Supplementary Material

1 Supplementary to Sec. 3: Deformable Latent Space

1.1 Network Architecture

Notation:

Convolution layer is written as `conv(out, k=3, s=1, p=1, d=1)`, where `out` means number of output channels, `k` means filter size (default 3), `s` means stride (default 1), `p` means padding (default 1), `d` means dilation (default 1), with no bias term.

Transposed convolution is written as `convt(out, k=3, s=1, p=1, d=1)`, with the same parameters as in `conv`, with no bias term.

Dynamic convolution is written as `dconv(w, out, k=3, s=1, p=1, d=1)`, with an input parameter `w`, and the rest of the parameters are the same as in `conv`, with no bias term.

Encoder f:

```
conv(64, 4, 2, 1) -- LeakyReLU(0.2)
  -- conv(128, 4, 2, 1) -- BatchNorm -- LeakyReLU(0.2)
  -- conv(256, 4, 2, 1) -- BatchNorm -- LeakyReLU(0.2)
  -- conv(256, 4, 2, 1) -- BatchNorm -- LeakyReLU(0.2);
```

MLP transformer of defocal distance d:

```
d -> MatrixMultiply(OutputDim=8) -- ReLU
  -- MatrixMultiply(OutputDim=8) -- L2Normalize -> w;
```

Deformer $D(\mathbf{h}) = \mathbf{h} +$

```
dconv(w0) -- BatchNorm -- dconv(w0)
  -- (BatchNorm -- ReLU -- dconv(w1)
  -- BatchNorm -- ReLU -- deconv(w1))
  -- (BatchNorm -- ReLU -- dconv(w2, p=2, s=2)
  -- BatchNorm -- ReLU -- dconv(w2)
  -- BatchNorm -- ReLU -- dconv(w2))
  -- (BatchNorm -- ReLU -- dconv(w3, p=2, s=2)
  -- BatchNorm -- ReLU -- dconv(w3)
  -- BatchNorm -- ReLU -- dconv(w3))
  -- tanh;
```

where `(...)` stands for a residual block, and `w0`, `w1`, `w2` and `w3` are computed from separate MLPs.

Decoder g:

```
convt(256, 4, 2, 1) -- BatchNorm -- ReLU
  -- convt(128, 4, 2, 1) -- BatchNorm -- ReLU
  -- convt(64, 4, 2, 1) -- BatchNorm -- ReLU
  -- convt(1, 4, 2, 1) -- ReLU;
```

All learnable weights are initialized with Kaiming initialization when training starts.

2 Supplementary to Sec. 4: Experiments and Evaluations

2.1 Kernel Formulation

The kernels used in the experiments are simulated zone plates multiplied by a band-limited filter. The band-limited filter can be written as:

$$M(r) = \begin{cases} 1 & r < \alpha R, \\ 0.54 + 0.46\cos(\pi\frac{r-\alpha R}{(\beta-\alpha)R}) & \alpha R \leq r < \beta R, \\ 0.08 & r \geq \beta R, \end{cases} \tag{1}$$

where r is the distance from a pixel to the center, $R = 64, \alpha = 0.3, \beta = 0.35$.

Zone plate probes are generated from simulations. Python source code is provided below:

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.special as ss

def dist(n):
    a = np.arange(n)
    a = np.where(a<np.float(n)/2.,a,np.abs(a-np.float(n)
     ↪  ))**2
    array=np.zeros((n,n))
    for i in range(np.int(n/2)+1):
        y=np.sqrt(a+i**2)
        array[:,i]=y
        if i!=0:
            array[:,n-i]=y
    return np.fft.fftshift(array)

def cal_zp(diameter_m,finest_zone_width_m,energy_kev,def
 ↪  ocal_distance_m,pixel_size_m,array_size,disp_flag=Fa
 ↪  lse):
```

```
wavelength_m = (12.4 / energy_kev) * 1.e-10
focal_length_m = diameter_m * finest_zone_width_m /
 ↪  wavelength_m

n = 1024
dr_m = diameter_m / 2 / n

z_defocal_m = focal_length_m + defocal_distance_m

array_size_new = np.int(array_size * 1.6)
line_m = (np.arange(array_size_new) -
 ↪  array_size_new/2) * pixel_size_m
k = 2 * np.pi / wavelength_m

line = np.zeros(array_size_new).astype(complex)

for ii in range(array_size_new):
    s = 0.
    for jj in range(n):
        r = jj * dr_m
        s += np.exp(1j * 0.5 * k * (1/z_defocal_m -
         ↪  1/focal_length_m) * r**2) * \
          ss.j0(k * line_m[ii] * r / z_defocal_m) *
           ↪  r * dr_m
    line[ii] = s * np.exp(1j * 0.5 * k *
     ↪  line_m[ii]**2 / z_defocal_m)

if disp_flag:
    plt.close('all')
    plt.figure()
    plt.subplot(221)
    plt.plot(np.abs(line))
    plt.title('Diameter:
     ↪  '+np.str(diameter_m/1e-6)+'um')
    plt.subplot(222)
    plt.plot(np.angle(line))
    plt.title('Outmost zone:
     ↪  '+np.str(finest_zone_width_m/1e-9)+'nm')

zp_array =
 ↪  np.zeros((array_size,array_size)).astype(complex)
dummy = dist(array_size)
n_max = np.int(np.max(dummy))
for i in range(n_max):
    tmp = line[np.int(array_size_new/2)+i]
```

```
        r_in = i
        r_out = i + 1
        index = np.where((dummy >= r_in) & (dummy <
         ↪  r_out))
        zp_array[index] = tmp

    if disp_flag:
        plt.subplot(223)
        plt.imshow(np.abs(zp_array))
        plt.title('Energy: '+np.str(energy_kev)+'keV')
        plt.subplot(224)
        plt.imshow(np.angle(zp_array))
        plt.title('Pixel size:
         ↪  '+np.str(pixel_size_m/1e-9)+'nm')
        plt.show()

