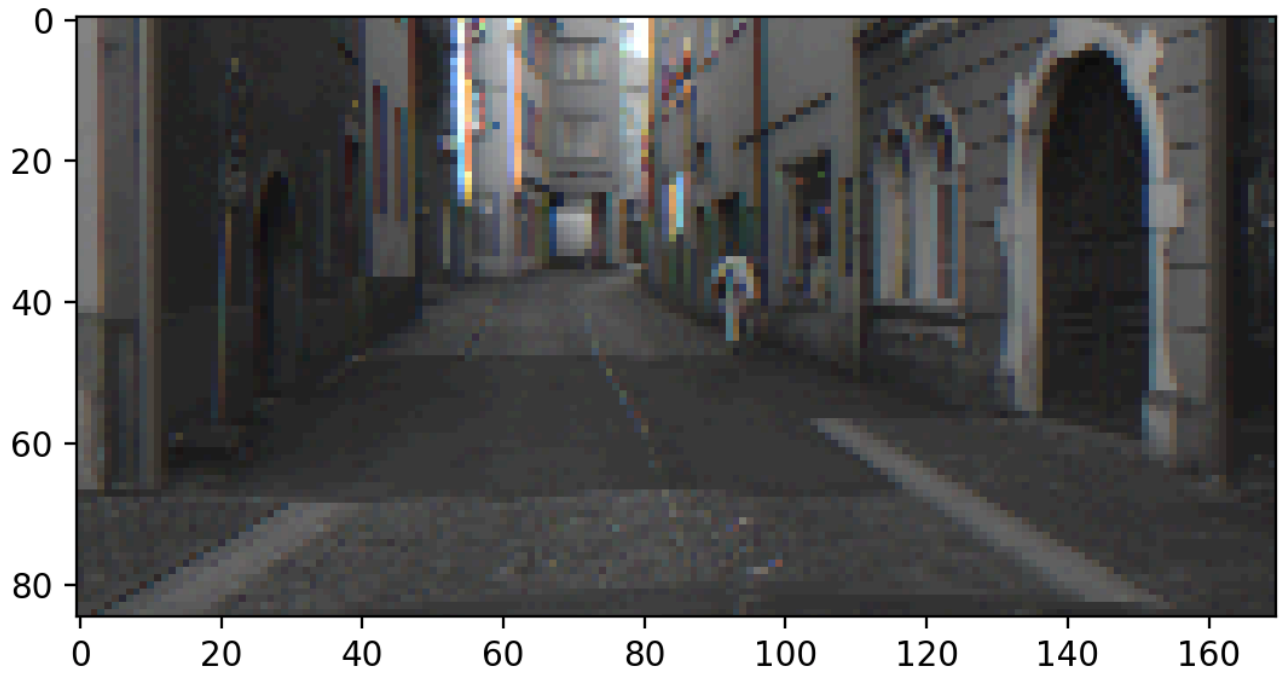
//image
As seen above, the classes 0, unlabeled works the best, and 2: sidewalk, 8: traffic_sign, 10: terrain, and 13: rider all did well and achieved above 85% accuracy. This makes sense as unlabeled is a class for everything not listed so it would hopefully be easy for the network to characterize things that do not meet any of these labels and unlabeled. Sidewalk, and traffic sign have distinctive shapes that are fairly uniform across many places and thus many images so their high accuracy makes sense. Terrain and rider are less distinctive, but terrain usually is bordered by pavement or buildings so that may have aided classification, while rider is simply any human in a vehicle which may have contributed to its high accuracy. These high accuracies could also be the result of the training set being very good for these two classes.
The classes that did not work well are 5: fence, 12: person, and 17: train. This is likely because there is much variation across fences and persons, both in shape, size, and color. Trains also vary but generally

keep the same shape and are on tracks so this is surprising but it could also be the result of a weak training set for the class of trains.

5. Can you visualize your results, by plotting out the labels and predictions of the images? Please include at least two examples (HINT: check the unit test in ptsemseg/loader/cityscapes loader.py) See screenshots.

