1 Optimization

Constrained Least Square Problem

$$\min_x \frac{1}{2}\|Ax-b\|_2^2 \quad \text{s.t.} \quad x^Tx \le \varepsilon$$

$A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$, $\varepsilon \in \mathbb{R}$, $\varepsilon > 0$ and $x \in \mathbb{R}^n$

→ Solution to the above problem has to satisfy the KKT conditions.

$\nabla_x \mathcal{L}(x,\lambda) = 0$ — 1

$x^Tx \le \varepsilon$ — 2

$\lambda \ge 0$ — 3

$\lambda(x^Tx - \varepsilon) = 0$ — 4

$$\mathcal{L}(x,\lambda) = \frac{1}{2}\|Ax-b\|_2^2 + \lambda(x^Tx - \varepsilon)$$

$\cancel{\nabla_x \mathcal{L}}$ $\quad L(x,\lambda) = \frac{1}{2}(x^TA^T - b^T)(Ax-b) + \lambda(x^Tx - \varepsilon)$

$$\mathcal{L}(x,\lambda) = \frac{1}{2}\left[x^TA^TAx - 2b^TAx + b^Tb\right] + \lambda(x^Tx - \varepsilon)$$

1 $\quad \nabla_x \mathcal{L}(x,\lambda) = \frac{1}{2}\left[2A^TAx - 2A^Tb + 0\right] + 2\lambda x$

$$\nabla_x \mathcal{L}(x,\lambda) = A^TAx - A^Tb + 2\lambda x = A^T(Ax-b) + 2\lambda x$$

2 Case I: $x^Tx < \varepsilon \Rightarrow \lambda = 0$

Unconstrained LS problem -

$$\min_{x \in \mathbb{R}^n} \frac{1}{2} \|Ax - b\|_2^2$$

Closed form solution : $\hat{x} = (A^TA)^{-1} A^T b$

(3) Case 2: $x^T x = \varepsilon$

(a) $\nabla_x \mathcal{L}(x, \lambda) = 0 \Rightarrow A^T(Ax - b) + 2\lambda x = 0$

$\Rightarrow A^TAx - A^Tb + 2\lambda x = 0$

$\Rightarrow (A^TA + 2\lambda I)x = A^Tb$

$\Rightarrow x = (A^TA + 2\lambda I)^{-1} A^T b = h(\lambda)$ [Assuming $(A^TA + 2\lambda I)$ is invertible]

$h(\lambda) = (A^TA + 2\lambda I)^{-1} A^T b$

(b) To prove: $h(\lambda)^T h(\lambda)$ is monotonically decreasing for $\lambda \geq 0$

$h(\lambda)^T h(\lambda) = b^TA (A^TA + 2\lambda I)^{-1} (A^TA + 2\lambda I)^{-1} A^T b$

$\|h(\lambda)\|_2^2 = \|(A^TA + 2\lambda I)^{-1} A^T b\|_2^2$

Now, A^TA is a PSD matrix. ~~Since $\lambda \geq 0$~~

$\|h(\lambda)\|_2^2 = \|(U\Lambda U^T + 2\lambda I)^{-1} A^T b\|_2^2$

$$A^TA = (U\Sigma V^T)^T (U\Sigma V^T) = V\Sigma U^T U \Sigma V^T = V\Sigma^2 V^T$$

$$A^TA = V \begin{bmatrix} \sigma_1^2 & & & \\ & \sigma_2^2 & & \\ & & \ddots & \\ & & & \sigma_n^2 \end{bmatrix} V^T \qquad [\because U^TU = I]$$

Now,

$$\|h(\lambda)\|_2^2 = \|(A^TA + 2\lambda I)^{-1} A^Tb\|_2^2$$

$$= \|(V\Sigma^2V^T + 2\lambda VV^T)^{-1} A^Tb\|_2^2$$

$$= \|(V(\Sigma^2 + 2\lambda I)V^T)^{-1} A^Tb\|_2^2$$

$$= \|V \underbrace{(\Sigma^2 + 2\lambda I)^{-1}}_{\text{diagonal matrix}} V^T A^Tb\|_2^2$$

$$\|h(\lambda)\|_2^2 = \|V(\Sigma^2 + 2\lambda I)^{-1} V^TV\Sigma U^Tb\|_2^2$$

$$= \|V(\Sigma^2 + 2\lambda I)^{-1} \Sigma U^Tb\|_2^2$$

$$(\Sigma^2 + 2\lambda I)^{-1} = \begin{bmatrix} \frac{1}{\sigma_1^2 + 2\lambda} & 0 & \cdots & 0 \\ 0 & \frac{1}{\sigma_2^2 + 2\lambda} & 0 \cdots & 0 \\ & & \ddots & \\ 0 & & & \frac{1}{\sigma_n^2 + 2\lambda} \end{bmatrix}$$

for $\lambda \geq 0$, as we increase λ, $\frac{1}{\sigma_i^2 + 2\lambda}$ will decrease and hence the overall factor would decrease.

Thus, $h(\lambda)^T h(\lambda)$ is monotonically decreasing for $\lambda \geq 0$ (Proved)

assignment0_problem1

September 30, 2022

0.0.1 Implementation

[1]:
```python
import numpy as np
npz = np.load('../data/HW0_P1.npz')
A = npz['A']
b = npz['b']
eps = npz['eps']
A.shape, A.dtype, b.shape, b.dtype, eps
```

[1]:
```
((100, 30), dtype('float64'), (100,), dtype('float64'), array(0.5))
```

[2]:
```python
def compute_hlambda(A, b, eps, lambd):
    term1 = np.matmul(A.T, A) + 2*lambd*np.eye(A.shape[1])
    term2 = np.linalg.inv(term1)
    term3 = np.matmul(A.T, b)

    hlambda = np.matmul(term2, term3)
    return hlambda
```

[3]:
```python
def compute_glambda(A, b, eps, lambd):
    hlambda = compute_hlambda(A, b, eps, lambd)
    glambda = np.matmul(hlambda.T, hlambda) - eps

    return glambda
```

