Bolin He, PID: A53316428, Hw04

Dec 3,2019

Q1 Hough Transform

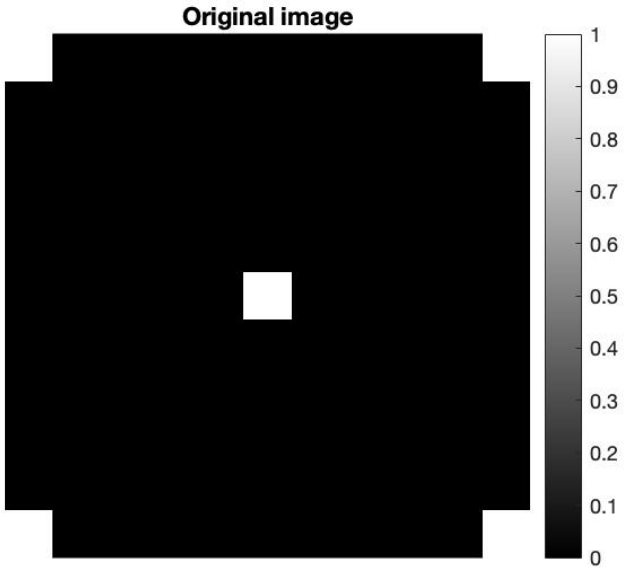
```
clear all;
clc;
% i
% function [rho,H] = HoughTransform(img)
%     % H: HT transform matrix
%
%     % Initialization
%     [x,y] = size(img);
%     theta = [-90:90];
%     len = ceil(sqrt((x-1)^2+(y-1)^2));
%     [M,N] = find(img);
%     rho = zeros(length(x),2*90+1);
%     H = zeros(2*len+1,2*90+1);
%
%     % Calcualte rho
%     for i = 1:length(M)
%         rho(i,:) =
 round((M(i)-1).*cosd(theta)+(N(i)-1).*sind(theta));
%     end
%
%     % Calculate H
%     for i = 1:length(M)
%         for j = 1:181
%             x_idx = rho(i,j)+len;
%             H(x_idx,j) = H(x_idx,j)+1;
%         end
%     end
%
%     Hs = H./max(H(:));
%     imshow(Hs,[],'XData',theta,'YData',-
len:len,'InitialMagnification','fit');
%     xlabel('\theta'), ylabel('\rho');
%     colorbar;colormap(gray);
%     axis on, axis normal
%
% end


% ii
test_img = [1 0 0 0 0 0 0 0 0 0 1
            0 0 0 0 0 0 0 0 0 0 0
            0 0 0 0 0 0 0 0 0 0 0
            0 0 0 0 0 0 0 0 0 0 0
            0 0 0 0 0 0 0 0 0 0 0
            0 0 0 0 0 1 0 0 0 0 0
            0 0 0 0 0 0 0 0 0 0 0
```

