EE-559 – Deep learning

7.3. Denoising autoencoders

François Fleuret

https://fleuret.org/ee559/

Jun 15, 2020
Beside dimension reduction, autoencoders can capture dependencies between signal components to restore a degraded inputs.

In that case, we can ignore the encoder/decoder structure, and such a model

$$\phi : \mathcal{X} \rightarrow \mathcal{X}.$$ 

is referred to as a **denoising** autoencoder.
Beside dimension reduction, autoencoders can capture dependencies between signal components to restore a degraded inputs.

In that case, we can ignore the encoder/decoder structure, and such a model

\[ \phi : \mathcal{X} \rightarrow \mathcal{X}. \]

is referred to as a **denoising** autoencoder.

The goal is not anymore to optimize \( \phi \) so that

\[ \phi(X) \approx X \]

but, given a perturbation \( \tilde{X} \) of the signal \( X \), to restore the signal, hence

\[ \phi(\tilde{X}) \approx X. \]
We can illustrate this notion in 2d with an additive Gaussian noise, and the quadratic loss, hence

\[
\hat{w} = \operatorname{argmin}_w \frac{1}{N} \sum_{n=1}^{N} \| x_n - \phi(x_n + \epsilon_n; w) \|^2,
\]

where \( x_n \) are the data samples, and \( \epsilon_n \) are Gaussian random noise vectors.
model = nn.Sequential(
    nn.Linear(2, 100),
    nn.ReLU(),
    nn.Linear(100, 2)
)

batch_size, nb_epochs = 100, 1000
optimizer = torch.optim.Adam(model.parameters(), lr = 1e-3)
criterion = nn.MSELoss()

for e in range(nb_epochs):
    for input in data.split(batch_size):
        noise = input.new(input.size()).normal_(0, 0.1)
        output = model(input + noise)
        loss = criterion(output, input)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
We can do the same on MNIST, for which we keep our deep autoencoder, and ignore its encoder/decoder structure.

```python
corrupted_input = corruptor.corrupt(input)

output = model(corrupted_input)
loss = 0.5 * (output - input).pow(2).sum() / input.size(0)

optimizer.zero_grad()
loss.backward()
optimizer.step()
```
We can do the same on MNIST, for which we keep our deep autoencoder, and ignore its encoder/decoder structure.

```
corrupted_input = corruptor.corrupt(input)
output = model(corrupted_input)
loss = 0.5 * (output - input).pow(2).sum() / input.size(0)

optimizer.zero_grad()
loss.backward()
optimizer.step()
```

We consider three types of corruptions, that go beyond additive noise:

<table>
<thead>
<tr>
<th>Original</th>
<th>Pixel erasure</th>
<th>Blurring</th>
<th>Block masking</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Original Image]</td>
<td>![Pixel Erasure]</td>
<td>![Blurring]</td>
<td>![Block Masking]</td>
</tr>
</tbody>
</table>
Original

Corrupted ($p = 0.9$)

Reconstructed
Original

\[
\begin{array}{cccccccccc}
7 & 2 & 1 & 0 & 4 & 1 & 4 & 9 & 5 & 9 \\
9 & 0 & 1 & 5 & 9 & 7 & 8 & 4 & 9 & 6 & 6 & 5 \\
4 & 0 & 7 & 4 & 0 & 1 & 3 & 1 & 3 & 4 & 7 & 2 \\
\end{array}
\]

Corrupted ($\sigma = 2$)

\[
\begin{array}{cccccccccc}
7 & 2 & 1 & 0 & 4 & 1 & 4 & 9 & 5 & 9 \\
9 & 0 & 1 & 5 & 9 & 7 & 8 & 4 & 9 & 6 & 6 & 5 \\
4 & 0 & 7 & 4 & 0 & 1 & 3 & 1 & 3 & 4 & 7 & 2 \\
\end{array}
\]

Reconstructed
Original

Corrupted ($\sigma = 4$)

Reconstructed
Original

Corrupted (16 × 16)

Reconstructed
A key weakness of this type of denoising is that the posterior

\[ \mu_{X|\tilde{X}} \]

may be non-deterministic, possibly multi-modal.
A key weakness of this type of denoising is that the posterior

\[ \mu_{X|\tilde{X}} \]

may be non-deterministic, possibly multi-modal.