[6]:
```python
def solve(A, b, eps):
    # your implementation here
    g0 = compute_glambda(A, b, eps, 0)
    g10 = compute_glambda(A, b, eps, 10)

    # Line search using Bisection method
    start = 0
    end = 10

    if (compute_glambda(A, b, eps, start) * compute_glambda(A, b, eps, end) >=␣
 ↪0):
        print("You have not assumed right a and b\n")
        return np.zeros(A.shape[1])
```

```
    mid = start
    while ((end-start) >= 0.0001):

        # Find middle point
        mid = (start+end)/2

        glambda_mid = compute_glambda(A, b, eps, mid)

        # Check if middle point is root
        if (glambda_mid == 0.0):
            break

        # Decide the side to repeat the steps
        if (glambda_mid * compute_glambda(A, b, eps, start) < 0):
            end = mid
        else:
            start = mid

    print("Optimal lambda : ","%.4f"%mid)
#     return np.zeros(30)

    return compute_hlambda(A, b, eps, mid)
```

[9]:

```
# Evaluation code, you need to run it, but do not modify
x = solve(A, b, eps)
print("\nEpsilon:", eps)
print("x norm square:", x@x)  # x@x should be close to or less then eps
print("Error:", x@x - eps)
print("\noptimal value:", ((A@x - b)**2).sum())
```

```
Optimal lambda :  0.8370

Epsilon: 0.5
x norm square: 0.5000027626912558
Error: 2.762691255764338e-06

optimal value: 17.220122507015343
```

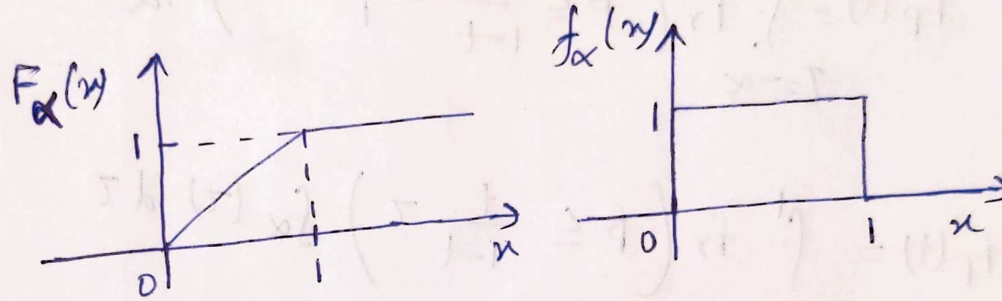
2. Probability

1. $\alpha, \beta \sim U[0,1] \longrightarrow \alpha' = \dfrac{\alpha}{\alpha+\beta}, \quad \beta' = \dfrac{\beta}{\alpha+\beta}$

$P = \alpha' A + \beta' B$ then P is not uniformly distributed between A and B.

Let $A = 0$, $B = 1$

$\alpha \sim U[0,1]$



CDF of α

PDF of α

(same PDF, CDF plots for β as well)

$$P = \alpha' A + \beta' B = \frac{\beta}{\alpha+\beta}$$

$$F_P(t) = Pr(P \leq t) = Pr\left(\frac{\beta}{\alpha+\beta} \leq t\right)$$

since $\beta \in [0,1]$ and $\alpha \in [0,1]$ and $\beta \leq \alpha + \beta$

$\hookrightarrow P = \dfrac{\beta}{(\alpha+\beta)} \in [0,1]$. Thus P is a random variable that takes values in the range $[0,1]$

$$F_P(t) = \begin{cases} 0 & t < 0 \\ 1 & t \geq 1 \\ ?? & 0 \leq t < 1 \end{cases}$$

[Need to estimate CDF only when $0 \leq t < 1$]

(2)

$$F_P(t) = Pr\left(\frac{\beta}{\alpha+\beta} \le t\right) = Pr(\beta \le t\alpha + t\beta)$$

$$= Pr(\beta(1-t) \le t\alpha) = Pr\left(\beta \le \frac{t\alpha}{1-t}\right)$$

$$= F_\beta\left(\frac{t\alpha}{1-t}\right) \qquad \begin{cases} 0 \le t < 1 \\ \Rightarrow 0 \le (1-t) \le 1 \end{cases}$$

$$F_P(t) = \int_{\tau=-\infty}^{\infty} Pr\left(\beta \le \frac{t}{1-t}\alpha \,\middle|\, \alpha = \tau\right) f_\alpha(\tau)\, d\tau$$

$$F_P(t) = \int_{\tau=0}^{1} Pr\left(\beta \le \frac{t}{1-t}\tau\right) f_\alpha(\tau)\, d\tau$$

$$= \int_{\tau=0}^{1} Pr\left(\beta \le \frac{t}{1-t}\tau\right)(1)\, d\tau = \int_{\tau=0}^{1} F_\beta\left(\frac{t}{1-t}\tau\right) d\tau$$

Substitute $\frac{t}{(1-t)}\tau = x$, $\Rightarrow d\tau = \frac{(1-t)}{t}dx$

$$F_P(t) = \int_{x=0}^{\frac{t}{1-t}} Pr(\beta \le x)\frac{(1-t)}{t}dx = \left(\frac{1-t}{t}\right)\int_{x=0}^{\frac{t}{1-t}} Pr(\beta \le x)\, dx$$

when $\frac{t}{(1-t)} \le 1 \Rightarrow t \le (1-t) \Rightarrow 0 \le t \le \frac{1}{2}$ then

$$F_P(t) = \left(\frac{1-t}{t}\right)\int_{x=0}^{\frac{t}{1-t}} x\, dx = \frac{1}{2}\left(\frac{1-t}{t}\right)\frac{t^2}{(1-t)^2} = \frac{t}{2(1-t)}$$

when $\frac{t}{(1-t)} > 1 \Rightarrow t > 1-t \Rightarrow \frac{1}{2} < t < 1$ then

$$F_P(t) = \left(\frac{1-t}{t}\right)\left[\int_0^1 x\,dx + \int_1^{\frac{t}{1-t}} 1 \cdot dx\right]$$

$$F_P(t) = \frac{(1-t)}{t}\left[\frac{1}{2} + \frac{t}{(1-t)} - 1\right] = \left(\frac{1-t}{t}\right)\left[\frac{t}{(1-t)} - \frac{1}{2}\right]$$

$$= \frac{3t-1}{2t} = \frac{3}{2} - \frac{1}{2t}$$

Thus,

$$F_P(t) = Pr(P \le t) = \begin{cases} 0 & t < 0 \\ \frac{t}{2(1-t)} & 0 \le t \le \frac{1}{2} \\ \frac{3}{2} - \frac{1}{2t} & \frac{1}{2} < t < 1 \\ 1 & t \ge 1 \end{cases}$$

Cumulative density function of P.

