## Denoising an image using Classic restoration API

One can use the following lines to denoise a single image with default options using our Object-Oriented denoising API.

```
from aydin.restoration.denoise.classic import Classic

classic_restoration = Classic()
classic_restoration.train(noisy_image)
denoised_image = classic_restoration.denoise(noisy_image)
```

It is also easy to pass specific transforms to use before and/or after denoising. One can do the the following:

```
from aydin.restoration.denoise.classic import Classic

transforms = [
     {"class": RangeTransform, "kwargs": {}},
     {"class": PaddingTransform, "kwargs": {}},
 ]
classic_restoration = Classic(it_transforms=transforms)
classic_restoration.train(noisy_image)
denoised_image = classic_restoration.denoise(noisy_image)
```

One can also use the following lines to denoise a single image with default options using our procedural denoising endpoint.

```
from aydin.restoration.denoise.classic import classic_denoise

denoised_image = classic_denoise(noisy_image)
```

## Deconvolving an image using aydin API

One can use the following lines to deconvolve a single image with default options using our Object-Oriented denoising API.

```
from aydin.restoration.deconvolve.lr import LucyRichardson

lr = LucyRichardson(
     psf_kernel=psf_kernel, max_num_iterations=20, backend='scipy-cupy'
 )

lr.train(noisy_and_blurred_image, noisy_and_blurred_image)

lr_deconvolved_image = lr.deconvolve(noisy_and_blurred_image)
```

One can also use the following lines to deconvolve a single image with default options using our procedural deconvolving endpoint.

```
from aydin.restoration.deconvolve.lr import lucyrichardson

lr_deconvolved_image = lucyrichardson(noisy_and_blurred_image)
```

## Denoising an image using Noise2SelfCNN restoration API

One can use the following lines to denoise a single image with default options using our Object-Oriented denoising API.

```
from aydin.restoration.denoise.noise2selfcnn import Noise2SelfCNN

n2s = Noise2SelfCNN()
n2s.train(noisy_image)
denoised_image = n2s.denoise(noisy_image)
```

It is also easy to pass specific transforms to use before and/or after denoising. One can do the the following:

```
from aydin.restoration.denoise.noise2selfcnn import Noise2SelfCNN

transforms = [
     {"class": RangeTransform, "kwargs": {}},
     {"class": PaddingTransform, "kwargs": {}},
 ]
n2s = Noise2SelfCNN(it_transforms=transforms)
n2s.train(noisy_image)
denoised_image = n2s.denoise(noisy_image)
```

One can also use the following lines to denoise a single image with default options using our procedural denoising endpoint.

```
from aydin.restoration.denoise.noise2selfcnn import noise2self_cnn

denoised_image = noise2self_cnn(noisy_image)
```

## Denoising an image using Noise2SelfFGR restoration API

One can use the following lines to denoise a single image with default options using our Object-Oriented denoising API.

```
from aydin.restoration.denoise.noise2selffgr import Noise2SelfFGR

n2s = Noise2SelfFGR()
```

```
n2s.train(noisy_image)
denoised_image = n2s.denoise(noisy_image)
```

It is also easy to pass specific transforms to use before and/or after denoising. One can do the the following:

```
from aydin.restoration.denoise.noise2selffgr import Noise2SelfFGR

transforms = [
     {"class": RangeTransform, "kwargs": {}},
     {"class": PaddingTransform, "kwargs": {}},
 ]
n2s = Noise2SelfFGR(it_transforms=transforms)
n2s.train(noisy_image)
denoised_image = n2s.denoise(noisy_image)
```

One can also use the following lines to denoise a single image with default options using our procedural denoising endpoint.

```
from aydin.restoration.denoise.noise2selffgr import noise2self_fgr

denoised_image = noise2self_fgr(noisy_image)
```

## Denoising an image using ImageDenoiserClassic

One can use the following lines to denoise a single image with default options using our Object-Oriented denoising API.

```
from aydin.it.classic import ImageDenoiserClassic

it = ImageDenoiserClassic(method='lowpass')

it.train(noisy, noisy)
denoised = it.translate(noisy)
```

It is also easy to pass specific transforms to use before and/or after denoising. One can do the the following:

```
from aydin.it.classic import ImageDenoiserClassic

it = ImageDenoiserClassic(method='lowpass')
it.add_transform(RangeTransform())
it.add_transform(PaddingTransform())

it.train(noisy, noisy)
denoised = it.translate(noisy)
```

## Denoising an image using ImageTranslatorCNN

One can use the following lines to denoise a single image with default options using our Object-Oriented denoising API.

```
from aydin.it.cnn import ImageTranslatorCNN

it = ImageTranslatorCNN()

it.train(noisy, noisy)
denoised = it.translate(noisy)
```

It is also easy to pass specific transforms to use before and/or after denoising. One can do the the following:

```
from aydin.it.cnn import ImageTranslatorCNN

it = ImageTranslatorCNN()
it.add_transform(RangeTransform())
it.add_transform(PaddingTransform())

it.train(noisy, noisy)
denoised = it.translate(noisy)
```

## Denoising an image using ImageTranslatorFGR

One can use the following lines to denoise a single image with default options using our Object-Oriented denoising API.

```
from aydin.it.fgr import ImageTranslatorFGR

it = ImageTranslatorFGR()

it.train(noisy, noisy)
denoised = it.translate(noisy)
```

It is also easy to pass specific transforms to use before and/or after denoising. One can do the the following:

```
from aydin.it.fgr import ImageTranslatorFGR

it = ImageTranslatorFGR()
it.add_transform(RangeTransform())
it.add_transform(PaddingTransform())

it.train(noisy, noisy)
denoised = it.translate(noisy)
```