If we train an autoencoder with the quadratic loss, the best reconstruction is

\[ \phi(\tilde{X}) = \mathbb{E} \left[ X \mid \tilde{X} \right], \]

which may be very unlikely under \( \mu_{X|\tilde{X}} \).
A key weakness of this type of denoising is that the posterior

$$\mu_{X|\tilde{X}}$$

may be non-deterministic, possibly multi-modal.

If we train an autoencoder with the quadratic loss, the best reconstruction is

$$\phi(\tilde{X}) = \mathbb{E}[X|\tilde{X}],$$

which may be very unlikely under $$\mu_{X|\tilde{X}}$$. 
This is what we observe with very corrupted MNIST digits.
This is what we observe with very corrupted MNIST digits.

\[ \text{Mods of } X \mid \tilde{X} \]
This is what we observe with very corrupted MNIST digits.
This can be mitigated by using in place of loss a second network that assesses if the output is realistic.

Such methods are called adversarial since the second network aims at spotting the mistakes of the first, and the first aims at fooling the second.

It can be combined with a stochastic denoiser that samples an $X$ according to $X \mid \mathcal{N}$ instead of computing a deterministic reconstruction.

We will come back to that.
Noise2Noise
Denoising can be achieved without clean samples, if the noise is additive and unbiased. Consider $\epsilon$ and $\delta$ two unbiased and independent noises. We have

\[
\begin{align*}
\mathbb{E}\left[\|\phi(X + \epsilon; \theta) - (X + \delta)\|^2\right] \\
= \mathbb{E}\left[\|\phi(X + \epsilon; \theta) - X\|^2\right] - 2\mathbb{E}\left[\langle \delta, \phi(X + \epsilon; \theta) - X \rangle\right] + \mathbb{E}\left[\|\delta\|^2\right] \\
= \mathbb{E}\left[\|\phi(X + \epsilon; \theta) - X\|^2\right] - 2\langle \mathbb{E}[\delta], \mathbb{E}[\phi(X + \epsilon; \theta) - X] \rangle + \mathbb{E}\left[\|\delta\|^2\right] \\
= \mathbb{E}\left[\|\phi(X + \epsilon; \theta) - X\|^2\right] + \mathbb{E}\left[\|\delta\|^2\right].
\end{align*}
\]

Hence

\[
\arg\min_{\theta} \mathbb{E}\left[\|\phi(X + \epsilon; \theta) - (X + \delta)\|^2\right] = \arg\min_{\theta} \mathbb{E}\left[\|\phi(X + \epsilon; \theta) - X\|^2\right].
\]

Using $L_1$ instead of $L_2$ estimates the median instead of the mean, and similarly is stable to noise that keeps the median unchanged.
Lehtinen et al. (2018)’s Noise2Noise approach uses this for image restoration, as many existing image generative processes induce an unbiased noise.

*In many image restoration tasks, the expectation of the corrupted input data is the clean target that we seek to restore. Low-light photography is an example: a long, noise-free exposure is the average of short, independent, noisy exposures.*

*Physically accurate renderings of virtual environments are most often generated through a process known as Monte Carlo path tracing. /.../ The Monte Carlo integrator is constructed such that the intensity of each pixel is the expectation of the random path sampling process, i.e., the sampling noise is zero-mean.*

(Lehtinen et al., 2018)
A. Appendix

A.1. Network architecture

Table 2 shows the network architecture used in our experiments. (Lehtinen et al., 2018)