$F_P(t=0) = 0 \qquad F_P(t=0.5) = \frac{1}{2}$

$$f_P(t) = \frac{d}{dt} Pr(P \le t) = \begin{cases} 0 & t < 0 \\ \frac{1}{2(1-t)^2} & 0 \le t \le \frac{1}{2} \\ \frac{1}{2t^2} & \frac{1}{2} < t < 1 \\ 0 & t \ge 1 \end{cases}$$

Probability density function of P

$f_p(t=0) = \frac{1}{2}$ $\quad f_p(t=0.5) = 2$

(2) To prove: p' is uniformly distributed inside ABDC.

$$p' = A + \alpha(B-A) + \beta(C-A)$$

Let $A = \begin{bmatrix} a_x \\ a_y \end{bmatrix}$ $B = \begin{bmatrix} b_x \\ b_y \end{bmatrix}$ $C = \begin{bmatrix} c_x \\ c_y \end{bmatrix}$ $p' = \begin{bmatrix} p_x \\ p_y \end{bmatrix}$

$\alpha, \beta \sim U[0,1]$ $\quad$ Let $X = \begin{bmatrix} \alpha \\ \beta \end{bmatrix}$

$$p_x = a_x + \alpha(b_x - a_x) + \beta(c_x - a_x)$$
$$p_y = a_y + \alpha(b_y - a_y) + \beta(c_y - a_y)$$

$$J = \frac{\delta p'}{\delta X} = \begin{bmatrix} \delta p_x/\delta\alpha & \delta p_x/\delta\beta \\ \delta p_y/\delta\alpha & \delta p_y/\delta\beta \end{bmatrix} \qquad X = \begin{bmatrix} \alpha \\ \beta \end{bmatrix}$$

$$J = \begin{bmatrix} b_x - a_x & c_x - a_x \\ b_y - a_y & c_y - a_y \end{bmatrix}$$

$$\det(J) = (b_x - a_x)(c_y - a_y) - (c_x - a_x)(b_y - a_y)$$

$$\det(J^{-1}) = \frac{1}{\det(J)} = \frac{1}{(b_x - a_x)(c_y - a_y) - (c_x - a_x)(b_y - a_y)}$$

For P' to be uniformly distributed inside parallelogram ABDC, the PDF of P' should be uniform inside ABDC and 0 everywhere outside of it.

PDF of $P' = f_{P'}(t) = f_{\alpha,\beta}(H^{-1}(t))\,|\det(J^{-1})|$, $t \in \mathbb{R}^2$

$$\begin{bmatrix} p_x \\ p_y \end{bmatrix} = \begin{bmatrix} a_x + \alpha(b_x - a_x) + \beta(c_x - a_x) \\ a_y + \alpha(b_y - a_y) + \beta(c_y - a_y) \end{bmatrix}$$

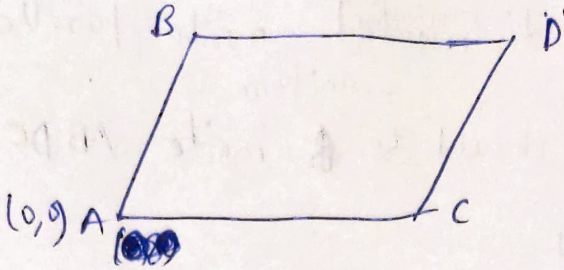
$$\begin{bmatrix} p_x - a_x \\ p_y - a_y \end{bmatrix} = \begin{bmatrix} b_x - a_x & c_x - a_x \\ b_y - a_y & c_y - a_y \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix}$$

$$\begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} b_x - a_x & c_x - a_x \\ b_y - a_y & c_y - a_y \end{bmatrix}^{-1} \begin{bmatrix} p_x - a_x \\ p_y - a_y \end{bmatrix}$$

$$\begin{bmatrix} \alpha \\ \beta \end{bmatrix} = H^{-1}(p) = J^{-1} \begin{bmatrix} p_x - a_x \\ p_y - a_y \end{bmatrix}$$

PDF of $P' = f_{P'}(p) = f_{\alpha,\beta}(H^{-1}(p))\,|\det(J^{-1})|$

$$f_{P'}(p) = f_{\alpha,\beta}\left(J^{-1}\begin{bmatrix} p_x - a_x \\ p_y - a_y \end{bmatrix}\right) |\det(J^{-1})|$$



Let $A = (0,0) = (a_x, a_y)$ [result holds for any general A]

then, $\det(J^{-1}) = \dfrac{1}{\det(J)} = \dfrac{1}{b_x c_y - c_x b_y}$

$$f_{P'}(p) = f_{\alpha,\beta}\left(J^{-1}\begin{bmatrix} p_x \\ p_y \end{bmatrix}\right)\frac{1}{|b_x c_y - c_x b_y|}$$

$$f_{P'}(p) = f_{\alpha,\beta}\left(\begin{bmatrix} b_x & c_x \\ b_y & c_y \end{bmatrix}^{-1}\begin{bmatrix} p_x \\ p_y \end{bmatrix}\right)\frac{1}{|b_x c_y - c_x b_y|}$$

$$= f_{\alpha,\beta}\left(\frac{1}{(b_x c_y - b_y c_x)}\begin{bmatrix} c_y & -c_x \\ -b_y & b_x \end{bmatrix}\begin{bmatrix} p_x \\ p_y \end{bmatrix}\right)\frac{1}{|b_x c_y - c_x b_y|}$$

$$f_{P'}(p) = f_{\alpha,\beta}\left(\frac{1}{\det(J)}\begin{bmatrix} c_y p_x - c_x p_y \\ -b_y p_x + b_x p_y \end{bmatrix}\right)\frac{1}{|\det(J)|}$$

$$= f_\alpha\left(\frac{1}{\det(J)}(c_y p_x - c_x p_y)\right) f_\beta\left(\frac{1}{\det(J)}(-b_y p_x + b_x p_y)\right)\frac{1}{|\det(J)|}$$

$$f_{P'}(p) = f_\alpha\left(\frac{p_x c_y - p_y c_x}{b_x c_y - b_y c_x}\right) f_\beta\left(\frac{p_y b_x - b_y p_x}{b_x c_y - b_y c_x}\right)\frac{1}{|\det(J)|}$$

If point $p = \begin{bmatrix} p_x \\ p_y \end{bmatrix}$ lies within the parallelogram ABDC,

then the quantities $0 \le \dfrac{p_x c_y - p_y c_x}{b_x c_y - b_y c_x} \le 1$ and

$$0 \le \frac{p_y b_x - b_y p_x}{b_x c_y - b_y c_x} \le 1$$

Thus, for any point p inside the parallelogram

$$f_{p'}(p) = f_\alpha(\cdot)\, f_\beta(\cdot)\, \frac{1}{|\det(J)|}$$

$$= 1 \cdot 1 \cdot \frac{1}{|\det(J)|} = \frac{1}{|\det(J)|}$$

for any point p outside ABDC, $f_{p'}(p) = 0$.