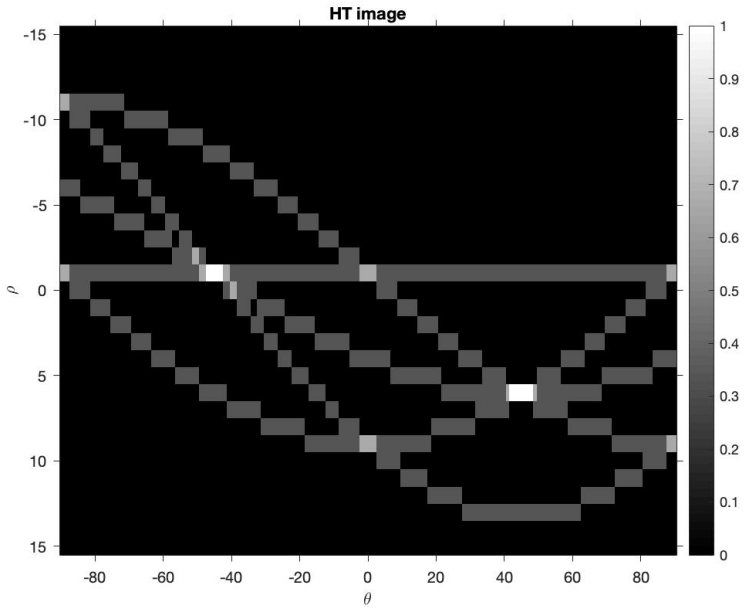
```
            0 0 0 0 0 0 0 0 0 0 0
            0 0 0 0 0 0 0 0 0 0 0
            0 0 0 0 0 0 0 0 0 0 0
            1 0 0 0 0 0 0 0 0 0 1];

imshow(test_img,'InitialMagnification',5000);
title('Original image');
colorbar;
figure;
HoughTransform(test_img);
title('HT image');
colorbar;

% select point (45,0) and (-45,7)
figure;
hold on;
xx = 0:10;
y1 = round(0/sind(45)-cosd(45)/sind(45).*xx)+10;
y2 = round(7/sind(-45)-cosd(-45)/sind(-45).*xx)+10;
for i = 1:length(xx)
        test_img(xx(i)+1,y1(i)+1) = 1;
        test_img(xx(i)+1,y2(i)+1) = 1;
end
imshow(test_img,'InitialMagnification',5000)
colorbar;
title('Original image with lines')

% iii
lane = imread('lane.png');
lane2 = double(rgb2gray(lane));
E = edge(lane2,'sobel');
title('Image after Sobel Filter');
figure
imshow(lane)
title("Original image");
colorbar;
figure
imshow(E)
title("Binary edge image");
colorbar;
figure
[rho2, H2] = HoughTransform(E);
title("HT")
colorbar;

% Threshold
[A,B] = find(H2 > 0.75*max(H2(:)))
[row,col] = size(E);
len = ceil(sqrt((row-1)^2+(col-1)^2));
Amap = A-len;
Bmap = B-91;
L = zeros(length(A),2);
figure
imshow(lane);
```

```
hold on;

for i=1:length(A)

 L(i,:)=[round((Amap(i))*sind(Bmap(i))+1),round((Amap(i))*cosd(Bmap(i))+1)];
    j=1:col;
    k=-(1/cotd(Bmap(i)))*(j-L(i,1))+L(i,2);
plot(j,k,'r','linewidth',1)
set(gca,'ydir','reverse');
ylim([1 row]);xlim([1 col])
end

title('Original image with lines');
colorbar;

% iv
figure()
imshow(lane);
colorbar;
hold on;
% Select theta as [-30,40]U[30,40]
drive = find(abs(Bmap)>=30 & abs(Bmap)<=40);
L2 = zeros(length(A),2);
for i=1:length(drive)

 L2(i,:)=[round((Amap(drive(i)))*sind(Bmap(drive(i)))+1),round((Amap(drive(i)))*co
    j=1:col;
    k=-(1/cotd(Bmap(drive(i))))*(j-L2(i,1))+L2(i,2);
   plot(j,k,'g','linewidth',2)
   set(gca,'ydir','reverse');
   ylim([1 row]);xlim([1 col])
end
title('Driver lane only')
close all;
fprintf('We choose theta as [-30,40]U[30,40]');


A =

         363
        1005
        1036
        1147
        1172
        1304
        1307
        1332
        1337
        1357
        1479
        1479
        1482
        1482
        1614
```

```
        1618
        1915


B =

     1
    53
    56
    68
    70
    84
    85
    87
    87
    91
   105
   106
   106
   107
   127
   128
   181

We choose theta as [-30,40]U[30,40]
```

Published with MATLAB® R2018a


Original image
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0


HT image
-15
-10
-5
0
5
10
15
ρ
-80
-60
-40
-20
0
20
40
60
80
θ
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0