<table>
<thead>
<tr>
<th>NAME</th>
<th>N_{out}</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>INPUT</td>
<td>n</td>
<td></td>
</tr>
<tr>
<td>ENC_CONV0</td>
<td>48</td>
<td>Convolution 3 × 3</td>
</tr>
<tr>
<td>ENC_CONV1</td>
<td>48</td>
<td>Convolution 3 × 3</td>
</tr>
<tr>
<td>POOL1</td>
<td>48</td>
<td>Maxpool 2 × 2</td>
</tr>
<tr>
<td>ENC_CONV2</td>
<td>48</td>
<td>Convolution 3 × 3</td>
</tr>
<tr>
<td>POOL2</td>
<td>48</td>
<td>Maxpool 2 × 2</td>
</tr>
<tr>
<td>ENC_CONV3</td>
<td>48</td>
<td>Convolution 3 × 3</td>
</tr>
<tr>
<td>POOL3</td>
<td>48</td>
<td>Maxpool 2 × 2</td>
</tr>
<tr>
<td>ENC_CONV4</td>
<td>48</td>
<td>Convolution 3 × 3</td>
</tr>
<tr>
<td>POOL4</td>
<td>48</td>
<td>Maxpool 2 × 2</td>
</tr>
<tr>
<td>ENC_CONV5</td>
<td>48</td>
<td>Convolution 3 × 3</td>
</tr>
<tr>
<td>POOL5</td>
<td>48</td>
<td>Maxpool 2 × 2</td>
</tr>
<tr>
<td>ENC_CONV6</td>
<td>48</td>
<td>Convolution 3 × 3</td>
</tr>
<tr>
<td>UPSAMPLE5</td>
<td>48</td>
<td>Upsample 2 × 2</td>
</tr>
<tr>
<td>CONCAT5</td>
<td>96</td>
<td>Concatenate output of POOL4</td>
</tr>
<tr>
<td>DEC_CONV5A</td>
<td>96</td>
<td>Convolution 3 × 3</td>
</tr>
<tr>
<td>DEC_CONV5B</td>
<td>96</td>
<td>Convolution 3 × 3</td>
</tr>
<tr>
<td>UPSAMPLE4</td>
<td>96</td>
<td>Upsample 2 × 2</td>
</tr>
<tr>
<td>CONCAT4</td>
<td>144</td>
<td>Concatenate output of POOL3</td>
</tr>
<tr>
<td>DEC_CONV4A</td>
<td>96</td>
<td>Convolution 3 × 3</td>
</tr>
<tr>
<td>DEC_CONV4B</td>
<td>96</td>
<td>Convolution 3 × 3</td>
</tr>
<tr>
<td>UPSAMPLE3</td>
<td>96</td>
<td>Upsample 2 × 2</td>
</tr>
<tr>
<td>CONCAT3</td>
<td>144</td>
<td>Concatenate output of POOL2</td>
</tr>
<tr>
<td>DEC_CONV3A</td>
<td>96</td>
<td>Convolution 3 × 3</td>
</tr>
<tr>
<td>DEC_CONV3B</td>
<td>96</td>
<td>Convolution 3 × 3</td>
</tr>
<tr>
<td>UPSAMPLE2</td>
<td>96</td>
<td>Upsample 2 × 2</td>
</tr>
<tr>
<td>CONCAT2</td>
<td>144</td>
<td>Concatenate output of POOL1</td>
</tr>
<tr>
<td>DEC_CONV2A</td>
<td>96</td>
<td>Convolution 3 × 3</td>
</tr>
<tr>
<td>DEC_CONV2B</td>
<td>96</td>
<td>Convolution 3 × 3</td>
</tr>
<tr>
<td>UPSAMPLE1</td>
<td>96</td>
<td>Upsample 2 × 2</td>
</tr>
<tr>
<td>CONCAT1</td>
<td>96+n</td>
<td>Concatenate input</td>
</tr>
<tr>
<td>DEC_CONV1A</td>
<td>64</td>
<td>Convolution 3 × 3</td>
</tr>
<tr>
<td>DEC_CONV1B</td>
<td>32</td>
<td>Convolution 3 × 3</td>
</tr>
<tr>
<td>DEV_CONV1C</td>
<td>m</td>
<td>Convolution 3 × 3, linear act.</td>
</tr>
</tbody>
</table>
Figure 6. Comparison of various loss functions for training a Monte Carlo denoiser with noisy target images rendered at 8 samples per pixel (spp). In this high-dynamic range setting, our custom relative loss $L_{\text{HDR}}$ is clearly superior to $L_2$. Applying a non-linear tone map to the inputs is beneficial, while applying it to the target images skews the distribution of noise and leads to wrong, visibly too dark results.

Figure 7. Denoising a Monte Carlo rendered image. (a) Image rendered with 64 samples per pixel. (b) Denoised 64 spp input, trained using 64 spp targets. (c) Same as previous, but trained on clean targets. (d) Reference image rendered with 131 072 samples per pixel. PSNR values refer to the images shown here, see text for averages over the entire validation set.

(Lehtinen et al., 2018)
We observe that if we turn the $k$-space sampling operation as a Bernoulli process where $10\%$ of spectrum samples retained and scaled by $1/p$. (b) Reconstruction by a network trained with noisy target images similar to the input image. (c) Same as previous, but training done with clean target images similar to the reference image. (d) Original, uncorrupted image. PSNR values refer to the images shown here, see text for averages over the entire validation set.