    zp_array /= np.max(np.abs(zp_array))
    return zp_array
```

2.2 Additional Results on Natural Images

Deblurring results of hyper-Laplacian (HL), MLP, multi-input fully convolutional network (MFCN), generalized low-rank approximation (GLRA), and our method on natural image data are shown in figures below. From left to right, the images are: original, 0.1μm, 3μm, 5μm, 7μm, 10μm and 15μm, respectively.

2.3 Additional Results on Fluorescence Images

Deblurring results of HL, MLP, MFCN, GLRA and our method on fluorescence data are shown in figures below.

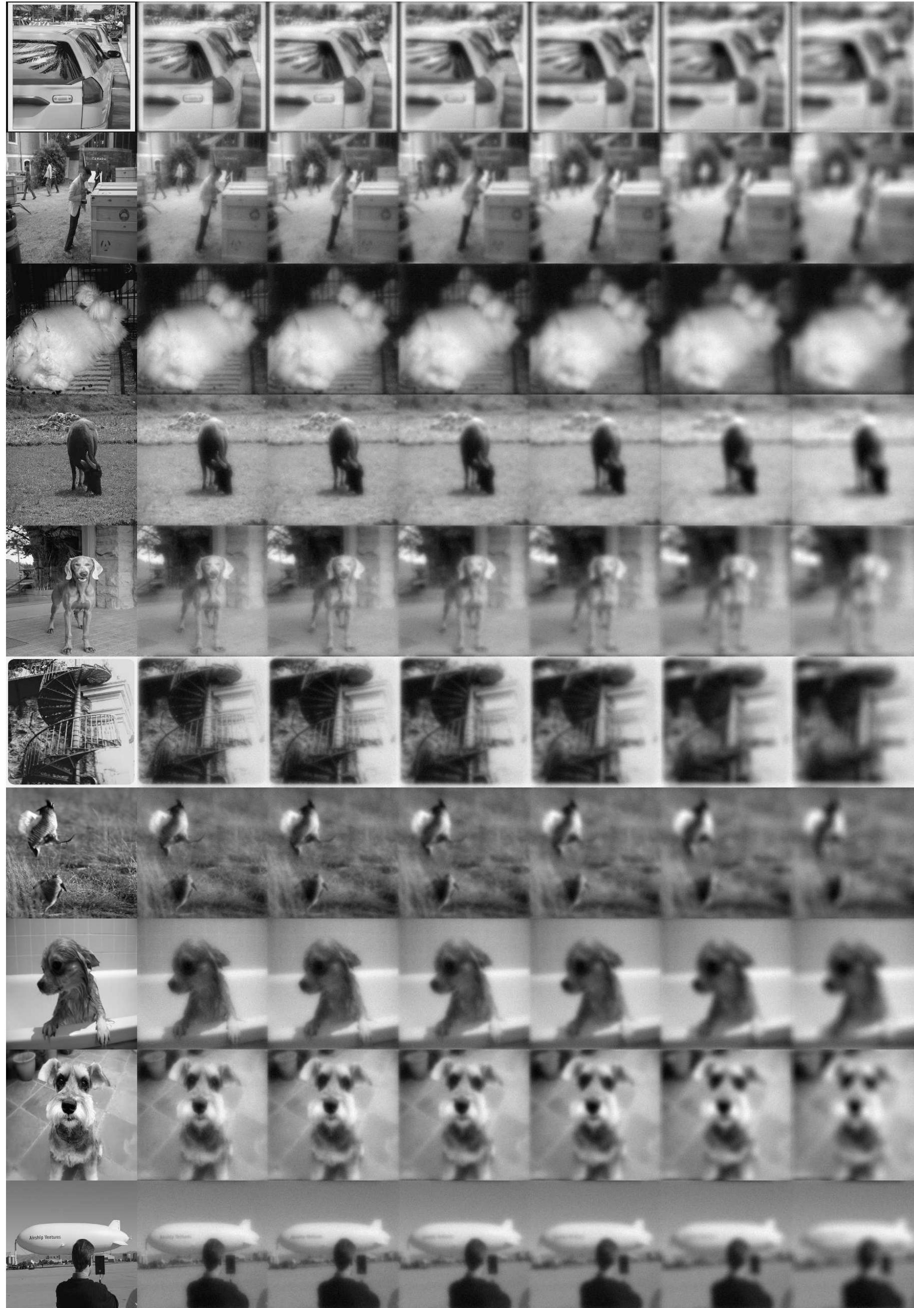

Figure 1: Blurred images with different kernels

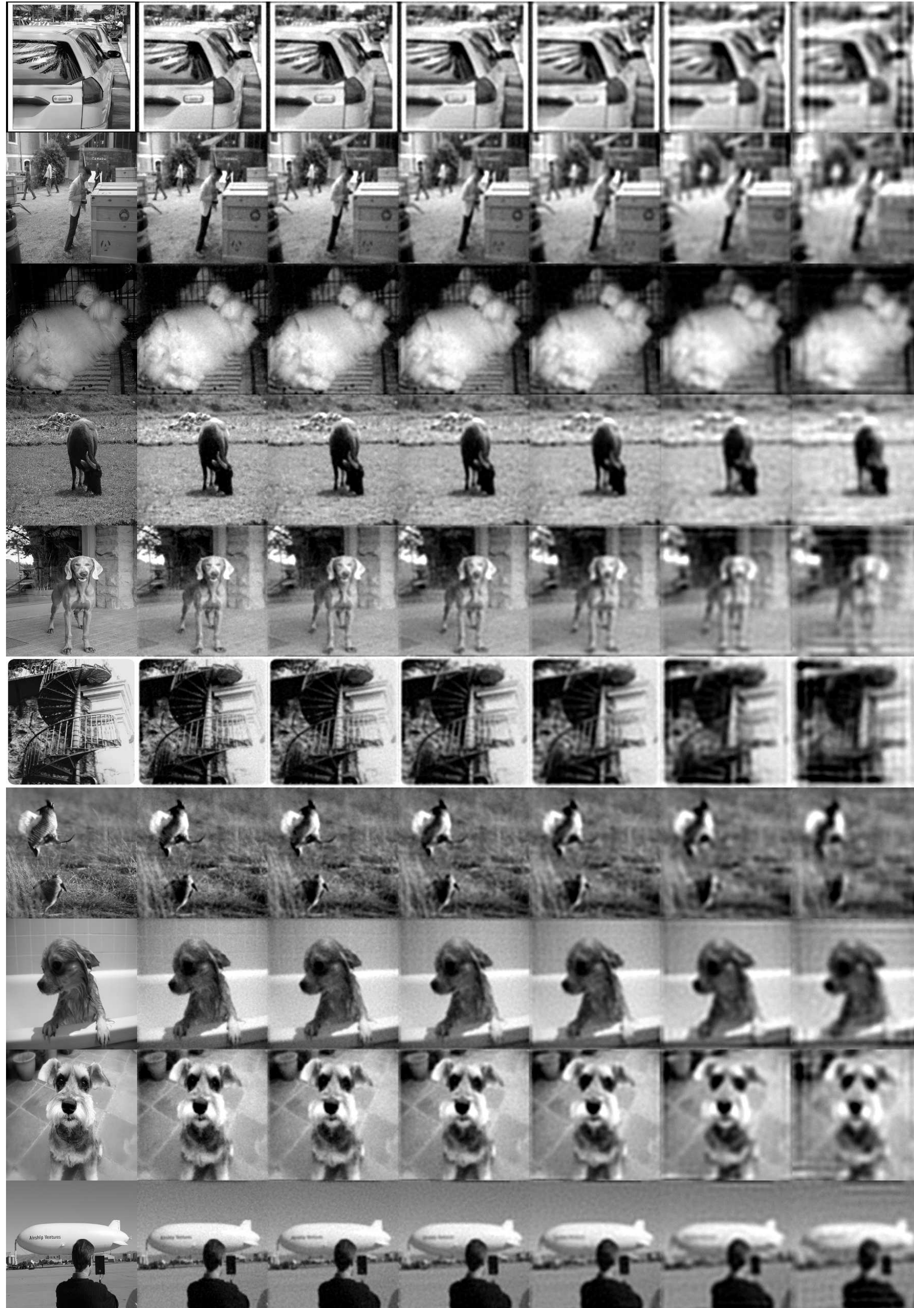

Figure 2: HL

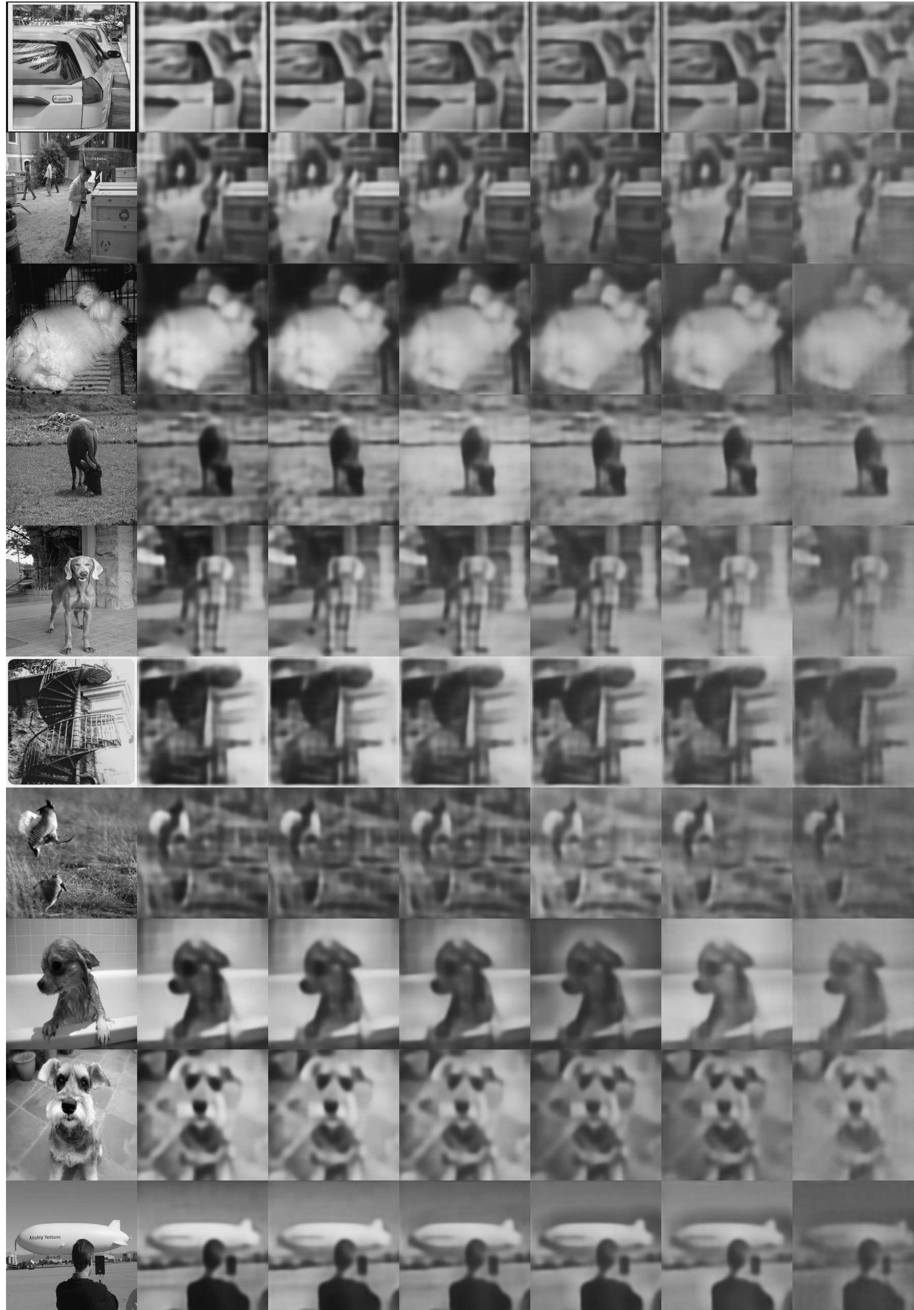