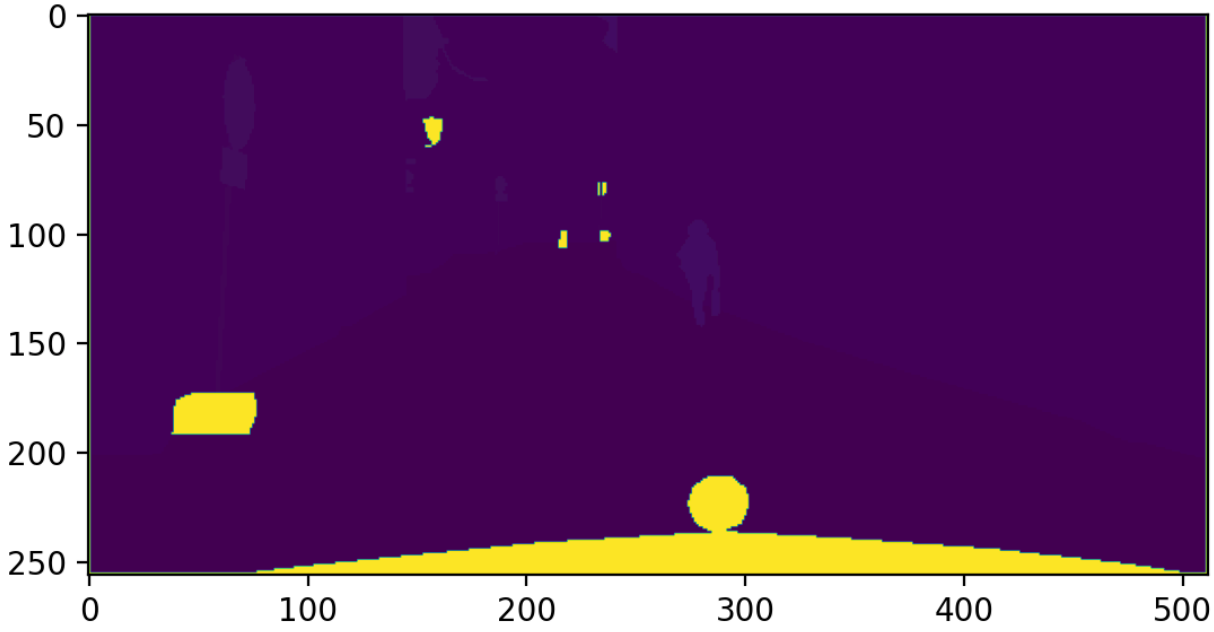
//image

Image 1.