Figure 9. MRI reconstruction example. (a) Input image with only $10\%$ of spectrum samples retained and scaled by $1/p$. (b) Reconstruction by a network trained with noisy target images similar to the input image. (c) Same as previous, but training done with clean target images similar to the reference image. (d) Original, uncorrupted image. PSNR values refer to the images shown here, see text for averages over the entire validation set.

(Lehtinen et al., 2018)
Super-resolution
A special case of denoising is to increase an image resolution. We use an encoder/decoder whose encoder's input is smaller than the decoder's output.

**Encoder**

<table>
<thead>
<tr>
<th>Tensor sizes / operations</th>
<th>Encoder operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>$1 	imes 14 	imes 14$</td>
<td>$14 	imes 10$</td>
</tr>
<tr>
<td>$32 	imes 10 	imes 10$</td>
<td>$10 	imes 6$</td>
</tr>
<tr>
<td>$32 	imes 6 	imes 6$</td>
<td>$6 	imes 3$</td>
</tr>
<tr>
<td>$32 	imes 3 	imes 3$</td>
<td>$3 	imes 1$</td>
</tr>
</tbody>
</table>
MNISTUpscaler(
  (encoder): Sequential(
    (0): Conv2d(1, 32, kernel_size=(5, 5), stride=(1, 1))
    (1): ReLU(inplace=True)
    (2): Conv2d(32, 32, kernel_size=(5, 5), stride=(1, 1))
    (3): ReLU(inplace=True)
    (4): Conv2d(32, 32, kernel_size=(4, 4), stride=(1, 1))
    (5): ReLU(inplace=True)
    (6): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1))
  )
  (decoder): Sequential(
    (0): ConvTranspose2d(32, 32, kernel_size=(4, 4), stride=(1, 1))
    (1): ReLU(inplace=True)
    (2): ConvTranspose2d(32, 32, kernel_size=(3, 3), stride=(2, 2))
    (3): ReLU(inplace=True)
    (4): ConvTranspose2d(32, 32, kernel_size=(4, 4), stride=(2, 2))
    (5): ReLU(inplace=True)
    (6): ConvTranspose2d(32, 32, kernel_size=(5, 5), stride=(1, 1))
    (7): ReLU(inplace=True)
    (8): ConvTranspose2d(32, 1, kernel_size=(5, 5), stride=(1, 1))
  )
)
for original in train_input.split(batch_size):
    input = F.avg_pool2d(original, kernel_size = 2)
    output = model(input)
    loss = (output - original).pow(2).sum() / output.size(0)

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
Original

```
721041495906
901597349665
407401313472
```

Input

```
721041495906
901597349665
407401313472
```

Bilinear interpolation

```
721041495906
901597349665
407401313472
```
Lim et al. (2017) use two different resnets.

<table>
<thead>
<tr>
<th></th>
<th>EDSR</th>
<th>MDSR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nb blocks</td>
<td>32</td>
<td>80</td>
</tr>
<tr>
<td>Channels</td>
<td>256</td>
<td>64</td>
</tr>
<tr>
<td>Nb parameters</td>
<td>43M</td>
<td>8M</td>
</tr>
</tbody>
</table>

Figure 6: Qualitative comparison of our models with other works on ×4 super-resolution.

(Lim et al., 2017)
Lim et al. (2017) use two different resnets.

<table>
<thead>
<tr>
<th></th>
<th>EDSR</th>
<th>MDSR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nb blocks</td>
<td>32</td>
<td>80</td>
</tr>
<tr>
<td>Channels</td>
<td>256</td>
<td>64</td>
</tr>
<tr>
<td>Nb parameters</td>
<td>43M</td>
<td>8M</td>
</tr>
</tbody>
</table>

Figure 6: Qualitative comparison of our models with other works on \( \times 4 \) super-resolution.
Lim et al. (2017) use two different resnets.

<table>
<thead>
<tr>
<th></th>
<th>EDSR</th>
<th>MDSR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nb blocks</td>
<td>32</td>
<td>80</td>
</tr>
<tr>
<td>Channels</td>
<td>256</td>
<td>64</td>
</tr>
<tr>
<td>Nb parameters</td>
<td>43M</td>
<td>8M</td>
</tr>
</tbody>
</table>

Figure 6: Qualitative comparison of our models with other works on $\times 4$ super-resolution.
Autoencoders as self-training
Vincent et al. (2010) interpret training the autoencoder as maximizing the mutual information between the input and the latent states.
Vincent et al. (2010) interpret training the autoencoder as maximizing the mutual information between the input and the latent states.