Figure 3: MLP

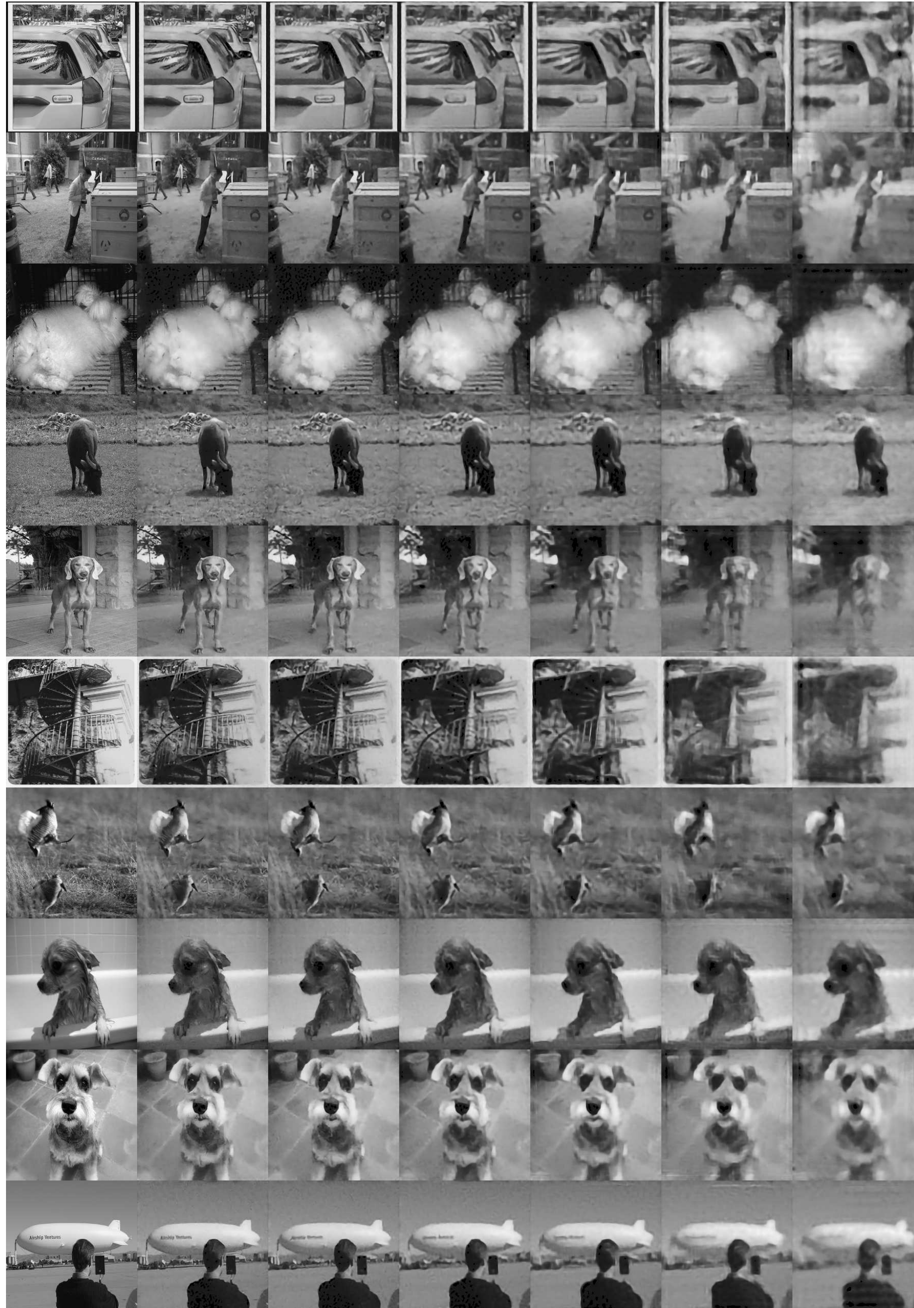

Figure 4: MFCN

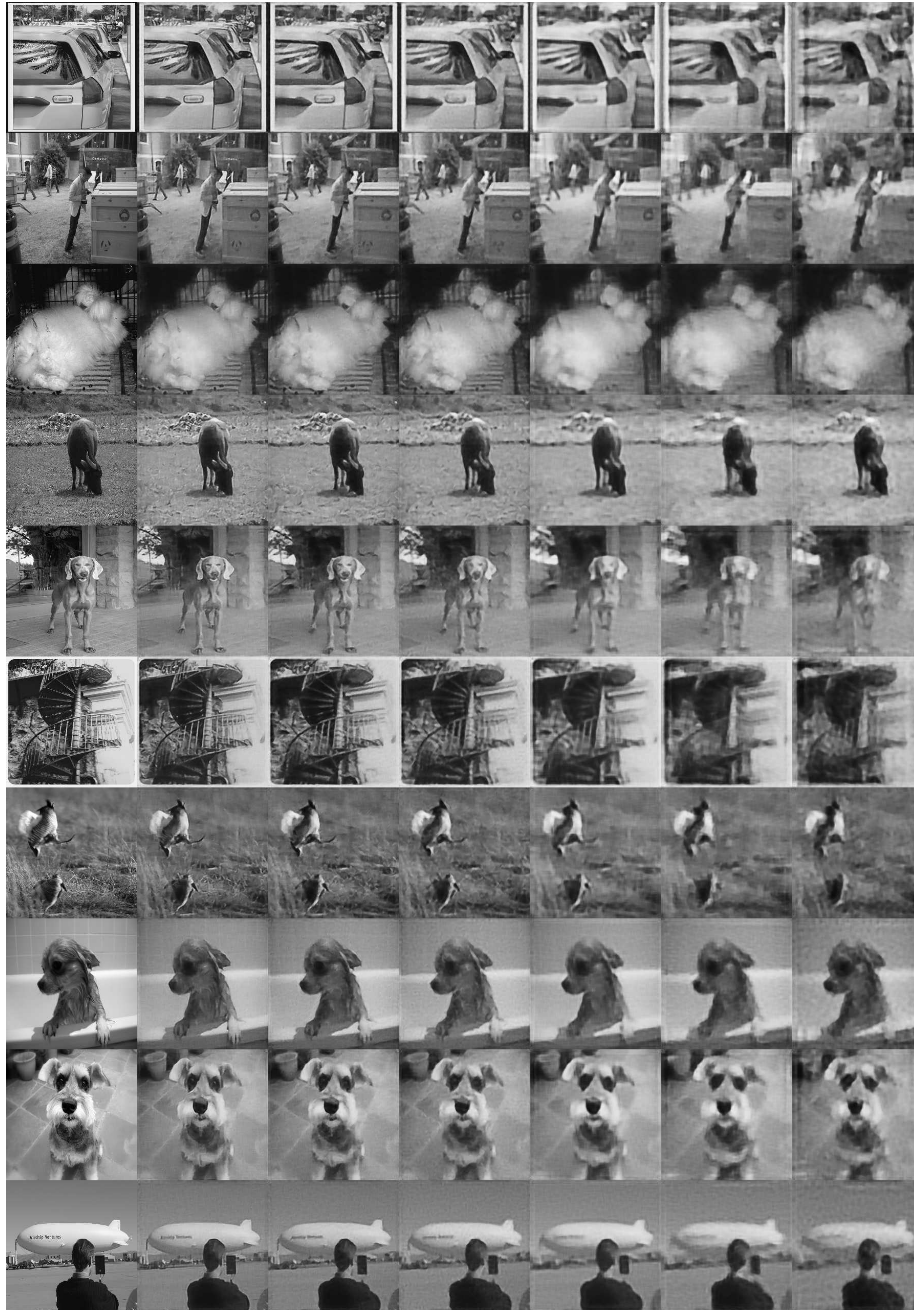

Figure 5: GLRA

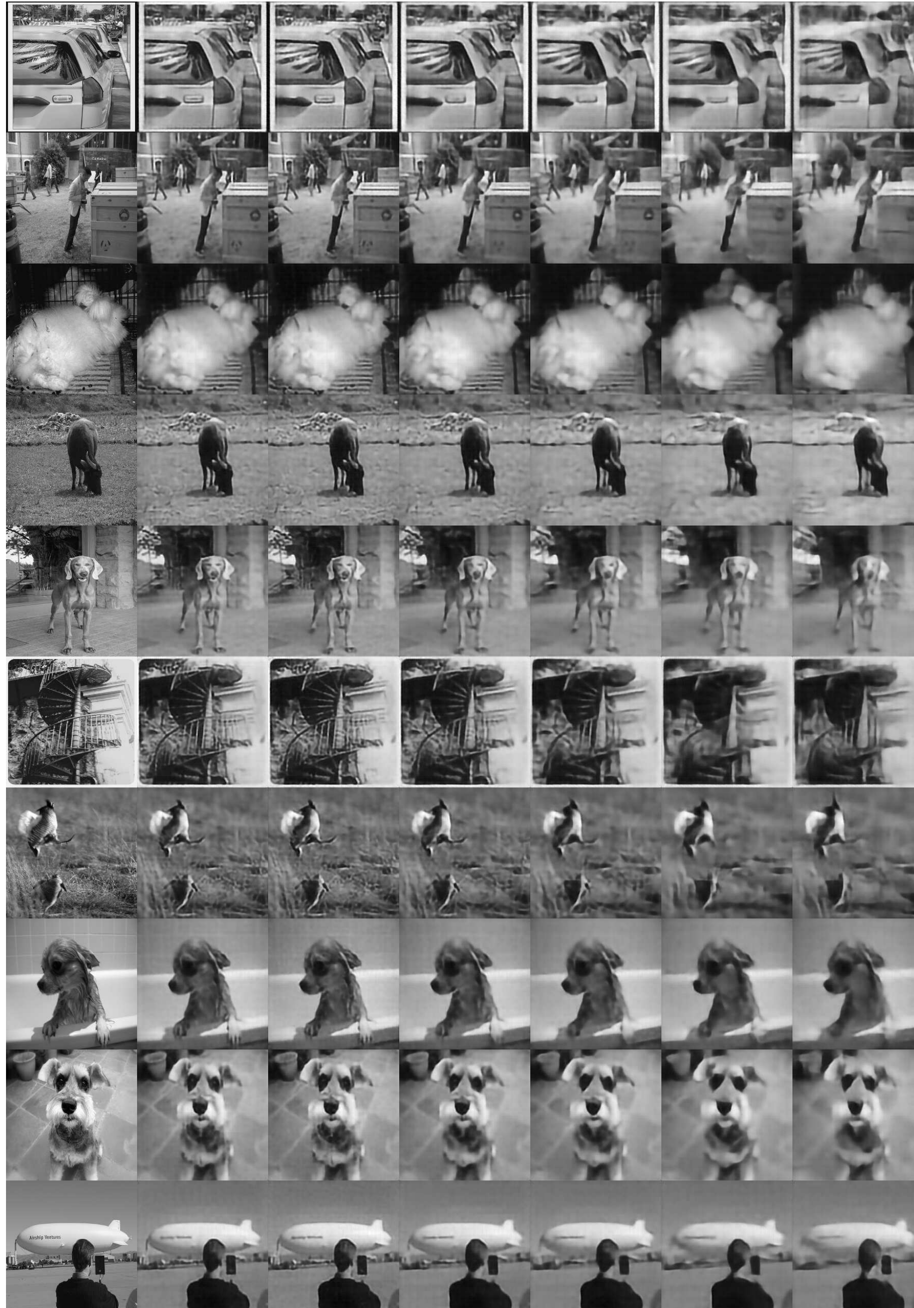

Figure 6: Ours