Proved.

p' uniformly distributed inside ABDC.

assignment0_problem2

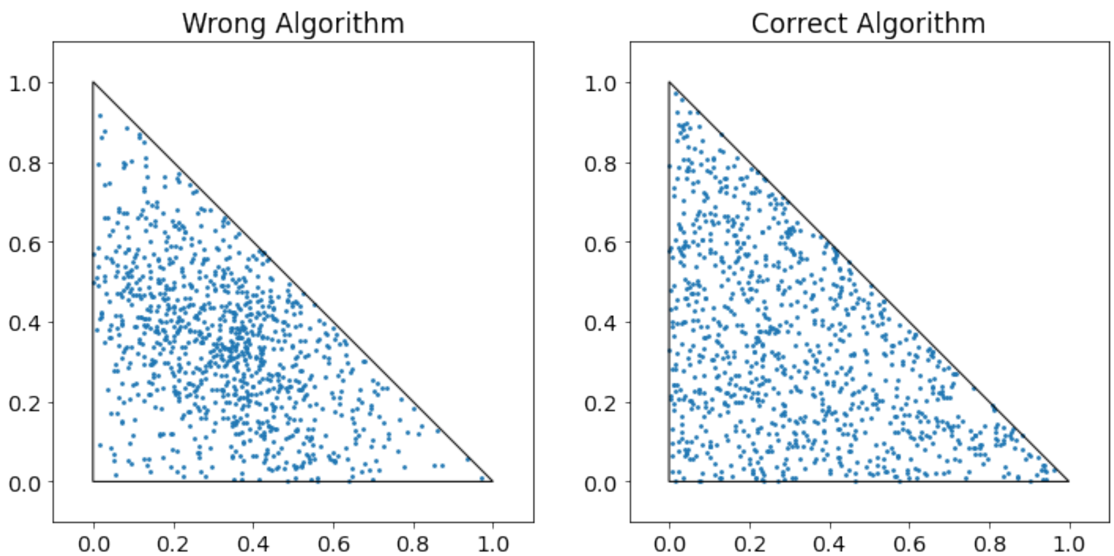
September 30, 2022

0.1 Problem 2

[1]:

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.patches import Polygon

pts = np.array([[0,0], [0,1], [1,0]])

plt.rcParams.update({'font.size': 14})
fig = plt.figure(figsize=(12,12))
title_lst = ["Wrong", "Correct"]

def draw_background(index):
    # DRAW THE TRIANGLE AS BACKGROUND
    p = Polygon(pts, closed=True, facecolor=(1,1,1,0), edgecolor=(0, 0, 0))

    plt.subplot(1, 2, index + 1)

    ax = plt.gca()
    ax.set_aspect('equal')
    ax.add_patch(p)
    ax.set_xlim(-0.1,1.1)
    ax.set_ylim(-0.1,1.1)
    plt.title(title_lst[index] + " Algorithm")

# plt.suptitle("Comparison: Wrong vs Correct Algorithm")

# YOUR CODE HERE
NUM_PTS = 1000

# wrong algorithm for sampling uniform points inside triangle
wrong_pts_lst = []
for i in range(NUM_PTS):
    alpha = np.random.uniform(0, 1)
    beta = np.random.uniform(0, 1)
    gamma = np.random.uniform(0, 1)

    norm_factor = alpha + beta + gamma
```

```
    alpha_dash = alpha / norm_factor
    beta_dash = beta / norm_factor
    gamma_dash = gamma / norm_factor

    wrong_pts_lst.append(alpha_dash * pts[0] + beta_dash * pts[1] + gamma_dash␣
↪* pts[2])

wrong_pts_lst = np.array(wrong_pts_lst)
# print(wrong_pts_lst.shape)

# correct algorithm for sampling uniform points inside triangle
correct_pts_lst = []
for i in range(NUM_PTS):
    alpha = np.random.uniform(0, 1)
    beta = np.random.uniform(0, 1)

    pot_pt = pts[0] + alpha * (pts[1] - pts[0]) + beta * (pts[2] - pts[0])

    if (pot_pt[1] > 0 and pot_pt[0] > 0 and (pot_pt[0] + pot_pt[1] - 1) < 0):
        correct_pts_lst.append(pot_pt)
    else:
        correct_pts_lst.append(pts[1] + pts[2] - pot_pt)

correct_pts_lst = np.array(correct_pts_lst)
# print(correct_pts_lst.shape)

draw_background(0)
# REPLACE THE FOLLOWING LINE USING YOUR DATA (incorrect method)
# plt.scatter(0.4+0.2*np.random.randn(1000), 0.4+0.2*np.random.randn(1000), s=3)
plt.scatter(wrong_pts_lst[:,0], wrong_pts_lst[:,1], s=3)

draw_background(1)
# REPLACE THE FOLLOWING LINE USING YOUR DATA (correct method)
# plt.scatter(0.4+0.2*np.random.randn(1000), 0.4+0.2*np.random.randn(1000), s=3)
plt.scatter(correct_pts_lst[:,0], correct_pts_lst[:,1], s=3)

plt.show()
```


Wrong Algorithm
Correct Algorithm
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

assignment0_problem3

September 30, 2022

[25]:
```
import time

import numpy as np
import matplotlib.pyplot as plt

import torch
import torch.nn as nn
import torch.optim as optim

from torchvision import transforms
```

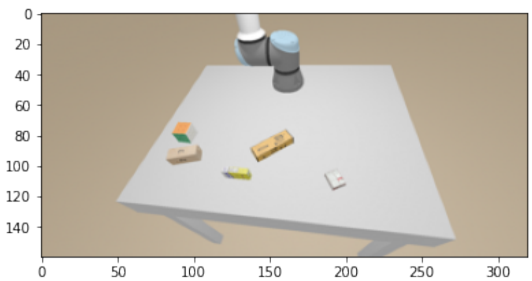
[26]:
```
# train_data = np.load("../data/train.npz")
train_data = np.load("/content/drive/MyDrive/academics_and_research/UCSD/
↪Fall_22/CSE291/train.npz")
train_images = train_data["images"]  # array with shape (N,Width,Height,3)
train_edges = train_data["edges"]  # array with shape (N,Width,Height)