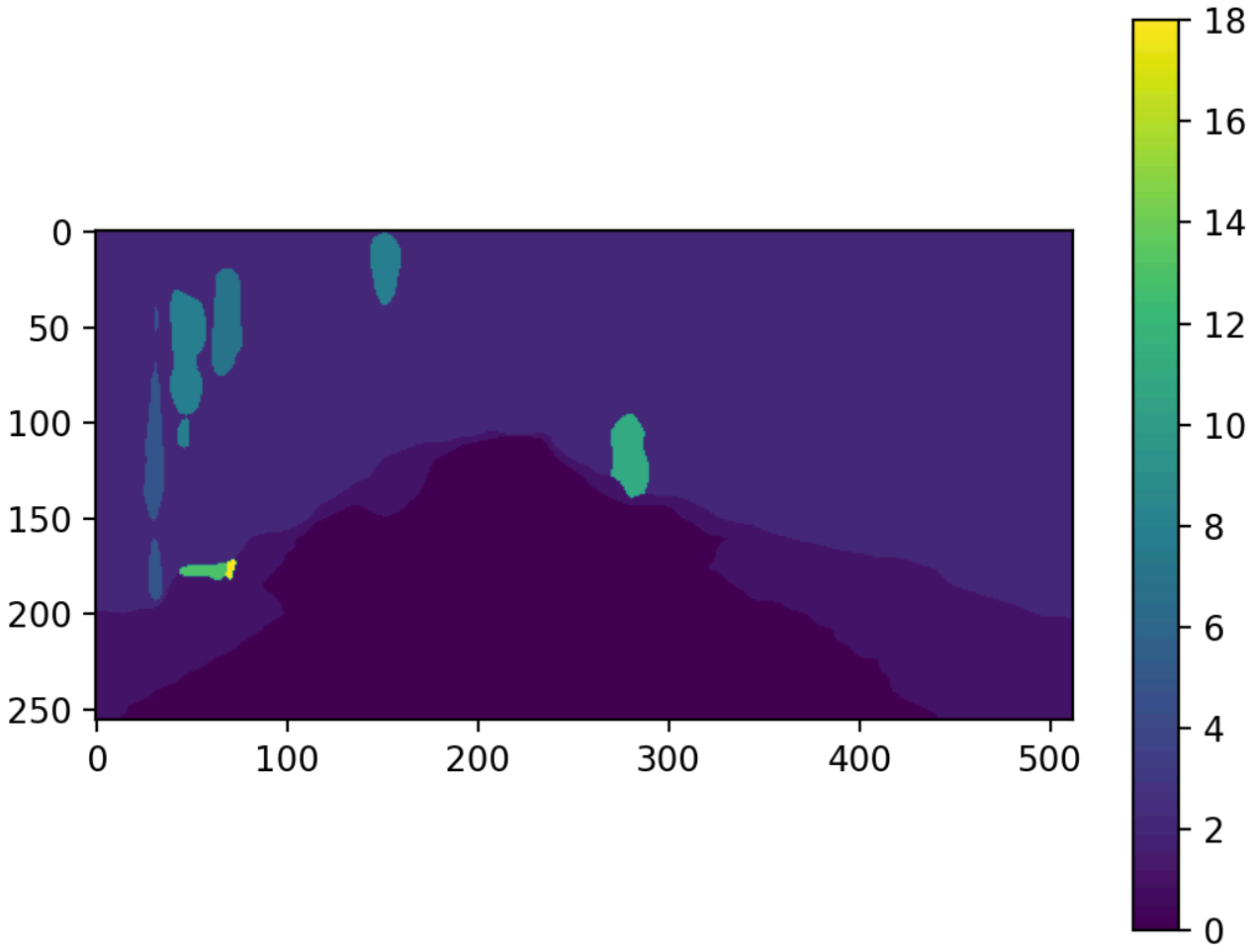
//image

Labels 1



//image

Predictions 1

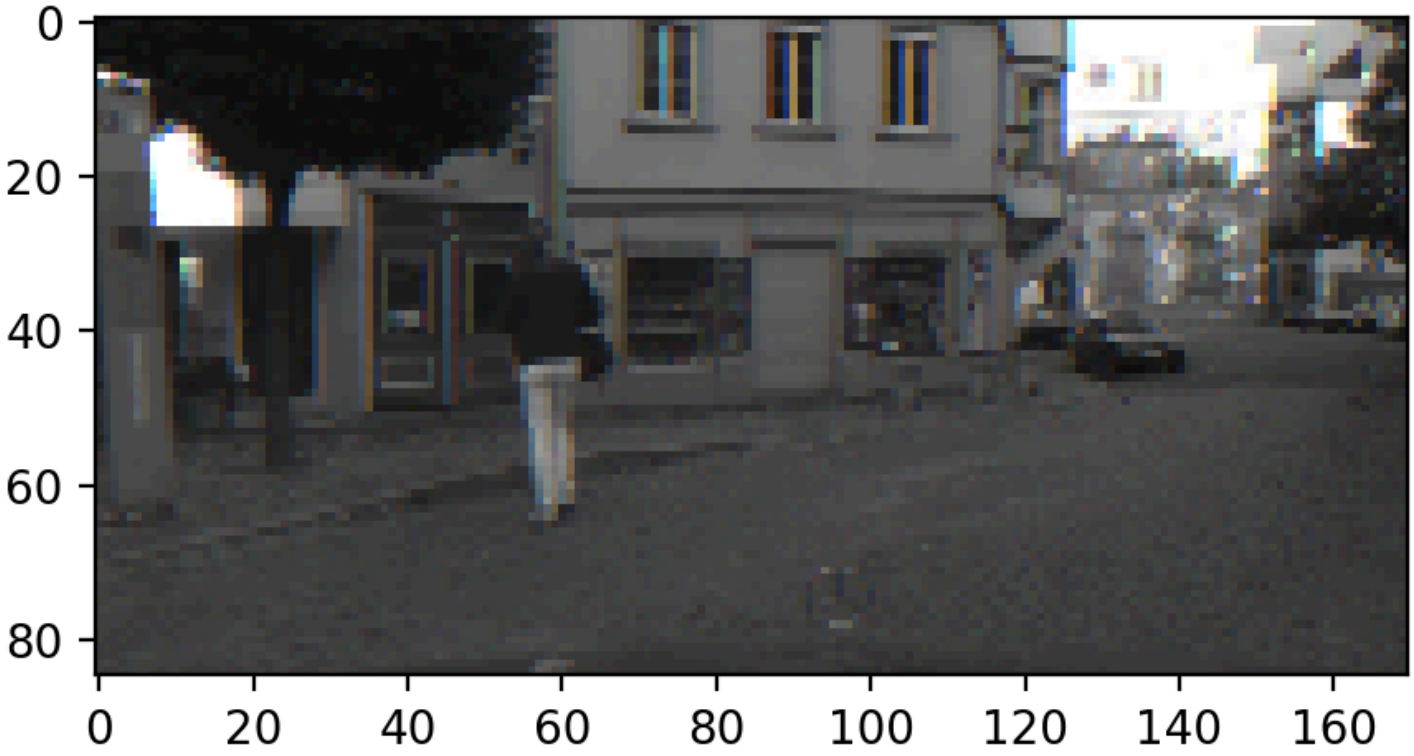


//image

In this first image it is clear that the labels line up with the image, with the person being labeled, and some of the road and the building being labeled. In the prediction it can clearly be seen the road and sidewalk is labeled (as 1 and 2, their labels, respectively) and the person walking is labeled as 12 (its label),. So while not perfect, the segmentation does an okay job on the labeling. It is actually somewhat difficult to assess the what should be labeled in the image just by looking since the image is so low resolution.