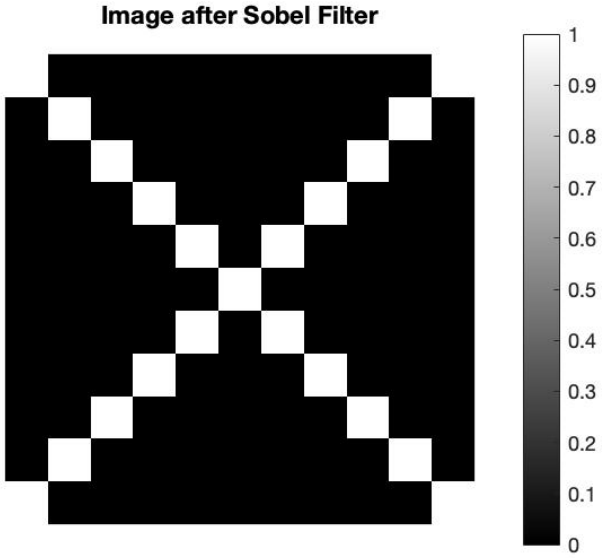
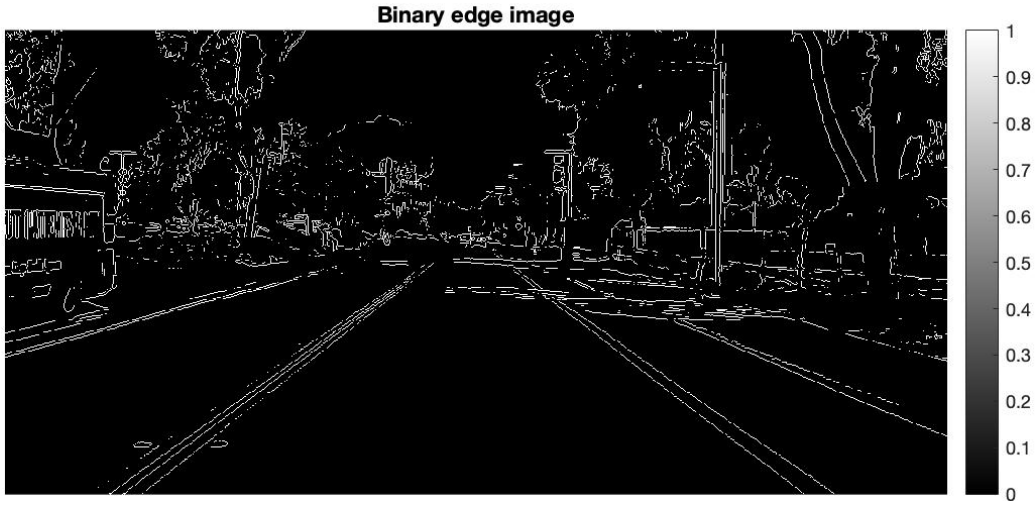
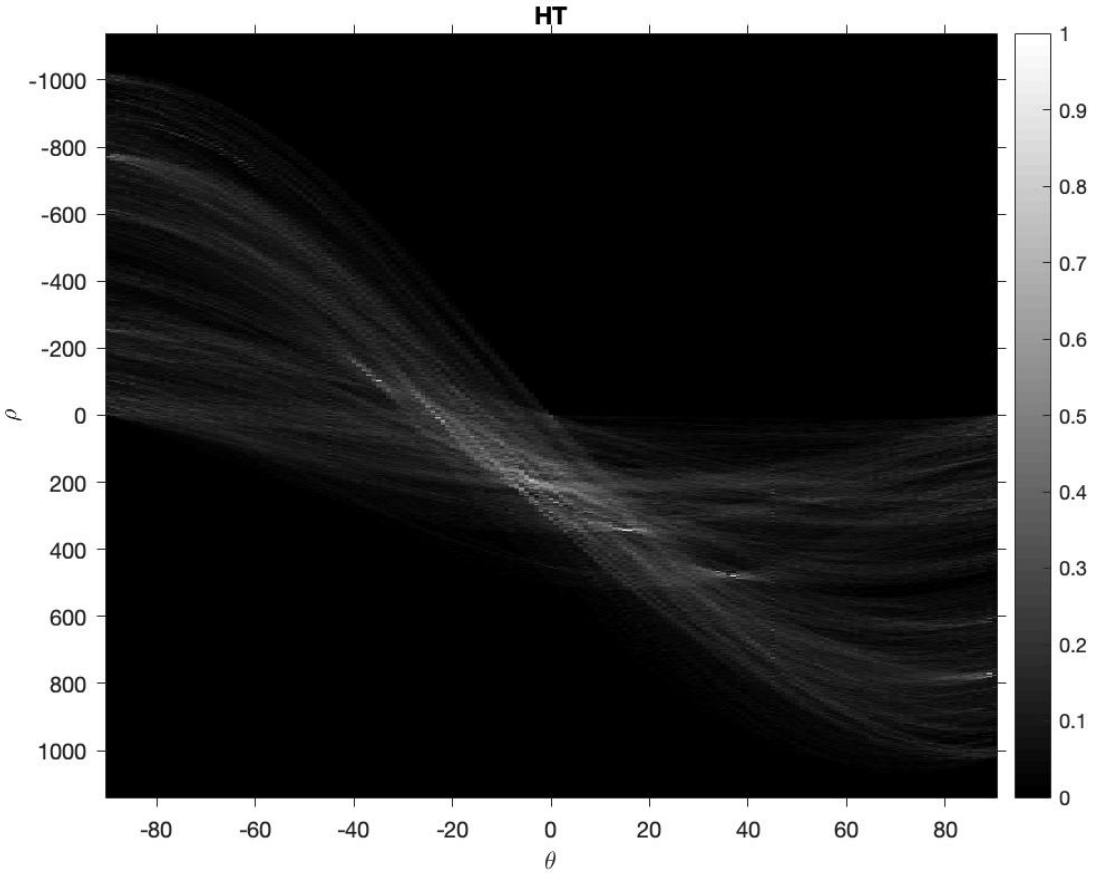


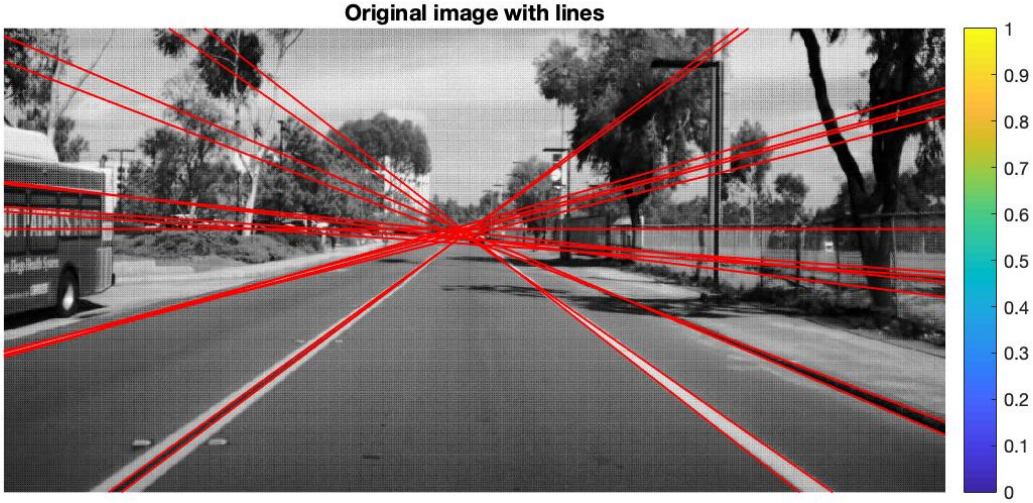
Image after Sobel Filter
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0