print("Before Processing")
print("Train Images - shape:", train_images.shape, ", Max:", train_images.
↪max(), ", Min:", train_images.min())
print("Train Edges - shape:", train_edges.shape, ", Unique:", np.
↪unique(train_edges))

train_images = train_images/255.0
train_edges = train_edges//255

print("\nAfter Processing")
print("Train Images - shape:", train_images.shape, ", Max:", train_images.
↪max(), ", Min:", train_images.min())
print("Train Edges - shape:", train_edges.shape, ", Unique:", np.
↪unique(train_edges))
```

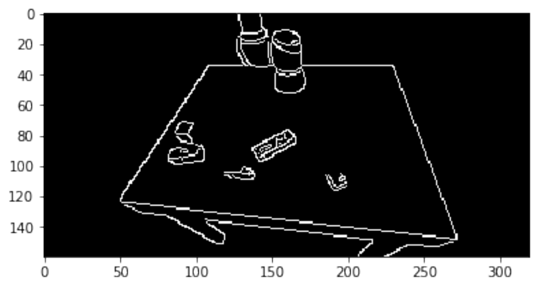
```
Before Processing
Train Images - shape: (1000, 160, 320, 3) , Max: 255 , Min: 0
Train Edges - shape: (1000, 160, 320) , Unique: [  0 255]

After Processing
Train Images - shape: (1000, 160, 320, 3) , Max: 1.0 , Min: 0.0
```

```
Train Edges - shape: (1000, 160, 320) , Unique: [0 1]
```

[3]:
```python
fig = plt.figure(figsize=(14, 10))
plt.subplot(1, 2, 1)
plt.imshow(train_images[0])
plt.subplot(1, 2, 2)
plt.imshow(train_edges[0], cmap="gray", interpolation='nearest')
```

[3]:
```
<matplotlib.image.AxesImage at 0x7f1d6a7aaf90>
```





[27]:
```python
mytransform = transforms.Compose([transforms.Normalize(mean=(0.5, 0.5, 0.5),␣
 ↪std=(0.5, 0.5, 0.5))])

def process_train_data(images, edges):
    images_tensor = torch.from_numpy(images)
    images_tensor = images_tensor.type(torch.FloatTensor)
    images_tensor = images_tensor.permute(0, 3, 1, 2)

    # mean_image = torch.mean(images_tensor, dim=0)
    # std_image = torch.std(images_tensor, dim=0)

    # mytransform = transforms.Compose([transforms.Normalize(mean=mean_image,␣
 ↪std=(1.0, 1.0, 1.0))])
    images_tensor = mytransform(images_tensor)

    edges_tensor = torch.from_numpy(edges)
    edges_tensor = edges_tensor.type(torch.FloatTensor)
    edges_tensor = edges_tensor.unsqueeze(dim=1)

    return images_tensor, edges_tensor
```

[28]:
```python
train_images_tensor, train_edges_tensor = process_train_data(train_images,␣
 ↪train_edges)

print("Train Images Tensor - shape:", train_images_tensor.shape, ", Max:",␣
 ↪train_images_tensor.max(), ", Min:", train_images_tensor.min())
```

```
print("Train Edges Tensor - shape:", train_edges_tensor.shape, ", Unique:", np.
 ↪unique(train_edges_tensor))
```

```
Train Images Tensor - shape: torch.Size([1000, 3, 160, 320]) , Max: tensor(1.) ,
Min: tensor(-1.)
Train Edges Tensor - shape: torch.Size([1000, 1, 160, 320]) , Unique: [0. 1.]
```

[29]:
```
NUM_TRAIN_SAMPLES, NUM_INPUT_CHANNELS, INPUT_IMAGE_HEIGHT, INPUT_IMAGE_WIDTH =␣
 ↪train_images_tensor.shape
NUM_TEST_SAMPLES = 4
NUM_EPOCHS = 50
TRAIN_BATCH_SIZE = 25
TEST_BATCH_SIZE = 1

INIT_LR = 5e-4
LR_STEP_SIZE = 20   # How often to decrease learning rate by gamma factor
SCHEDULER_GAMMA = 0.1  # LR is multiplied by gamma on schedule
```

[30]:
```
# define device type - cuda:0 or cpu - to be used for training and evaluation
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Device:", device)

# determine if we will be pinning memory during data loading
kwargs = {'num_workers': 4, 'pin_memory': True} if device.type == "cuda" else {}

# Additional Info when using cuda
if device.type == 'cuda':
    print("Number of GPU devices:", torch.cuda.device_count())
    print("GPU device name:", torch.cuda.get_device_name(0))
    print('Memory Usage:')
    print('Allocated:', round(torch.cuda.memory_allocated(0)/1024**3,1), 'GB')
    print('Cached:   ', round(torch.cuda.memory_reserved(0)/1024**3,1), 'GB')
```

```
Device: cuda
Number of GPU devices: 1
GPU device name: Tesla T4
Memory Usage:
Allocated: 0.0 GB
Cached:    1.2 GB
```

[31]:
```
# Build and train your neural network here, optionally save the weights
class Block(nn.Module):
    def __init__(self, in_channels, out_channels):
        super().__init__()
        # store the convolution and RELU layers

        self.conv1 = nn.Conv2d(in_channels, out_channels, 3)
```

```
        self.relu = nn.ReLU()
        self.conv2 = nn.Conv2d(out_channels, out_channels, 3)

    def forward(self, x):
        # apply CONV => RELU => CONV block to the inputs and return it
        output = self.conv2(self.relu(self.conv1(x)))
        return output
```

[32]:
```
class Encoder(nn.Module):
    def __init__(self, channels=(3, 16, 32, 64)):
        super().__init__()
        # store the encoder blocks and maxpooling layer

        block_list = [Block(channels[i], channels[i+1]) for i in␣
 ↪range(len(channels)-1)]
        self.enc_block = nn.ModuleList(block_list)

        self.pool = nn.MaxPool2d(kernel_size=2)

    def forward(self, x):
        # initialize an empty list to store the intermediate outputs
        block_outputs = []