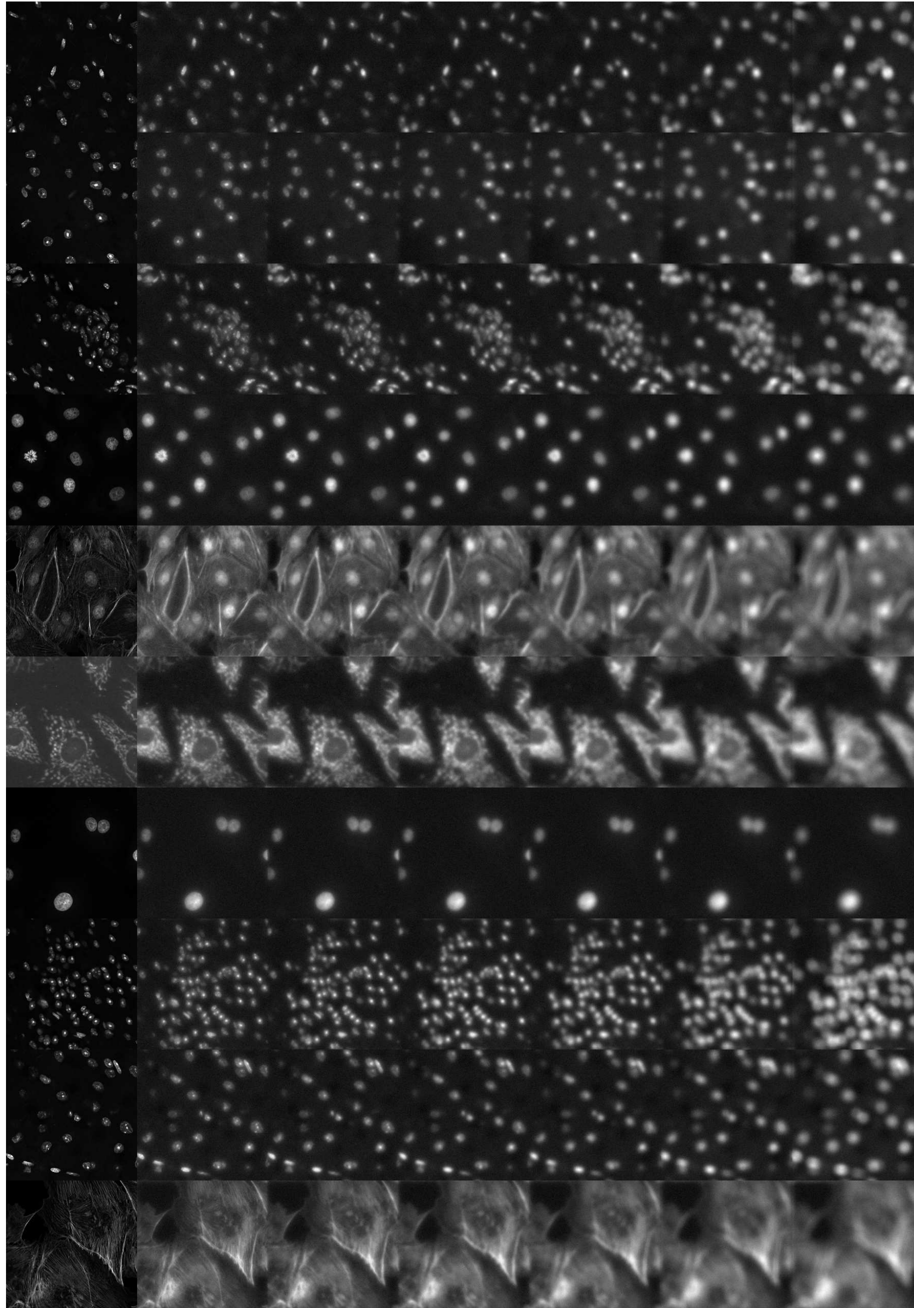

Figure 7: Blurred images with different kernels

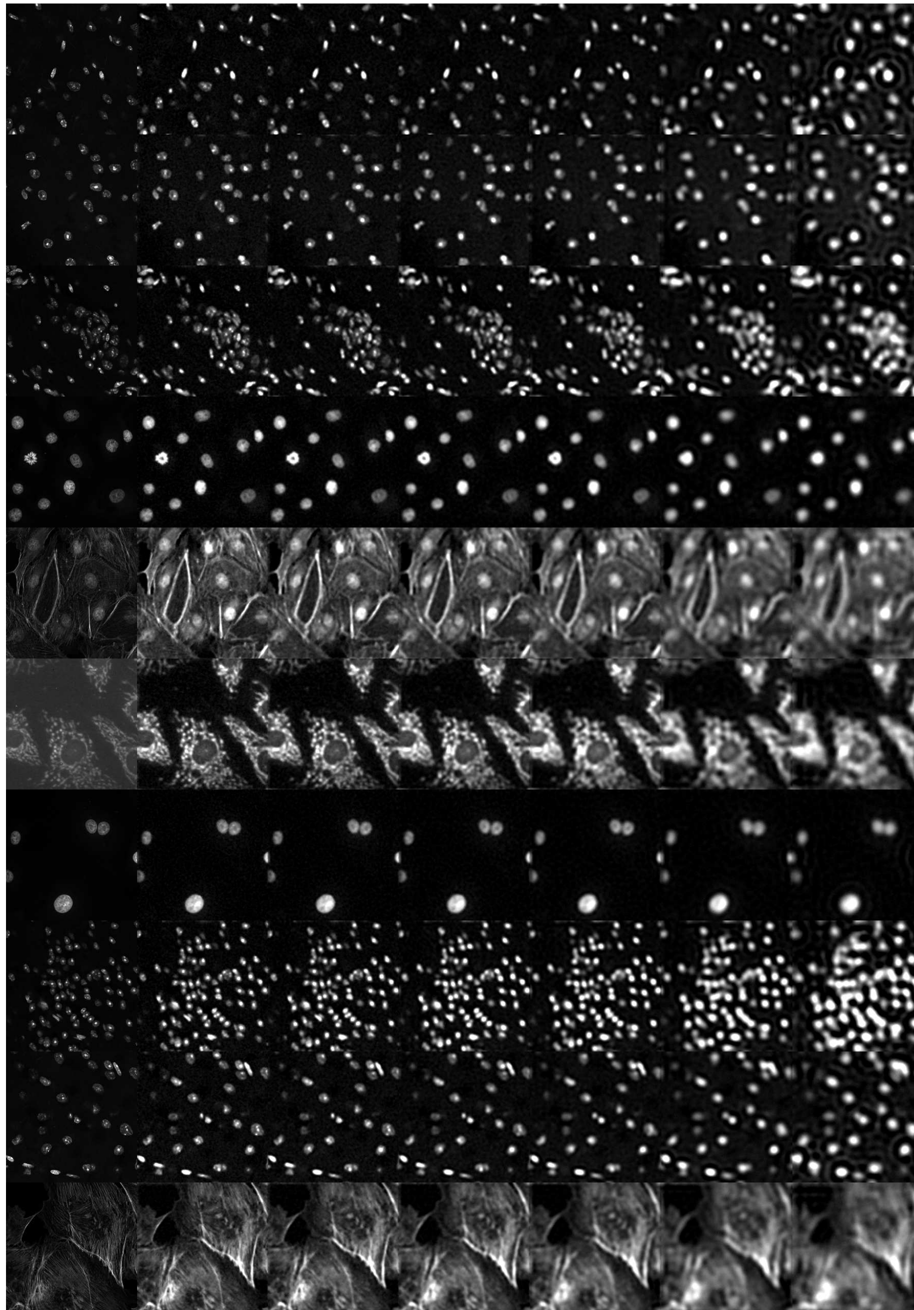

Figure 8: HL

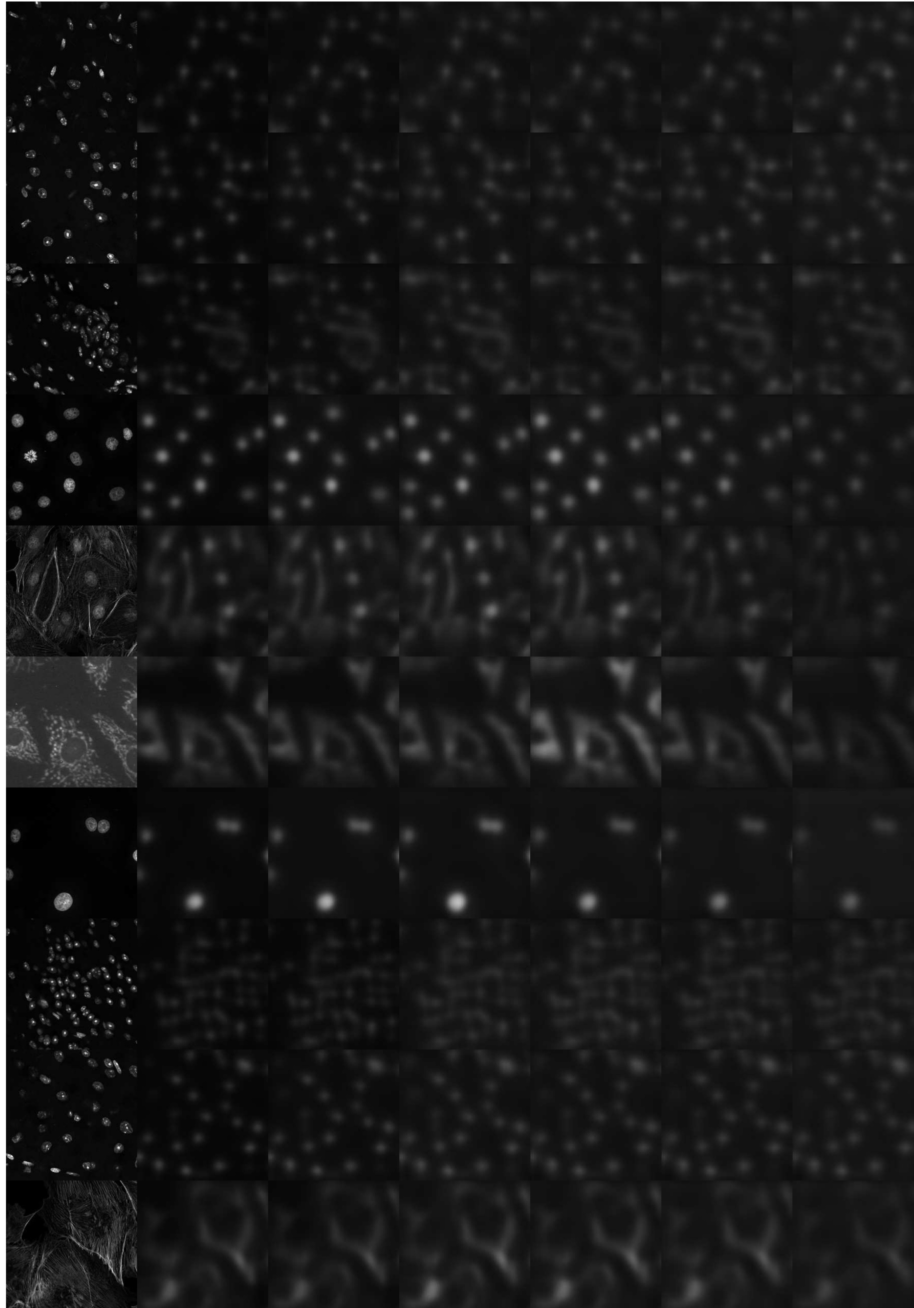