Original image
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1

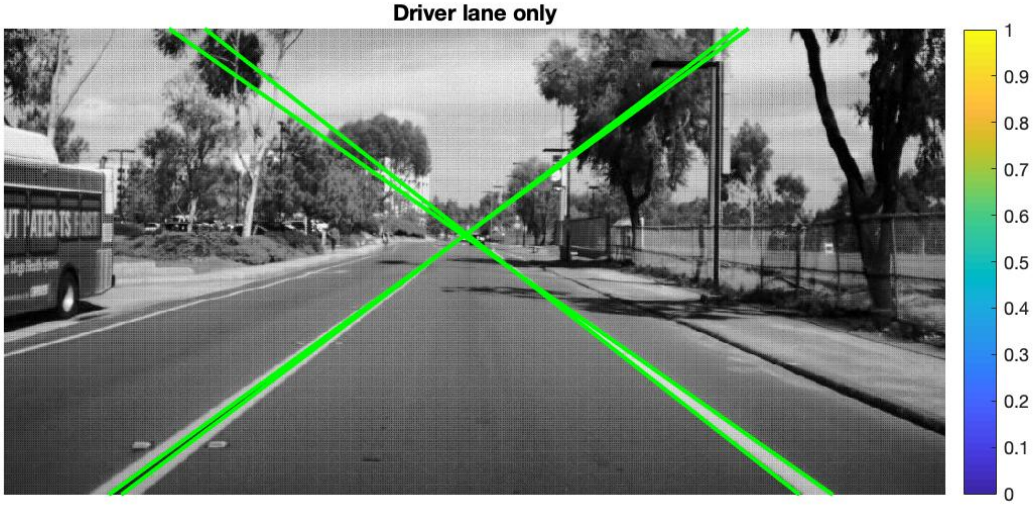

Binary edge image
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0


HT
-1000
-800
-600
-400
-200
0
200
400
600
800
1000
ρ
-80
-60
-40
-20
0
20
40
60
80
θ
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0

Original image with lines



Driver lane only



```
% Q2 K-Means Segmentation
close all;
clear all;
clc;

im = imread('white-tower.png');
im = double(im);

features = createDataset(im);

nclusters = 7;
id = randi(size(features,1),1,nclusters);
centers = features(id,:);

[idx, centers] = KMeansCluster(features,centers);

im_seg = mapValues(im, idx);

% Print centers
fprintf('The centers of seven clusters are:\n')
table(centers)


function features = createDataset(im)
    N1 = reshape(im(:,:,1),[],1);
    N2 = reshape(im(:,:,2),[],1);
    N3 = reshape(im(:,:,3),[],1);
    features = [N1,N2,N3];
end


function [idx, centers] = kMeansCluster(features, centers)

    [M,N] = size(features);
    [center_row,center_col] = size(centers);

% Iterate for 100 times
    % Find the nearest center
    for i = 1:100;
        Distance = pdist2(features,centers);
        [~,idx] = min(Distance,[],2);
        centers = zeros(center_row,3);
        % Recalculate the center
        for j = 1:center_row
            center_avg = find(idx == j);
            centers(j,:) = mean(features(center_avg,:));
        end
    end
end


function [im_seg] = mapValues(im,idx)
```

```
    N1 = reshape(im(:,:,1),[],1);
    N2 = reshape(im(:,:,2),[],1);
    N3 = reshape(im(:,:,3),[],1);
    features = [N1,N2,N3];
    [a,b,c] = size(im);
    im_seg=zeros(a*b,c);

   for i = 1:7
       idx2 = find(idx == i);
       Mean = mean(features(idx2,:));
       for j = 1:length(idx2)
        im_seg(idx2(j),:) = Mean;
       end
   end

%    Plot
   im_seg = reshape(im_seg,[720,1280,3]);
   imshow(uint8(im));
   title('Original Image');
   figure;
   imshow(uint8(im_seg));
   title('Image after segmentation');
end
```