Let $X$ be a sample, $Z = f(X; \theta)$ its latent representation, and $q(x, z)$ the distribution of $(X, Z)$. 

However, there is no expression of $q(x, z)$ in any reasonable setup.
Vincent et al. (2010) interpret training the autoencoder as maximizing the mutual information between the input and the latent states.

Let $X$ be a sample, $Z = f(X; \theta)$ its latent representation, and $q(x, z)$ the distribution of $(X, Z)$.

We have

$$\argmax_{\theta} I(X; Z) = \argmax_{\theta} \mathbb{E}_{q(x, z)} \left[ \log q(X | Z) \right].$$

However, there is no expression of $q(X | Z)$ in any reasonable setup.
For any distribution $p$ we have

$$\mathbb{E}_{q(X,Z)} \left[ \log q(X \mid Z) \right] \geq \mathbb{E}_{q(X,Z)} \left[ \log p(X \mid Z) \right].$$
For any distribution $p$ we have

$$\mathbb{E}_{q(X,Z)} \left[ \log q(X \mid Z) \right] \geq \mathbb{E}_{q(X,Z)} \left[ \log p(X \mid Z) \right].$$

So we can in particular try to find a “good $p$”, so that the left term is a good approximation of the right one.
If we consider the following model for $p$

$$p(\cdot | Z = z) = \mathcal{N}(g(z; \eta), \sigma)$$

where $g$ is deterministic and $\sigma$ fixed
If we consider the following model for $p$

$$p(\cdot | Z = z) = \mathcal{N}(g(z; \eta), \sigma)$$

where $g$ is deterministic and $\sigma$ fixed, we get

$$\mathbb{E}_{q(X,Z)} \left[ \log p(X | Z) \right] = -\frac{1}{2\sigma^2} \mathbb{E}_{q(X,Z)} \left[ \|X - g(f(X; \theta); \eta)\|^2 \right] + k.$$
If we consider the following model for $p$

$$p(\cdot \mid Z = z) = \mathcal{N}(g(z; \eta), \sigma)$$

where $g$ is deterministic and $\sigma$ fixed, we get

$$\mathbb{E}_{q(X, Z)} \left[ \log p(X \mid Z) \right] = -\frac{1}{2\sigma^2} \mathbb{E}_{q(X, Z)} \left[ \| X - g(f(X; \theta); \eta) \|^2 \right] + k.$$  

If optimizing $\eta$ makes the bound tight, the final loss is the reconstruction error

$$\argmax_{\theta} \mathbb{I}(X; Z) \simeq \argmin_{\theta, \eta} \left( \min_{\theta} \frac{1}{N} \sum_{n=1}^{N} \| x_n - g(f(x_n; \theta); \eta) \|^2 \right).$$
If we consider the following model for \( p \)

\[
p(\cdot | Z = z) = \mathcal{N}(g(z; \eta), \sigma)
\]

where \( g \) is deterministic and \( \sigma \) fixed, we get

\[
\mathbb{E}_{q(X,Z)} \left[ \log p(X | Z) \right] = -\frac{1}{2\sigma^2} \mathbb{E}_{q(X,Z)} \left[ \|X - g(f(X; \theta); \eta)\|^2 \right] + k.
\]

If optimizing \( \eta \) makes the bound tight, the final loss is the reconstruction error

\[
\arg\max_\theta \ I(X; Z) \simeq \arg\min_\theta \left( \min_\eta \frac{1}{N} \sum_{n=1}^N \|x_n - g(f(x_n; \theta); \eta)\|^2 \right).
\]

This abstract view of the encoder as “maximizing information” justifies its use to build generic encoding layers.
In the perspective of building a good feature representation, just retaining information is not enough, otherwise the identity would be a good choice.
In the perspective of building a good feature representation, just retaining information is not enough, otherwise the identity would be a good choice.

In their work, Vincent et al. consider a denoising auto-encoder, which makes the model retain information about structures beyond local noise.
Figure 6: Weight decay vs. Gaussian noise. We show typical filters learnt from natural image patches in the over-complete case (200 hidden units). *Left:* regular autoencoder with weight decay. We tried a wide range of weight-decay values and learning rates: filters never appeared to capture a more interesting structure than what is shown here. Note that some local blob detectors are recovered compared to using no weight decay at all (Figure 5 right). *Right:* a denoising autoencoder with additive Gaussian noise ($\sigma = 0.5$) learns Gabor-like local oriented edge detectors. Clearly the filters learnt are qualitatively very different in the two cases.