## How to implement Noise2Noise using ImageTranslatorFGR

It is quite easy to train Noise2Noise model with image translators provided in Aydin API. You can see an example below using ImageTranslatorFGR:

```
from aydin.it.fgr import ImageTranslatorFGR

it = ImageTranslatorFGR()
it.add_transform(RangeTransform())
it.add_transform(PaddingTransform())

for noisy_image1, noisy_image2 in noisy_pairs:
    it.train(noisy_image1, noisy_image2)

denoised = it.translate(noisy_image)
```

Noise2Noise is a great method to train a model from pairs of images sharing the same information signal with different noise instances. You can find more information about how to prepare noisy image pairs on the original paper. More information about the paper is available here and the code that is published with paper can be found on github.

## How to implement supervised denoising using image translators

It is quite easy to run supervised denoising with image translators provided in Aydin API. You can see a quick example below using ImageTranslatorFGR:

```
from aydin.it.fgr import ImageTranslatorFGR

it = ImageTranslatorFGR()
it.add_transform(RangeTransform())
it.add_transform(PaddingTransform())

it.train(noisy, groundtruth)
denoised = it.translate(noisy)
```

Similar to ImageTranslatorFGR implementation, same can be achieved with ImageTranslatorCNN as shown below:

```
from aydin.it.cnn import ImageTranslatorCNN

it = ImageTranslatorCNN()
it.add_transform(RangeTransform())
it.add_transform(PaddingTransform())
```

```
it.train(noisy, groundtruth)
denoised = it.translate(noisy)
```

## Apply Attenuation transform

You can use the following lines to apply attenuation transform on a single image using our transforms API.

```
from aydin.it.transforms.attenuation import AttenuationTransform

attenuation_transform = AttenuationTransform(axes=0)

preprocessed = attenuation_transform.preprocess(image)
postprocessed = attenuation_transform.postprocess(preprocessed)
```

## Apply Deskew transform

You can use the following lines to apply deskew transform on a single image using our transforms API.

```
from aydin.it.transforms.deskew import DeskewTransform

 deskew_transform = DeskewTransform(delta=-3, z_axis=0, skew_axis=1)

 deskewed_image = deskew_transform.preprocess(image)
 skewed_image = deskew_transform.postprocess(deskewed_image)
```

## Apply fixed pattern transform

You can use the following lines to apply fixed pattern transform on a single image using our transforms API.

```
from aydin.it.transforms.fixedpattern import FixedPatternTransform

fixedpattern_transform = FixedPatternTransform(axes=[1, 2])

preprocessed = fixedpattern_transform.preprocess(image)
```

## Apply high pass transform

You can use the following lines to apply high pass transform on a single image using our transforms API.

```
from aydin.it.transforms.highpass import HighpassTransform
```

```
highpass_transform = HighpassTransform()

preprocessed = highpass_transform.preprocess(image)
postprocessed = highpass_transform.postprocess(preprocessed)
```

## Apply histogram equalisation transform

You can use the following lines to apply histogram equalisation transform on a single image using our transforms API.

```
from aydin.it.transforms.histogram import HistogramEqualisationTransform

histogram_transform = HistogramEqualisationTransform()

preprocessed = histogram_transform.preprocess(image)
postprocessed = histogram_transform.postprocess(preprocessed)
```

## Apply motion stabilisation transform

You can use the following lines to apply motion stabilisation transform on a single image using our transforms API.

```
from aydin.it.transforms.motion import MotionStabilisationTransform

motion_transform = MotionStabilisationTransform(axes=0)

preprocessed_image = motion_transform.preprocess(image.copy())
postprocessed_image = motion_transform.postprocess(preprocessed_image.copy())
```

## Apply padding transform

You can use the following lines to apply padding transform on a single image using our transforms API.

```
from aydin.it.transforms.padding import PaddingTransform

padding_transform = PaddingTransform(pad_width=17)

preprocessed = padding_transform.preprocess(image)
postprocessed = padding_transform.postprocess(preprocessed)
```

## Apply periodic noise suppression transform

You can use the following lines to apply periodic noise suppression transform on a single image using our transforms API.

```
from aydin.it.transforms.periodic import PeriodicNoiseSuppressionTransform

periodic_noise_suppression_transform = PeriodicNoiseSuppressionTransform()

preprocessed = periodic_noise_suppression_transform.preprocess(image)
postprocessed = periodic_noise_suppression_transform.postprocess(preprocessed)
```

## Apply range transform

You can use the following lines to apply range transform on a single image using our transforms API.

```
from aydin.it.transforms.range import RangeTransform

range_transform = RangeTransform(mode="minmax")

preprocessed = range_transform.preprocess(image)
postprocessed = range_transform.postprocess(preprocessed)
```

## Apply salt-pepper transform

You can use the following lines to apply salt-pepper transform on a single image using our transforms API.

```
from aydin.it.transforms.salt_pepper import SaltPepperTransform

salt_pepper_transform = SaltPepperTransform()

corrected = salt_pepper_transform.preprocess(image)
```

## Apply variance stabilisation transform

You can use the following lines to apply variance stabilisation transform on a single image using our transforms API.

```
from aydin.it.transforms.variance_stabilisation import VarianceStabilisationTransform

vst = VarianceStabilisationTransform(mode="anscomb")

preprocessed = vst.preprocess(image)
postprocessed = vst.postprocess(preprocessed)
```