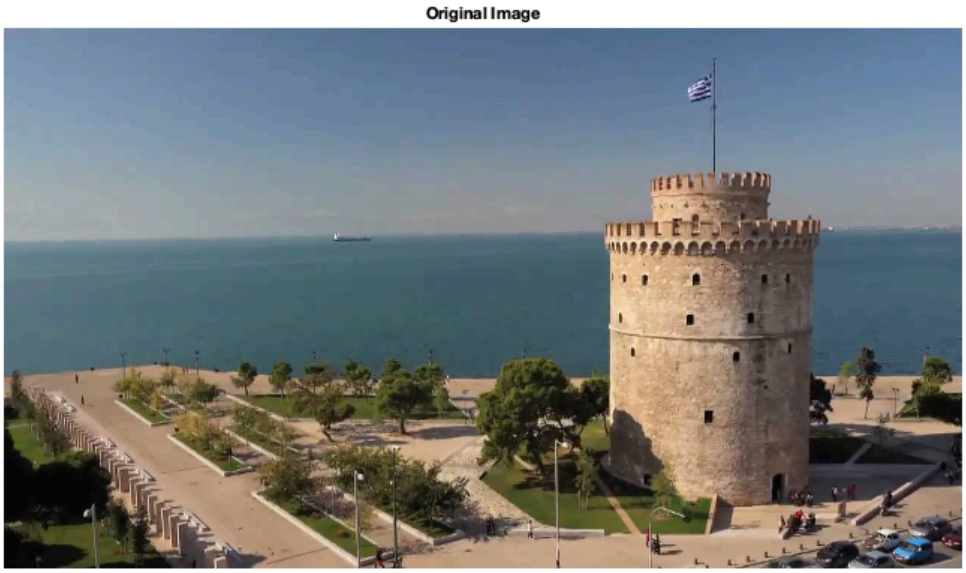
The centers of seven clusters are:

ans =

7×1 table

centers
86.715
73.06
33.395
101.14
159.97
138.7
204.43

77.177	57.434
99.244	109.61
30.118	22.249
126.2	154.49
134.15	110.56
152.95	165.88
172.78	142.96



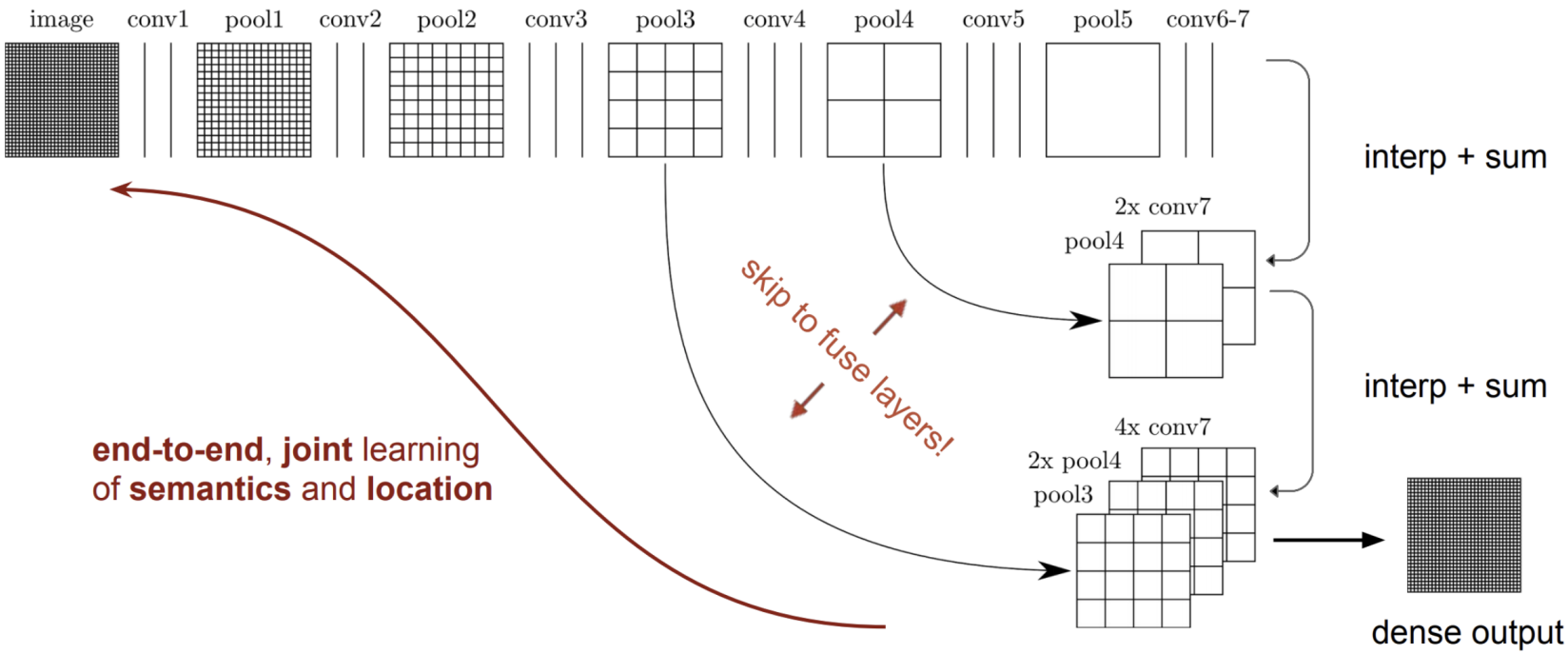
Published with MATLAB® R2018a

Q1

From the figures below, we can clearly see a multi-stream network that fuses features/predictions across layers. Basically, there are variants of the FCN architecture, which mainly differ in the spatial precision of their output. For example, the figures below show the FCN-32, FCN-16 and FCN-8 variants. In the figures, convolutional layers are represented as vertical lines between pooling layers, which explicitely show the relative size of the feature maps.