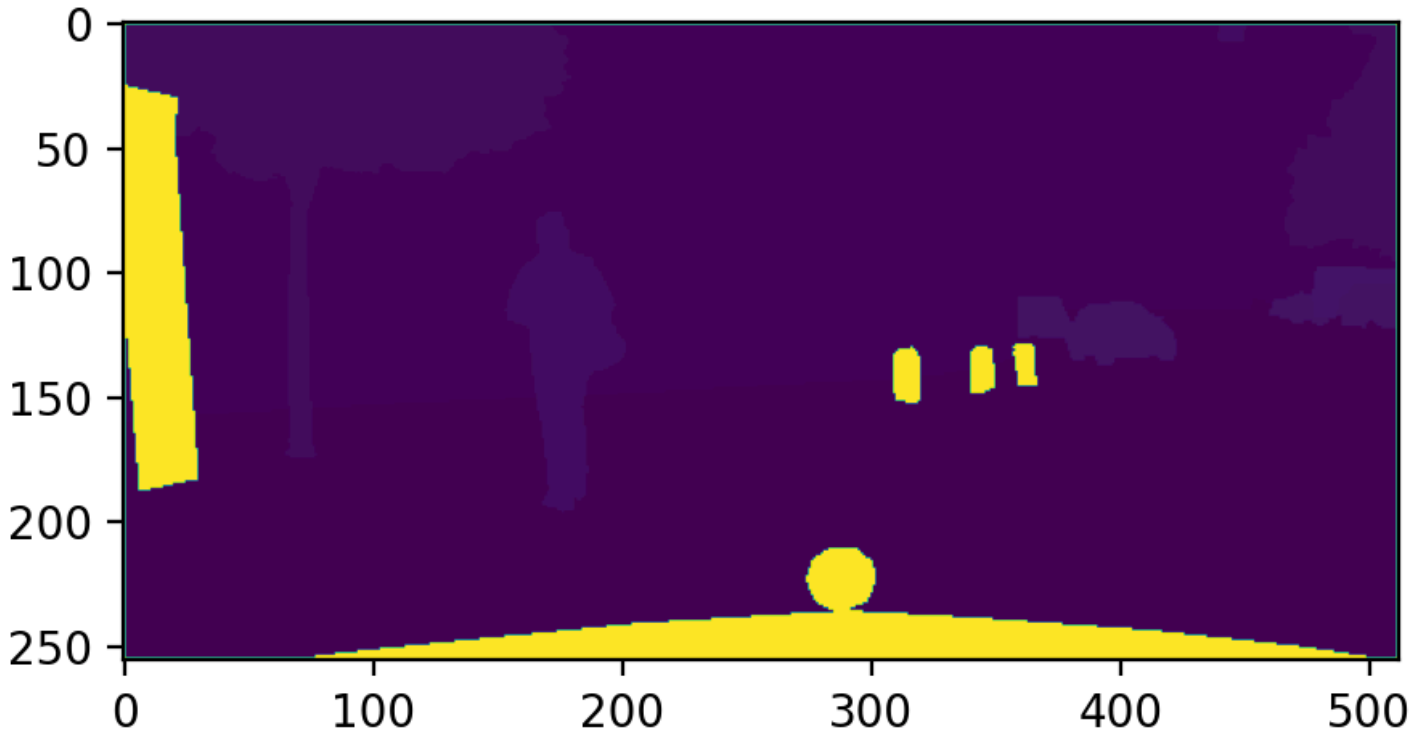
//image

Image 2



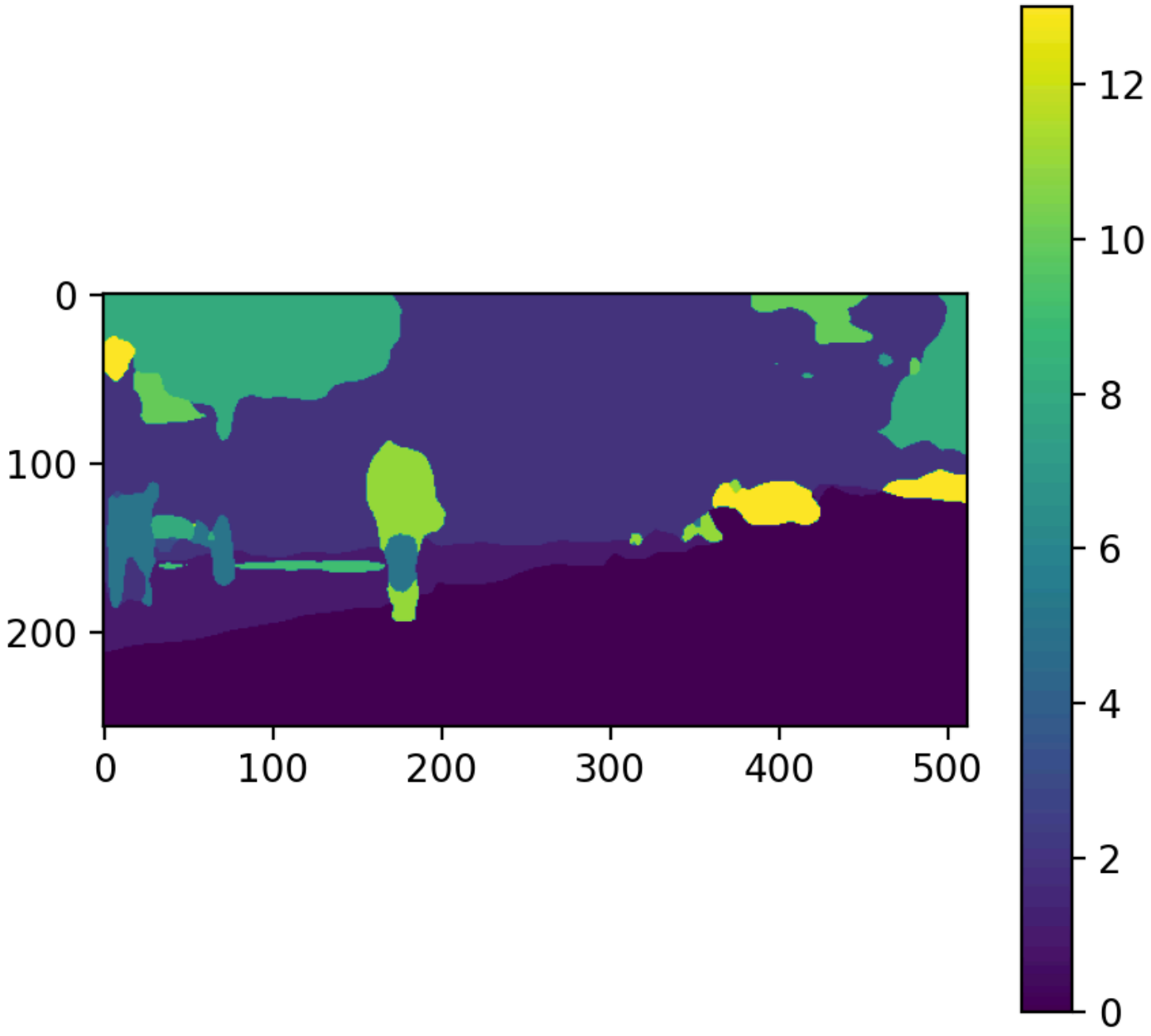
//image

Labels 2



//image

Prediction 2



//image

It is clear that the labels line up with the first image, as the lady is clearly seen in the labels. Also labeled in labels is the vegetation, vehicles, and either a pole or traffic sign of some kind. The road/sidewalk appears to labeled in spots as well. In the prediction, the person is labeled (as 12, its label), (although the legs as labeled as a pole, 6), the Vegetation Is labeled (as 9, its label), street and sidewalk are labeled as 1 and 2, respectively, their labels. The cars appear to labeled 14, their correct label. The pole/traffic sign is labeled as a pole, which is a good labeling. Overall a solid prediction.

6. Please take a photo of a nearby city street, and show the output image from the model. Does the output image look reasonable?

//image

Street

//image

Prediction

//image

The output is very reasonable. The color scaling was problematic for some reason, so no colorbar, but labeling can clearly be identified. The person in from of the camera is partially labeled, The cars are labeled, down the street and in the driveway, although a trashcan was also mistaken for a car on the left side of the street. The trees are labeled as vegetation, the road is labeled well, as is the sky. The driveways appear to labeled as sidewalks, which is good, and the grass lawns are labeled as terrain. There is a few small obscure labels from down the street, which are likely error, but overall a decent prediction.

