# Exercise 8.1

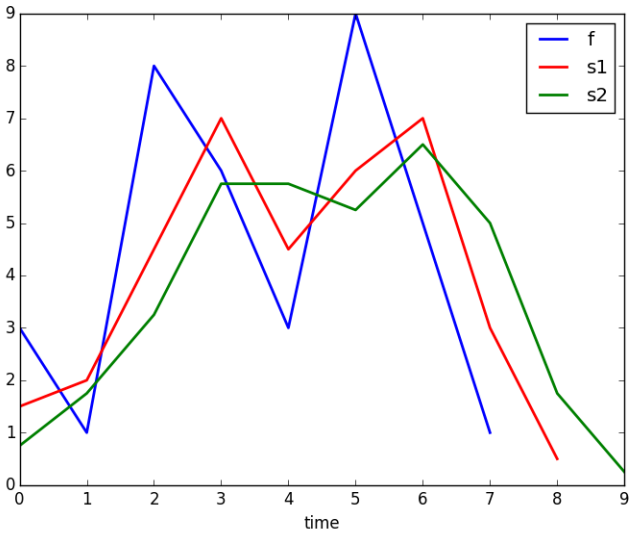
a)

$$\begin{aligned} s1[t] &= (f * k)[t] \\ &= f[t] * k[0] + f[t-1] * k[1] = \frac{1}{2}(f[t] + f[t-1]) \end{aligned}$$

Hence, $s1 = \frac{1}{2}[3, 1+3, 8+1, 6+8, 3+6, 9+3, 5+9, 1+5, 1] = [1.5, 2, 4.5, 7, 4.5, 6, 7, 3, 0.5]$
b)
The same we have:

$$\begin{aligned} s2 &= s1 * k \\ &= [0.75, 1.75, 3.25, 5.75, 5.75, 5.25, 6.5, 5, 1.75, 0.25] \end{aligned}$$

c)




- Convolving f for n times with n $\to \infty$, the resulted signal would be averaged and be closed to zero. With n $= \infty$, the resulted signal would be 0.
d)

$$\begin{aligned} k' &= k * k = [0.25, 0.5, 0.25] \\ s3 &= f * k' = [0.75, 1.75, 3.25, 5.75, 5.75, 5.25, 6.5, 5., 1.75, 0.25] \end{aligned}$$

e)
We can see that s2 = s3. This is due the associative property of convolution: $s2 = (f * k) * k = f * (k * k) = s3$
Proof: given arbitrary signal $a, b, c$, we have:

$$\begin{aligned} ((a * b) * c)[t] &= \sum_i (a * b)[i].c[t-i] \\ &= \sum_i \sum_j a[j].b[i-j].c[t-i] \qquad (1) \end{aligned}$$

And:

$$\begin{aligned}(a * (b * c))[t] &= \sum_x a[x].(b * c)[t - x] \\ &= \sum_x a[x]. \sum_y b[y].c[t - x - y] \\ &= \sum_x \sum_y a[x].b[y].c[t - x - y] \qquad (2)\end{aligned}$$

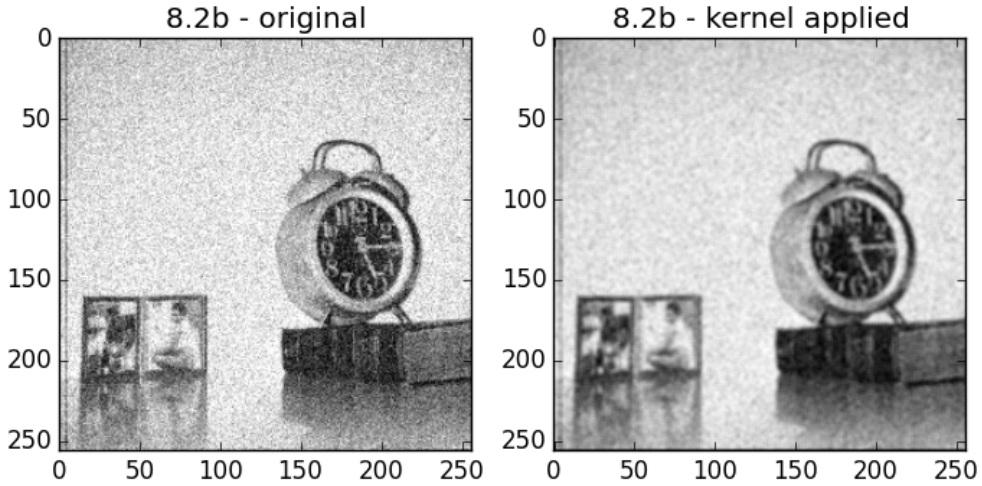
Because $y$ is only dependent on $x$, we can set $x = j$ and $y = i - j$. Now we have:

$$\begin{aligned}a[x].b[y].c[t - x - y] &= a[j].b[i - j].c[t - j - (i - j)] \\ &= a[j].b[i - j].c[t - i] \qquad (3)\end{aligned}$$