As shown below, these 3 different architectures differ in the stride of the last convolution, and the skip connections used to obtain the output segmentation maps. We will use the term downsampling path to refer to the network up to conv7 layer and we will use the term upsampling path to refer to the network composed of all layers after conv7. It is worth noting that the 3 FCN architectures share the same downsampling path, but differ in their respective upsampling paths.

For the model FCN8s, it sums the 2x upsampled conv7 (with a stride 2 transposed convolution) with pool4, upsamples them with a stride 2 transposed convolution and sums them with pool3, and applies a transposed convolution layer with stride 8 on the resulting feature maps to obtain the segmentation map.



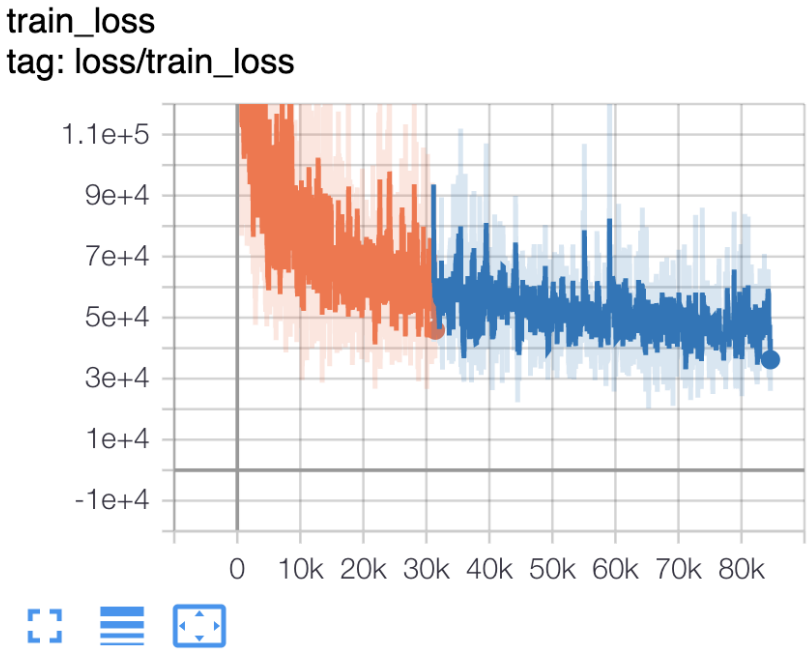
Reference: Jonathan Long, Fully Convolutional Networks for Semantic Segmentation.

Q2

Yes, we use the weights from Vgg16, so we did not train the model from scratch.

Q3

The result from Tensorboard is:



Q4

Matrics used by original paper:
·pixel accuracy
·mean accuraccy
·mean IU
·frequency weighted IU

Among all the classes performance:
·Best: class 0
·Worst: class 12

Q5

In [2]:
```
%matplotlib notebook

import numpy as np
import torch
import torch.utils.data as td
import torchvision as tv
import matplotlib.pyplot as plt
import torch
import sys
sys.path.insert(0, "hw4data/ece253_pytorch_semseg_hw_design_v3/")

from ptsemseg.loader import cityscapes_loader as city
from ptsemseg.models import fcn
from ptsemseg.models import get_model
from ptsemseg.loader import get_loader
from ptsemseg.metrics import runningScore
from ptsemseg.utils import convert_state_dict
from ptsemseg.utils import recursive_glob
from PIL import Image
import yaml

torch.backends.cudnn.benchmark = True
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(device)
```

```
cuda
```

In [3]:
```
# Setup Model
with open(r'hw4data/ece253_pytorch_semseg_hw_design_v3/runs/fcn8s_cityscapes/20
    cfg = yaml.load(file, Loader=yaml.FullLoader)

state_path = 'hw4data/ece253_pytorch_semseg_hw_design_v3/runs/fcn8s_cityscapes/
```

In [4]:
```
data_loader = get_loader(cfg["data"]["dataset"])
data_path = cfg["data"]["path"]

t_loader = data_loader(
    data_path,
    is_transform=True,
    split=cfg["data"]["train_split"],
    img_size=(cfg["data"]["img_rows"], cfg["data"]["img_cols"])
)

v_loader = data_loader(
    data_path,
    is_transform=True,
    split=cfg["data"]["val_split"],
    img_size=(cfg["data"]["img_rows"], cfg["data"]["img_cols"]),
)

ts_loader = data_loader(
    data_path,
    is_transform=True,
    split="test", test_mode=True,
    img_size=(cfg["data"]["img_rows"], cfg["data"]["img_cols"])
)
```

```
Found 2975 train images
Found 500 val images
Found 1525 test images
```

In [5]:
```
trainloader = td.DataLoader(
        t_loader,
        batch_size=cfg["training"]["batch_size"],
        num_workers=cfg["training"]["n_workers"],
        shuffle=True
    )

valloader = td.DataLoader(
        v_loader, batch_size=cfg["training"]["batch_size"], num_workers=cfg["t
    )

testloader = td.DataLoader(
        ts_loader, batch_size=cfg["training"]["batch_size"], num_workers=cfg["
    )
```