Figure 9: MLP

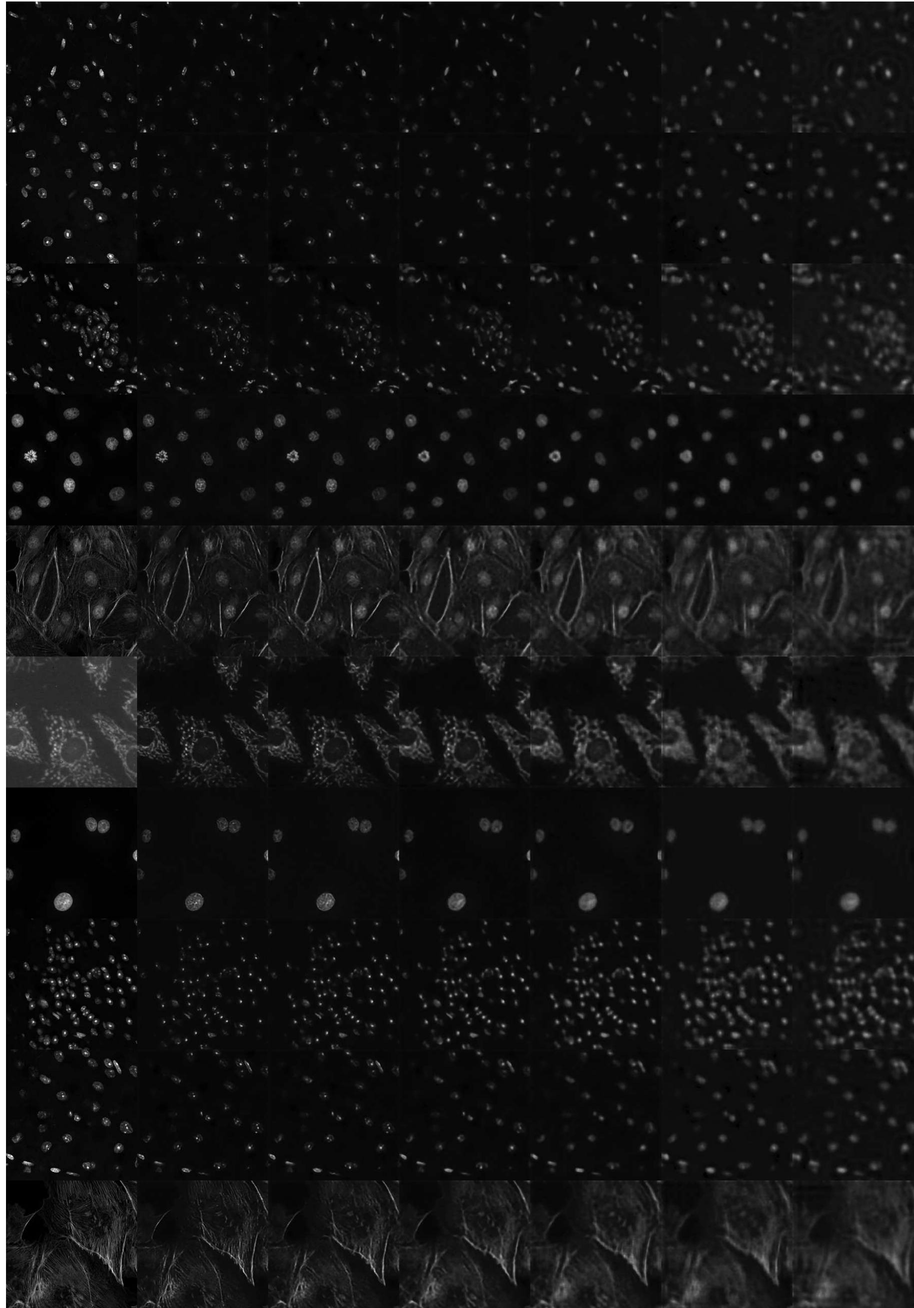

Figure 10: MFCN

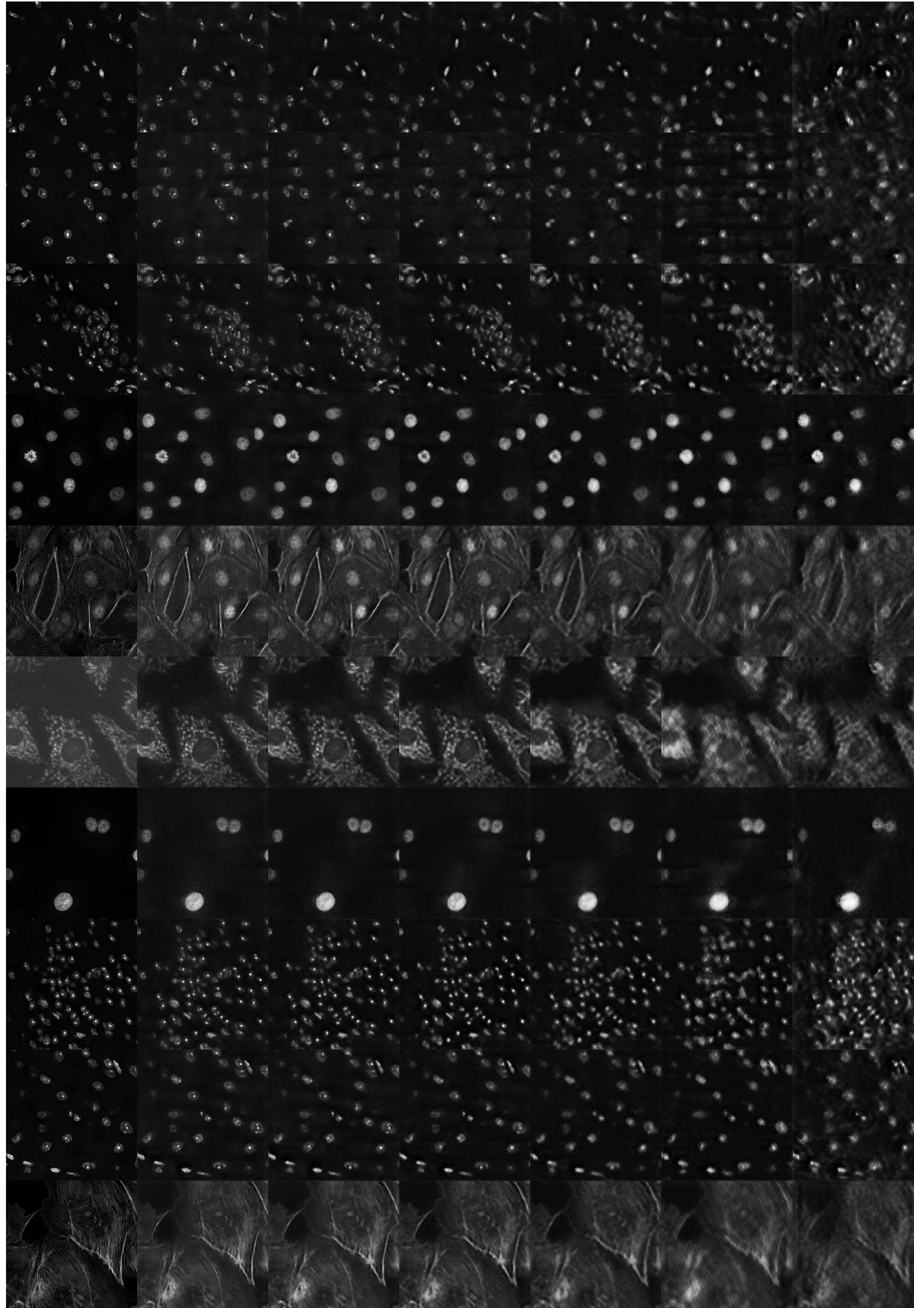

Figure 11: GLRA

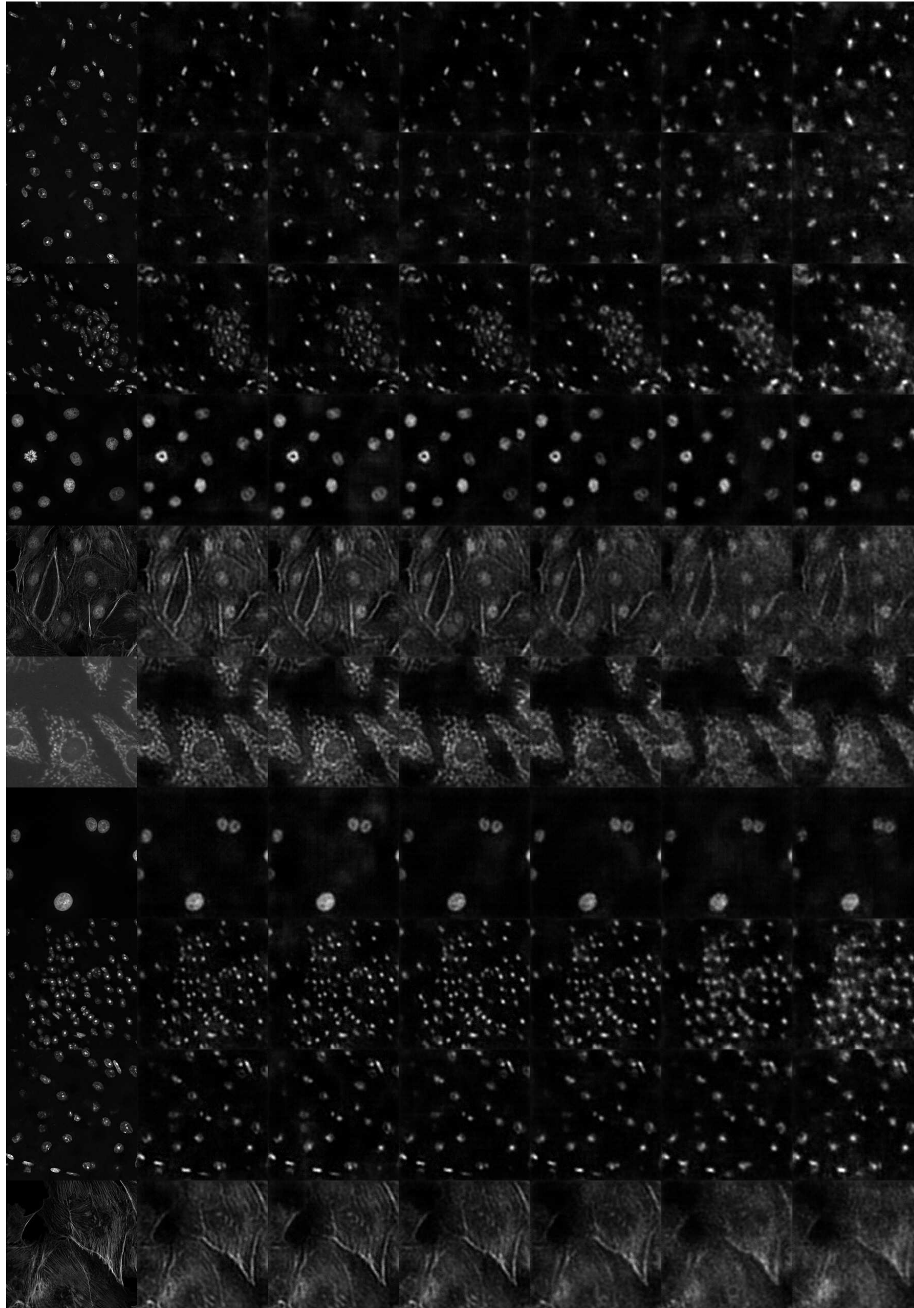

Figure 12: Ours