From (1), (2) and (3) $\rightarrow$ **Proved**

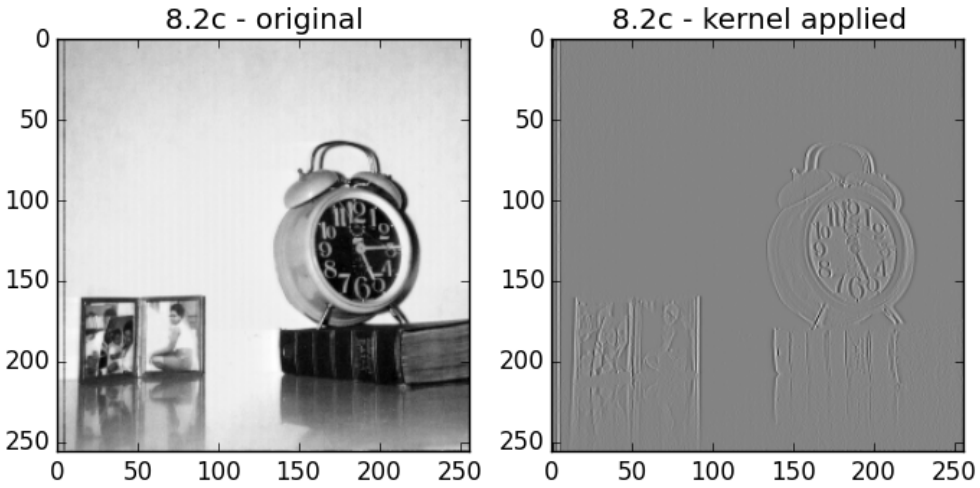
# Exercise 8.2

b) See **82.py**
The kernel will average each image point with 8 surrounding, the effect is the image will be blurred. Image will be denoised




when the kernel is applied.
c) See **82.py**
Applying this kernel, each point would be the difference between it and its 2 horizontal neighbor points . One application of this kernel is edge detection.

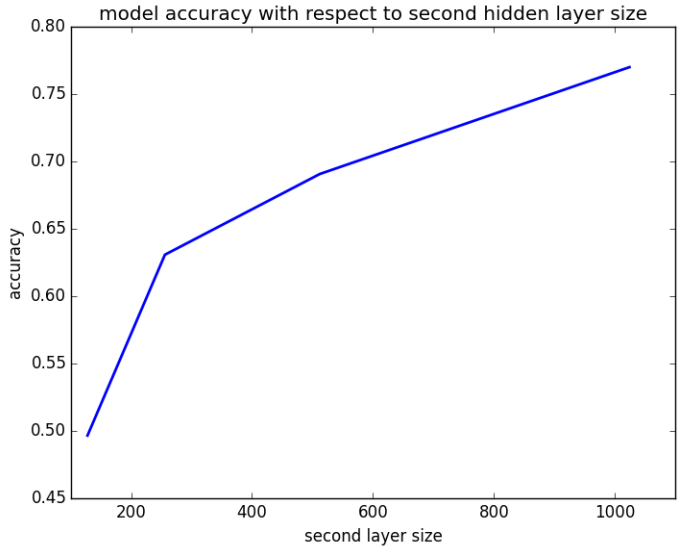
Vinh Thinh Ho (2562630)
Noshaba Cheema (2562653)




d)
The padding zero is not always optimal. For example, if the boundary have is dark (high gray value), one of it neighbor is also almost dark (because 2 neighbor points often have close values), the other neighbor is the padding would have value 0. After applying kernel, this point would have very negative value (always become light after rescaling), so it is less informative. A better padding is we can pad each point with a random point inside the image, or we could pad each point with the average value of points inside the image.

# Exercise 8.3

See **83.py**
We trained using half of MNIST data, with only 1 epoch, using $batch\_size = 50$, and we vary the size of second hidden layer: 128, 256, 512, 1024. Below is the result:




We can see that when increasing the size of the second hidden layer, the accuracy of the model also increases.

# Exercise 8.4