In [6]:
```
n_classes = t_loader.n_classes
bs = cfg["training"]["batch_size"]

model = get_model(cfg['model'], n_classes)
state = convert_state_dict(torch.load(state_path)["model_state"])
model.load_state_dict(state)
model.eval()
model.to(device);
```

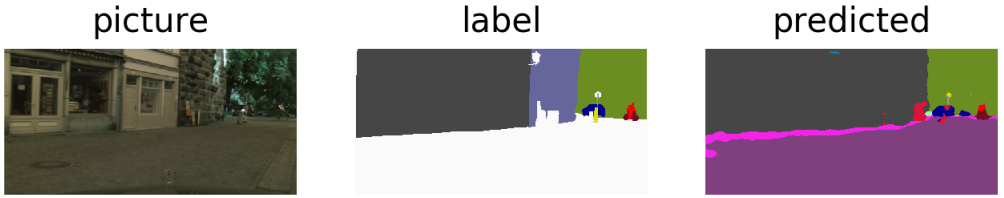
Run this block multiple times to test different image

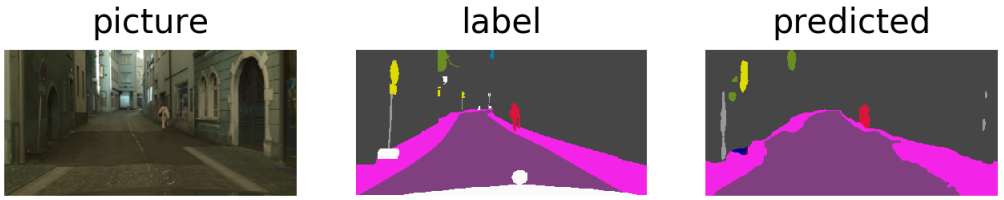
In [7]:
```
# 0: visualize train set;
# 1: visualize validation set;
# 2: visualize test set
vis_mode = 1

if vis_mode == 0:
    loader, td_loader = t_loader, trainloader
elif vis_mode == 1:
    loader, td_loader = v_loader, valloader
elif vis_mode == 2:
    loader, td_loader = ts_loader, testloader

dataiter = iter(td_loader)
```

```
WARN: resizing labels yielded fewer classes
```

In [8]:
```
def getImg(img_path):
    test_img = Image.open(img_path).convert('RGB')
    transform = tv.transforms.Compose([
        tv.transforms.Resize((cfg["data"]["img_rows"], cfg["data"]["img_cols"]
        tv.transforms.ToTensor(),
        tv.transforms.Normalize((0.5,0.5,0.5), (0.5,0.5,0.5))
        ])
    return transform(test_img).unsqueeze(0)

test_img1 = getImg("hw4data/test1.jpg")
test_img2 = getImg("hw4data/test2.jpg")
test_img3 = getImg("hw4data/test3.jpg")
```

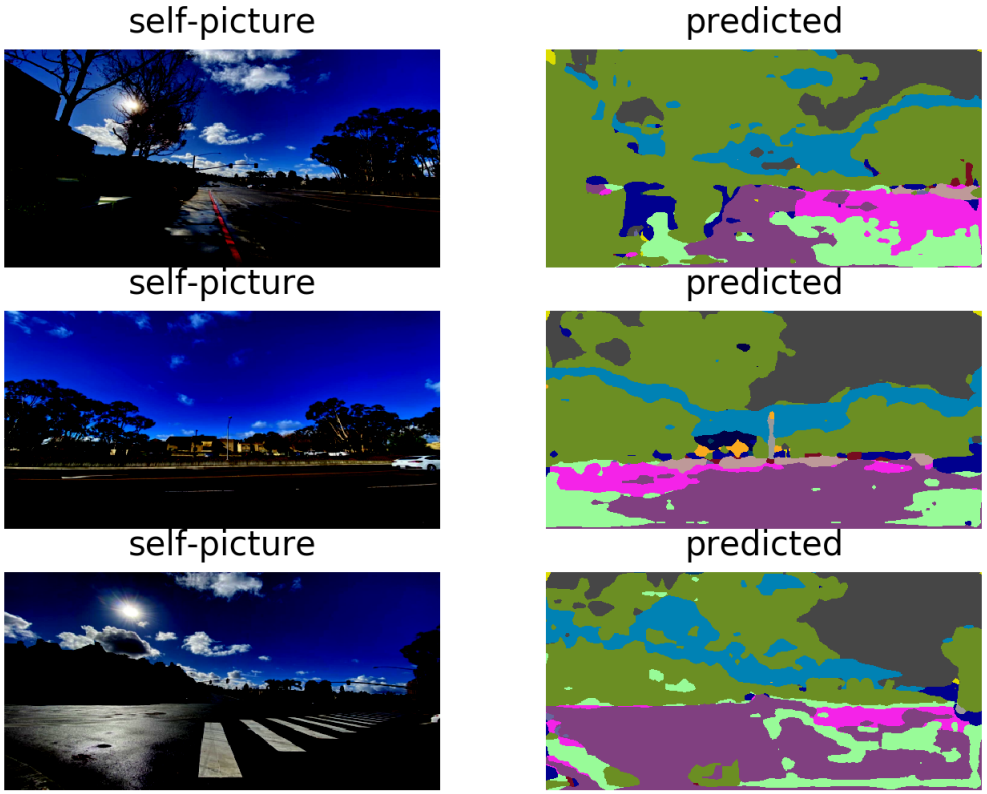
In [9]:
```
# uncomment to use provided datasets
images, labels = dataiter.next()

# uncomment the follwoing two lines to use self-taken pictures
# images = torch.cat([test_img1, test_img2, test_img3], 0)
# labels = torch.ones([images.shape[0], images.shape[2], images.shape[3]]) * 2

bs = images.shape[0]
imgs = images.to(device, dtype=torch.float)
pred = model(imgs).data.max(1)[1].cpu().numpy()

# show images
f, axarr = plt.subplots(bs, 3)

for j in range(bs):
    toshow = images.numpy()[j]
    toshow = np.moveaxis(toshow, [0, 1, 2], [2, 0, 1])
    axarr[j][0].imshow(toshow)
    axarr[j][1].imshow(loader.decode_segmap(labels.numpy()[j]))
    axarr[j][2].imshow(loader.decode_segmap(pred[j]))

    axarr[j][0].axis('off')
    axarr[j][1].axis('off')
    axarr[j][2].axis('off')
    axarr[j][0].set_title('picture')
    axarr[j][1].set_title('label')
    axarr[j][2].set_title('predicted')
plt.show()
```