(Vincent et al., 2010)
Figure 7: Filters obtained on natural image patches by denoising autoencoders using other noise types. *Left:* with 10% salt-and-pepper noise, we obtain oriented Gabor-like filters. They appear slightly less localized than when using Gaussian noise (contrast with Figure 6 right). *Right:* with 55% zero-masking noise we obtain filters that look like oriented gratings. For the three considered noise types, denoising training appears to learn filters that capture meaningful natural image statistics structure.

(Vincent et al., 2010)
Vincent et al. build deep MLPs whose layers are initialized successively as encoders trained within a noisy autoencoder.
Vincent et al. build deep MLPs whose layers are initialized successively as encoders trained within a noisy autoencoder.
Vincent et al. build deep MLPs whose layers are initialized successively as encoders trained within a noisy autoencoder.
Vincent et al. build deep MLPs whose layers are initialized successively as encoders trained within a noisy autoencoder.
Vincent et al. build deep MLPs whose layers are initialized successively as encoders trained within a noisy autoencoder.
Vincent et al. build deep MLPs whose layers are initialized successively as encoders trained within a noisy autoencoder.
Vincent et al. build deep MLPs whose layers are initialized successively as encoders trained within a noisy autoencoder.
Vincent et al. build deep MLPs whose layers are initialized successively as encoders trained within a noisy autoencoder.

A final classifying layer is added and the full structure can be fine-tuned.

This approach, and others in the same spirit (Hinton et al., 2006), were seen as strategies to complement gradient-descent for building deep nets.
We considered only a coarse choice of noise levels \( \nu \) in influence of the level of corruption \( \nu \) for the image classification problems. Clearly it was not necessary to pick the noise near a better effects (we exploit the hypothesis that features of regularization help to capture P

tzanetakis

<table>
<thead>
<tr>
<th>Data Set</th>
<th>SVM_{rbf}</th>
<th>DBN-1</th>
<th>SAE-3</th>
<th>DBN-3</th>
<th>SDAE-3 (\nu)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MNIST</td>
<td>1.40±0.23</td>
<td>1.21±0.21</td>
<td>1.40±0.23</td>
<td>1.24±0.22</td>
<td>1.28±0.22 (25%)</td>
</tr>
<tr>
<td>basic</td>
<td>3.03±0.15</td>
<td>3.94±0.17</td>
<td>3.46±0.16</td>
<td>3.11±0.15</td>
<td>2.84±0.15 (10%)</td>
</tr>
<tr>
<td>rot</td>
<td>11.11±0.28</td>
<td>14.69±0.31</td>
<td>10.30±0.27</td>
<td>10.30±0.27</td>
<td>9.53±0.26 (25%)</td>
</tr>
<tr>
<td>bg-rand</td>
<td>14.58±0.31</td>
<td>9.80±0.26</td>
<td>11.28±0.28</td>
<td>6.73±0.22</td>
<td>10.30±0.27 (40%)</td>
</tr>
<tr>
<td>bg-img</td>
<td>22.61±0.37</td>
<td>16.15±0.32</td>
<td>23.00±0.37</td>
<td>16.31±0.32</td>
<td>16.68±0.33 (25%)</td>
</tr>
<tr>
<td>bg-img-rot</td>
<td>55.18±0.44</td>
<td>52.21±0.44</td>
<td>51.93±0.44</td>
<td>47.39±0.44</td>
<td>43.76±0.43 (25%)</td>
</tr>
<tr>
<td>rect</td>
<td>2.15±0.13</td>
<td>4.71±0.19</td>
<td>2.41±0.13</td>
<td>2.60±0.14</td>
<td>1.99±0.12 (10%)</td>
</tr>
<tr>
<td>rect-img</td>
<td>24.04±0.37</td>
<td>23.69±0.37</td>
<td>24.05±0.37</td>
<td>22.50±0.37</td>
<td>21.59±0.36 (25%)</td>
</tr>
<tr>
<td>convex</td>
<td>19.13±0.34</td>
<td>19.92±0.35</td>
<td>18.41±0.34</td>
<td>18.63±0.34</td>
<td>19.06±0.34 (10%)</td>
</tr>
<tr>
<td>tzanetakis</td>
<td>14.41±2.18</td>
<td>18.07±1.31</td>
<td>16.15±1.95</td>
<td>18.38±1.64</td>
<td>16.02±1.04 (0.05)</td>
</tr>
</tbody>
</table>

(Vincent et al., 2010)
The end
References