7. Based on your analysis of the model and training, how can you get better results for prediction? (Give 2 possible options. Change the parameters? Change the network architecture? Or other thoughts? You can checkout the FCN paper [1] and the followup works.)

The first thing I would use to get better prediction is to train on higher resolution images. This will significantly increase training time, but it would greatly improve model accuracy, especially if it is classifying high res images. The higher training time would be acceptable if the model was going to implemented in an enterprise grade solution. The second improvement that could be made would be to reduce the class labels, especially on similar classes like car, truck, and bus, or traffic light and traffic sign, or even wall and fence. A third thing that could be done is to train on a more diverse group of images. From what I could see, the dataset appeared to all be based on pictures from streets in Europe (Bremen, Hamberg, Stuttgart, Zurich), all cities in Germany and Switzerland I beleive, which could lead to poor classification in other parts of the world.

To be noted:

- Upload the zip file to the server. Follow the steps in README.md to install environments

  and requirements

- Training time is around 5 hours.
- When you're running the server, save the URL so that you can access the tab later once you close it.
- Please read the FCN paper [1]. References

[1] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmenta- tion," in 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Jun. 2015, pp. 3431–3440, iSSN: 1063-6919.

[2] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The Cityscapes Dataset for Semantic Urban Scene Understanding," arXiv:1604.01685 [cs], Apr. 2016, arXiv: 1604.01685. [Online]. Available: http: //arxiv.org/abs/1604.01685