Q6

In [10]:
```
images = torch.cat([test_img1, test_img2, test_img3], 0)
labels = torch.ones([images.shape[0], images.shape[2], images.shape[3]]) * 250

bs = images.shape[0]
imgs = images.to(device, dtype=torch.float)
pred = model(imgs).data.max(1)[1].cpu().numpy()

# show images
f, axarr = plt.subplots(bs, 2)

for j in range(bs):
    toshow = images.numpy()[j]
    toshow = np.moveaxis(toshow, [0, 1, 2], [2, 0, 1])
    axarr[j][0].imshow(toshow)
    axarr[j][1].imshow(loader.decode_segmap(pred[j]))

    axarr[j][0].axis('off')
    axarr[j][1].axis('off')
    axarr[j][0].set_title('self-picture')
    axarr[j][1].set_title('predicted')
plt.show()
```



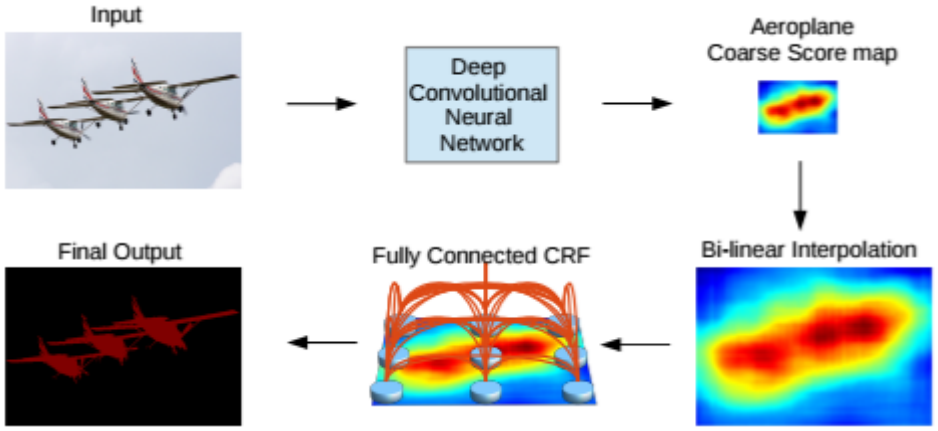

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
```

The result looks reasonable.

Q7

Improvement 1:

The upsampling process would lose some important informaiton. We can process Fully Connected Conditional Random Fields after pixels classification, which can improve the details in figures by post processing.



Reference: Liang-Chieh Chen, Semantic image segmentation with deep convolutional nets and fully connected crfs

Improvement 2:

We can try to achieve our goal by using RNN with CRF, which perform a better result during experiment.


Input Image
FCN-8s
DeepLab
CRF-RNN
Ground Truth
B-ground
Aero plane
Bicycle
Bird
Boat
Bottle
Bus
Car
Cat
Chair
Cow
Dining-Table
Dog
Horse
Motorbike
Person
Potted-Plant
Sheep
Sofa
Train
TV/Monitor

Reference:Shuai Zheng, Conditional random fields as recurrent neural networks

Improvement 3:

We can try to use dilated convolution instead of traditional convolution in the paper. In this case, we can increase the receptive fields and retain more information for upsampling.

Reference: Fisher Yu, MULTI-SCALE CONTEXT AGGREGATION BY DILATED CONVOLUTIONS