        # loop through the encoder blocks
        for block in self.enc_block:
            # pass the inputs through the current encoder block, store
            # the outputs, and then apply maxpooling on the output

            x = block(x)
            block_outputs.append(x)
            x = self.pool(x)

        # return the list containing the intermediate outputs
        return block_outputs
```

[33]:
```
class Decoder(nn.Module):
    def __init__(self, channels=(64, 32, 16)):
        super().__init__()
        # initialize the number of channels, upsampler blocks, and decoder␣
 ↪blocks
        self.channels = channels

        layer_list = [nn.ConvTranspose2d(channels[i], channels[i+1], 2, 2) for␣
 ↪i in range(len(channels)-1)]
        self.upconv = nn.ModuleList(layer_list)
```

```
        block_list = [Block(channels[i], channels[i+1]) for i in␣
 ↪range(len(channels)-1)]
        self.dec_blocks = nn.ModuleList(block_list)

    def crop(self, enc_features, x):
        # grab the dimensions of the inputs, and crop the encoder
        # features to match the dimensions
        _, _, height, width = x.shape
        enc_features = transforms.CenterCrop([height, width])(enc_features)
#          enc_features = nn.functional.center_crop(enc_features, [height,␣
 ↪width])

        # return the cropped features
        return enc_features

    def forward(self, x, enc_features):
        # loop through the number of channels
        for i in range(len(self.channels)-1):
            # pass the inputs through the upsampler blocks
            x = self.upconv[i](x)

            # crop the current features from the encoder blocks,
            # concatenate them with the current upsampled features,
            # and pass the concatenated output through the current
            # decoder block

            enc_feat = self.crop(enc_features[i], x)
            x = torch.cat([x, enc_feat], dim=1)
            x = self.dec_blocks[i](x)

        # return the final decoder output
        return x
```

[34]:
```
class UNet(nn.Module):
    def __init__(self, enc_channels=(3, 16, 32, 64), dec_channels=(64, 32, 16),␣
 ↪num_classes=1,
                 retain_dim=True, out_size=(INPUT_IMAGE_HEIGHT,␣
 ↪INPUT_IMAGE_WIDTH)):
        super().__init__()

        # initialize the encoder and decoder
        self.encoder = Encoder(enc_channels)
        self.decoder = Decoder(dec_channels)

        # initialize the regression head and store the class variables
        self.head = nn.Conv2d(dec_channels[-1], num_classes, 1)
        self.retain_dim = retain_dim
```

```python
        self.out_size = out_size

    def forward(self, x):
        # grab the features from the encoder
        enc_features = self.encoder(x)

        # pass the encoder features through decoder making sure that
        # their dimensions are suited for concatenation
        dec_features = self.decoder(enc_features[::-1][0], enc_features[::-1][1:
 ↪])

        # pass the decoder features through the regression head to
        # obtain the segmentation mask
        edge_map = self.head(dec_features)

        # check to see if we are retaining the original output
        # dimensions and if so, then resize the output to match them
        if self.retain_dim:
            edge_map = nn.functional.interpolate(edge_map, self.out_size)

        # return the edge map
        return edge_map
```

[35]:
```python
# initialize our UNet model
unet_model = UNet().to(device)

# initialize loss function and optimizer
loss_criterion = nn.BCEWithLogitsLoss().to(device)
# loss_criterion = nn.CrossEntropyLoss().to(device)

optimizer = optim.Adam(unet_model.parameters(), lr=INIT_LR)
# optimizer = optim.SGD(unet_model.parameters(), lr=INIT_LR)

scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=LR_STEP_SIZE,␣
 ↪gamma=SCHEDULER_GAMMA)

# calculate steps per epoch for training and test set
num_train_steps = len(train_images_tensor) // TRAIN_BATCH_SIZE
# print(num_train_steps)

# initialize a dictionary to store training history
train_history = {"train_loss": [], "test_loss": []}
```

[36]:
```python
def generate_train_batches():
    num_batches = NUM_TRAIN_SAMPLES // TRAIN_BATCH_SIZE

    for i in range(num_batches):
```

```
        yield i, train_images_tensor[i*TRAIN_BATCH_SIZE:
↪(i+1)*TRAIN_BATCH_SIZE], train_edges_tensor[i*TRAIN_BATCH_SIZE:
↪(i+1)*TRAIN_BATCH_SIZE]
```

[37]:

```
start_time = time.time()

for epoch in range(NUM_EPOCHS):
    # set the model in training mode
    unet_model.train()

    # initialize the total training and validation loss
    total_train_loss = 0
    total_test_loss = 0

    train_batches = generate_train_batches()

    # loop over the training set
    for itr, x_batch, y_batch in train_batches:
        # send the input to the device
        x_batch = x_batch.to(device)
        y_batch = y_batch.to(device)

        # print(x_batch.shape, y_batch.shape)

        # perform a forward pass and calculate the training loss
        y_pred = unet_model(x_batch)
        loss = loss_criterion(y_pred, y_batch)

        # first, zero out any previously accumulated gradients, then
        # perform backpropagation, and then update model parameters
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        # add the loss to the total training loss so far
        total_train_loss += loss

#     scheduler.step()

    # switch off autograd
#     with torch.no_grad():
#         # set the model in evaluation mode
#         unet_model.eval()

#         test_batches = generate_test_batches()
#         # loop over the validation/test set
#         for itr, x_batch in test_batches:
```

```
#               # send the input to the device
#               x_batch = x_batch.to(device)

#               # make the predictions and calculate the validation loss
#               y_pred = unet_model(x_batch)

    # calculate the average training and validation loss
    avg_train_loss = total_train_loss / num_train_steps

    # update our training history
    train_history["train_loss"].append(avg_train_loss.cpu().detach().numpy())

    # print the model training and validation information
    print("[INFO] EPOCH: {}/{}, Train Loss: {:.4f}".format(epoch + 1,␣
 ↪NUM_EPOCHS, avg_train_loss))

time_elapsed = (time.time() - start_time)/60.0
print("Total training time: {:.4f} min".format(time_elapsed))
```