Code Appendix

Problem 1

```
%hough transform (HT) using the (rho, theta) parameterization
%use accumulator cells w/ res of 1 deg in theta and 1 pixel in rho
%% Part 1 and 2

thetaRes = 1;
rhoRes = 1;
dim = 11;
testIm = zeros(dim, dim);
xs = [1, 11, 6, 1, 11];
ys = [1, 1, 6, 11, 11];
testIM(1, 1) = 1;
testIM(1, 11) = 1;
testIM(6, 6) = 1;
testIM(11, 1) = 1;
testIM(11, 11) = 1;
image = testIM;
figure()
imshow(image)
hold on
title('Original Image')
hold off

[HAccum, rows, thetaRad] = HT(image, thetaRes, rhoRes);



threshold = 2;

Ploot(image, threshold, rows, thetaRad, HAccum);

%% Part 3

image = imread('HW4_2020/data/lane.png');
image = imresize(image, .1);
figure()
imshow(image)
title('Original Image')

BW = rgb2gray(image);
imageTemp = edge(BW,'sobel');
image = imageTemp;
figure()
imshow(image)
title('Binary Edge Image')

[HAccum, rows, thetaRad] = HT(image, thetaRes, rhoRes);
threshold = 0.75*max(HAccum(:));
Ploot(image, threshold, rows, thetaRad, HAccum);


%% Part 4
image = imread('HW4_2020/data/lane.png');
image = imresize(image, .1);


BW = rgb2gray(image);
imageTemp = edge(BW,'sobel');
image = imageTemp;
```

```
[HAccum, rows, thetaRad] = HT(image, thetaRes, rhoRes);

threshold = 0.75*max(HAccum(:));
Ploot(image, threshold, rows, thetaRad, HAccum);


function [HAccum, rows, thetaRad] = HT(image, thetaRes, rhoRes)
sz = size(image);
dist = floor(sqrt( sz(1)*sz(1) + sz(2)*sz(2) ));
theta = (-90:thetaRes:90);
thetaRad = theta.*(pi/180);
rows = (-dist:rhoRes:dist);
HAccum = zeros(length(rows), length(thetaRad));
cosTheta = cos(thetaRad);
sinTheta = sin(thetaRad);
[yInd, xInd] = find(image~=0); %y is pts(1), x pts(2)
numPts = length(xInd);
figure()
for i = 1:numPts
 i;
 %yInd = pts(1);
 %xInd = pts(2);
 add = dist+1;
 for j = 1:length(thetaRad)
    y = yInd(i);
    x = xInd(i);
    r = x*cosTheta(j) + y*sinTheta(j);
    rowA = floor(r) + add;
    rTemp(j) = r;
    HAccum(rowA,j) = 1+ HAccum(rowA,j);
 end
 plot(theta, -rTemp)
 hold on
 colorbar
 xlabel('theta')
 ylabel('rho')
 grid on
 nothing = 0;
end
rows = (-dist:rhoRes:dist);
hold off
end


function Ploot(image, threshold, rows, thetaRad, HAccum)
numPts = length(rows);
sz = size(HAccum);

linePts = (1:numPts);
lineAxs=zeros(size(linePts));
countLines =0;
for i=1:sz(1)
 for j=1:sz(2)
  if HAccum(i,j)>=threshold+1
    countLines =countLines+1;
    lineAxs(countLines,:)=(rows(i)-linePts*cos(thetaRad(j)))/sin(thetaRad(j));
    end
 end
```

```
end

figure()
imshow(image)
hold on
for l = 1:countLines
   plot(linePts, lineAxs(l,:), 'b')
   hold on
end
title('Image with Hough Lines')
hold off
end
```

Problem 2

```
% N is number of pixels in image
% M = 3 for RGB
% k is number of clusters


%setup first function
im = imread('HW4_2020/data/white-tower.png');
sz = size(im);
N = sz(1)*sz(2);
M = 3;

%call first function
features = createDataset(im);

%setup second function
nclusters = 7;
rng(5);
id = randi(size(features, 1), 1, nclusters);
centers = features(id, :);

%call second function
[idx, centers] = kMeansClustering(features, centers);

%call third function
im_seg = mapValues(im, idx, centers);

%display image
figure (1)
imshow(im)
title('image before segmentation', 'FontSize', 24);
%display image seg
figure (2)
imshow(im_seg)
title('image after segmentation', 'FontSize', 24);

centers

%features = reshape(im, [N,M]);
function features = createDataset(im)
    sz = size(im);
    N = sz(1)*sz(2);
    M = 3;
    features = reshape(im, [N,M]);
```

```
end

function [idx, centers] = kMeansClustering(features, centers)
    %idx = zeros(N,1);
    %comannds: pdist2, find, check new center: sum of features in
    %class/points in that class
    newCenters = cast(centers, 'double');
     sz = size(centers);
     nclusters = sz(1);
    % FUNCTION 2
    %centers and features already defined
    %idx = zeros(N,1);
    for i = 1:100
        dist = pdist2(features, centers);
        [minDist, minIndex] = min(dist, [], 2);

        for k = 1:nclusters
           allPts = features( minIndex== k, : );
           tempCenter = mean(allPts,1);
           newCenters(k,:) = tempCenter;

        end
        if isequal(centers, newCenters)
            break
        end
        centers = newCenters;
    end
    idx = minIndex;
    nothing  =0;
end

function im_seg = mapValues(im, idx, centers)
    features = createDataset(im);
    sz = size(features);
    for l = 1:sz(1)
        k = idx(l);
        features(l,:) = centers(k, :);
    end
    %features(idx == k, :) = centers(k, :);
    im_seg = reshape(features, size(im) );
    nothing =0;
end
```

Problem 3:

See zip file, there was too many small changes to different scripts to make putting them in here viable.