```
[INFO] EPOCH: 1/50, Train Loss: 0.3317
[INFO] EPOCH: 2/50, Train Loss: 0.1584
[INFO] EPOCH: 3/50, Train Loss: 0.1380
[INFO] EPOCH: 4/50, Train Loss: 0.1319
[INFO] EPOCH: 5/50, Train Loss: 0.1268
[INFO] EPOCH: 6/50, Train Loss: 0.1231
[INFO] EPOCH: 7/50, Train Loss: 0.1212
[INFO] EPOCH: 8/50, Train Loss: 0.1187
[INFO] EPOCH: 9/50, Train Loss: 0.1170
[INFO] EPOCH: 10/50, Train Loss: 0.1144
[INFO] EPOCH: 11/50, Train Loss: 0.1126
[INFO] EPOCH: 12/50, Train Loss: 0.1100
[INFO] EPOCH: 13/50, Train Loss: 0.1077
[INFO] EPOCH: 14/50, Train Loss: 0.1055
[INFO] EPOCH: 15/50, Train Loss: 0.1050
[INFO] EPOCH: 16/50, Train Loss: 0.1029
[INFO] EPOCH: 17/50, Train Loss: 0.1012
[INFO] EPOCH: 18/50, Train Loss: 0.0997
[INFO] EPOCH: 19/50, Train Loss: 0.0979
[INFO] EPOCH: 20/50, Train Loss: 0.0969
[INFO] EPOCH: 21/50, Train Loss: 0.0956
[INFO] EPOCH: 22/50, Train Loss: 0.0939
[INFO] EPOCH: 23/50, Train Loss: 0.0925
[INFO] EPOCH: 24/50, Train Loss: 0.0904
[INFO] EPOCH: 25/50, Train Loss: 0.0897
[INFO] EPOCH: 26/50, Train Loss: 0.0881
[INFO] EPOCH: 27/50, Train Loss: 0.0865
[INFO] EPOCH: 28/50, Train Loss: 0.0857
```

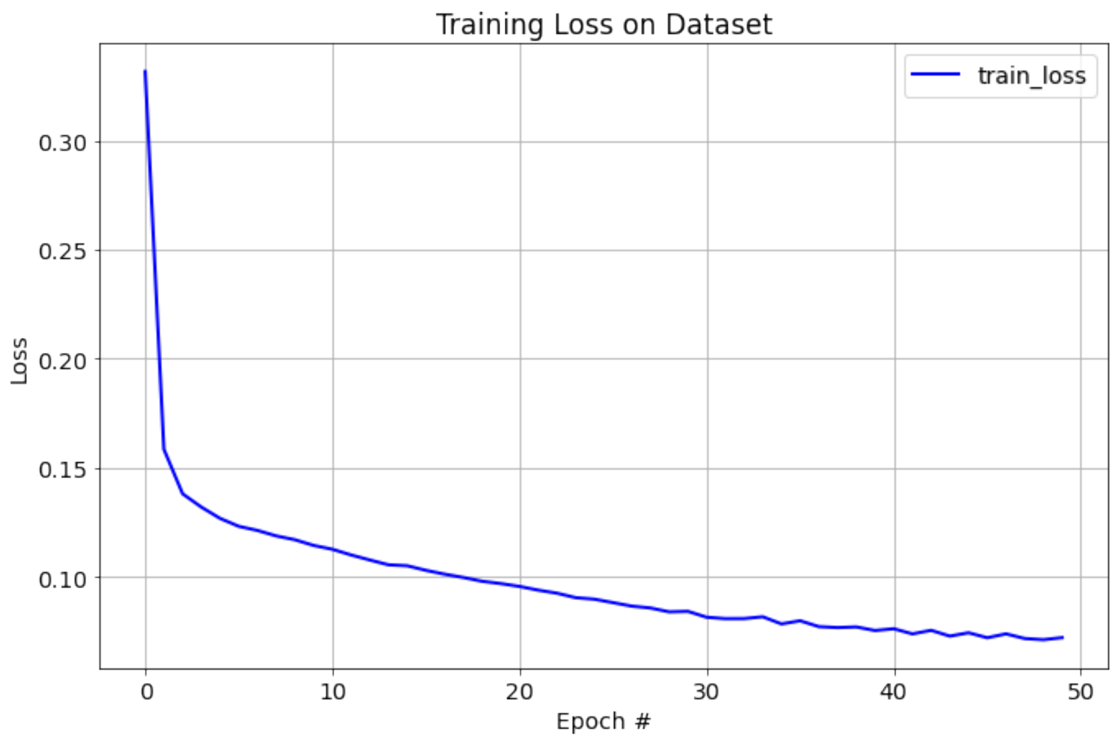
```
[INFO] EPOCH: 29/50, Train Loss: 0.0839
[INFO] EPOCH: 30/50, Train Loss: 0.0841
[INFO] EPOCH: 31/50, Train Loss: 0.0814
[INFO] EPOCH: 32/50, Train Loss: 0.0808
[INFO] EPOCH: 33/50, Train Loss: 0.0808
[INFO] EPOCH: 34/50, Train Loss: 0.0816
[INFO] EPOCH: 35/50, Train Loss: 0.0784
[INFO] EPOCH: 36/50, Train Loss: 0.0798
[INFO] EPOCH: 37/50, Train Loss: 0.0771
[INFO] EPOCH: 38/50, Train Loss: 0.0767
[INFO] EPOCH: 39/50, Train Loss: 0.0770
[INFO] EPOCH: 40/50, Train Loss: 0.0753
[INFO] EPOCH: 41/50, Train Loss: 0.0762
[INFO] EPOCH: 42/50, Train Loss: 0.0738
[INFO] EPOCH: 43/50, Train Loss: 0.0754
[INFO] EPOCH: 44/50, Train Loss: 0.0728
[INFO] EPOCH: 45/50, Train Loss: 0.0743
[INFO] EPOCH: 46/50, Train Loss: 0.0720
[INFO] EPOCH: 47/50, Train Loss: 0.0738
[INFO] EPOCH: 48/50, Train Loss: 0.0717
[INFO] EPOCH: 49/50, Train Loss: 0.0711
[INFO] EPOCH: 50/50, Train Loss: 0.0721
Total training time: 4.2565 min
```

[38]:

```python
# plot the training loss
# plt.style.use("ggplot")
plt.rcParams.update({'font.size': 14})
plt.figure(figsize=(11, 7))
plt.plot(train_history["train_loss"], label="train_loss", linewidth=2,␣
 ↪color='b')
# plt.plot(train_history["test_loss"], label="test_loss")
plt.title("Training Loss on Dataset")
plt.xlabel("Epoch #")
plt.ylabel("Loss")
plt.grid()
plt.legend(loc="upper right")
```

[38]:

```
<matplotlib.legend.Legend at 0x7f1d6065ea50>
```



[39]:
```
# Test on the testing set
# test_data = np.load("../data/test.npz")
test_data = np.load("/content/drive/MyDrive/academics_and_research/UCSD/Fall_22/
↪CSE291/test.npz")
test_images = test_data["images"]

print("Before Processing")
print("Test Images - shape:", test_images.shape, ", Max:", test_images.max(),␣
↪", Min:", test_images.min())

test_images = test_images/255.0

print("\nAfter Processing")
print("Test Images - shape:", test_images.shape, ", Max:", test_images.max(),␣
↪", Min:", test_images.min())
```

```
Before Processing
Test Images - shape: (4, 160, 320, 3) , Max: 255 , Min: 7

After Processing
Test Images - shape: (4, 160, 320, 3) , Max: 1.0 , Min: 0.027450980392156862
```

[40]:
```
def process_test_data(images, mean_image=None):
    images_tensor = torch.from_numpy(images)
    images_tensor = images_tensor.type(torch.FloatTensor)
    images_tensor = images_tensor.permute(0, 3, 1, 2)

    # images_tensor = (images_tensor - mean_image)
    images_tensor = mytransform(images_tensor)

    return images_tensor
```

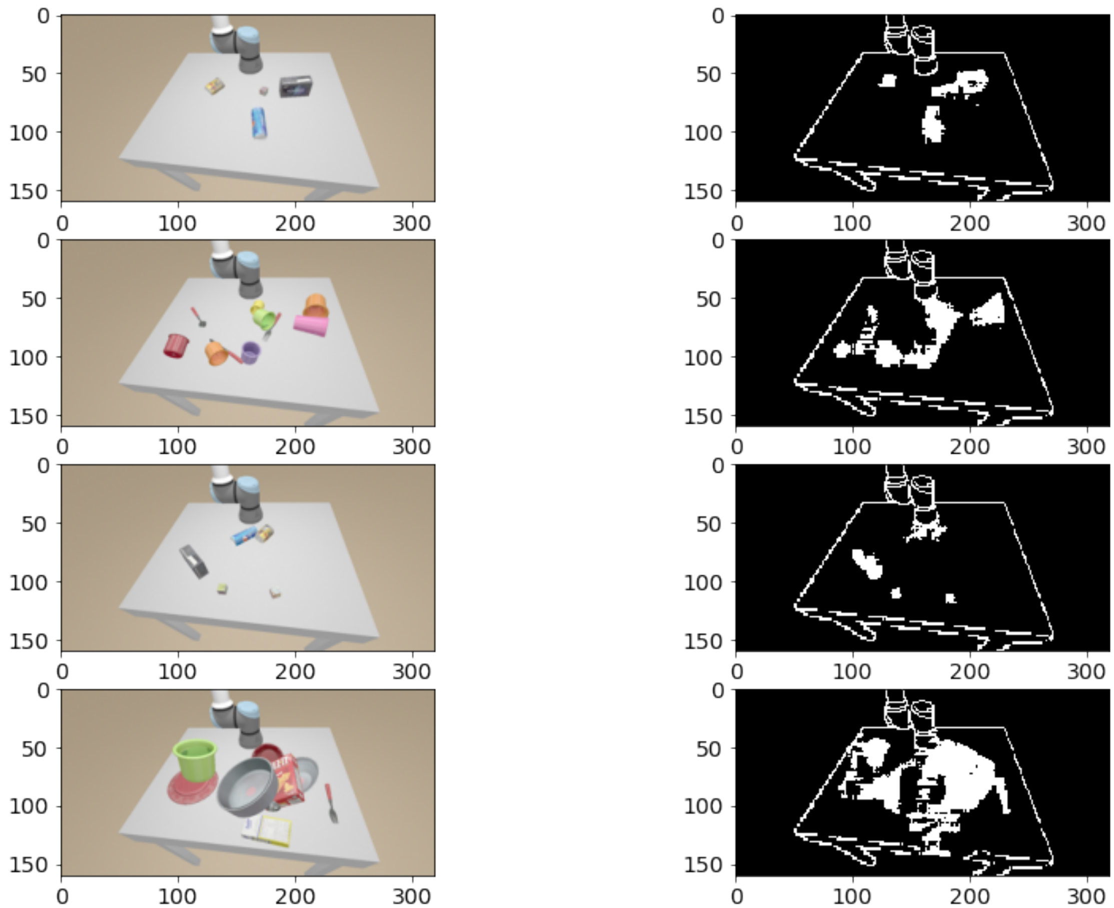
[41]:
```
test_images_tensor = process_test_data(test_images)
print("Test Images Tensor - shape:", test_images_tensor.shape, ", Max:",␣
 ↪test_images_tensor.max(), ", Min:", test_images_tensor.min())
```

```
Test Images Tensor - shape: torch.Size([4, 3, 160, 320]) , Max: tensor(1.) ,
Min: tensor(-0.9451)
```

[46]:
```
test_images_tensor = test_images_tensor.to(device)

pred_outputs = unet_model(test_images_tensor).squeeze()
pred_outputs = torch.sigmoid(pred_outputs)

pred_outputs = pred_outputs.cpu().detach().numpy()

# filter out the weak predictions and convert them to integers
THRESHOLD = 0.15
pred_edge_mask = (pred_outputs > THRESHOLD) * 255
pred_edge_mask = pred_edge_mask.astype(np.uint8)

print(pred_edge_mask.shape, pred_edge_mask.max(), pred_edge_mask.min(), np.
 ↪unique(pred_edge_mask))
```

```
(4, 160, 320) 255 0 [  0 255]
```

[47]:
```
plt.figure(figsize=(14, 10))

for i, img in enumerate(test_images[:4]):
    plt.subplot(4, 2, i * 2 + 1)
    plt.imshow(img)

    plt.subplot(4, 2, i * 2 + 2)
    # edge = evaluate your model on the test set, replace the following line

#     edge = np.zeros(img.shape[:2])
    plt.imshow(pred_edge_mask[i], cmap="gray", interpolation="nearest")
```



References:

1. https://towardsdatascience.com/creating-and-training-a-u-net-model-with-pytorch-for-2d-3d-semantic-segmentation-model-building-6ab09d6a0862
2. https://pyimagesearch.com/2021/11/08/u-net-training-image-segmentation-models-in-pytorch/