Multiplying monochrome images as matrices: A*B and softmax

Mishka (Michael Bukatin)

JuliaCon 2021 virtual poster

I am looking for collaborators

Matrix product is interesting and important.

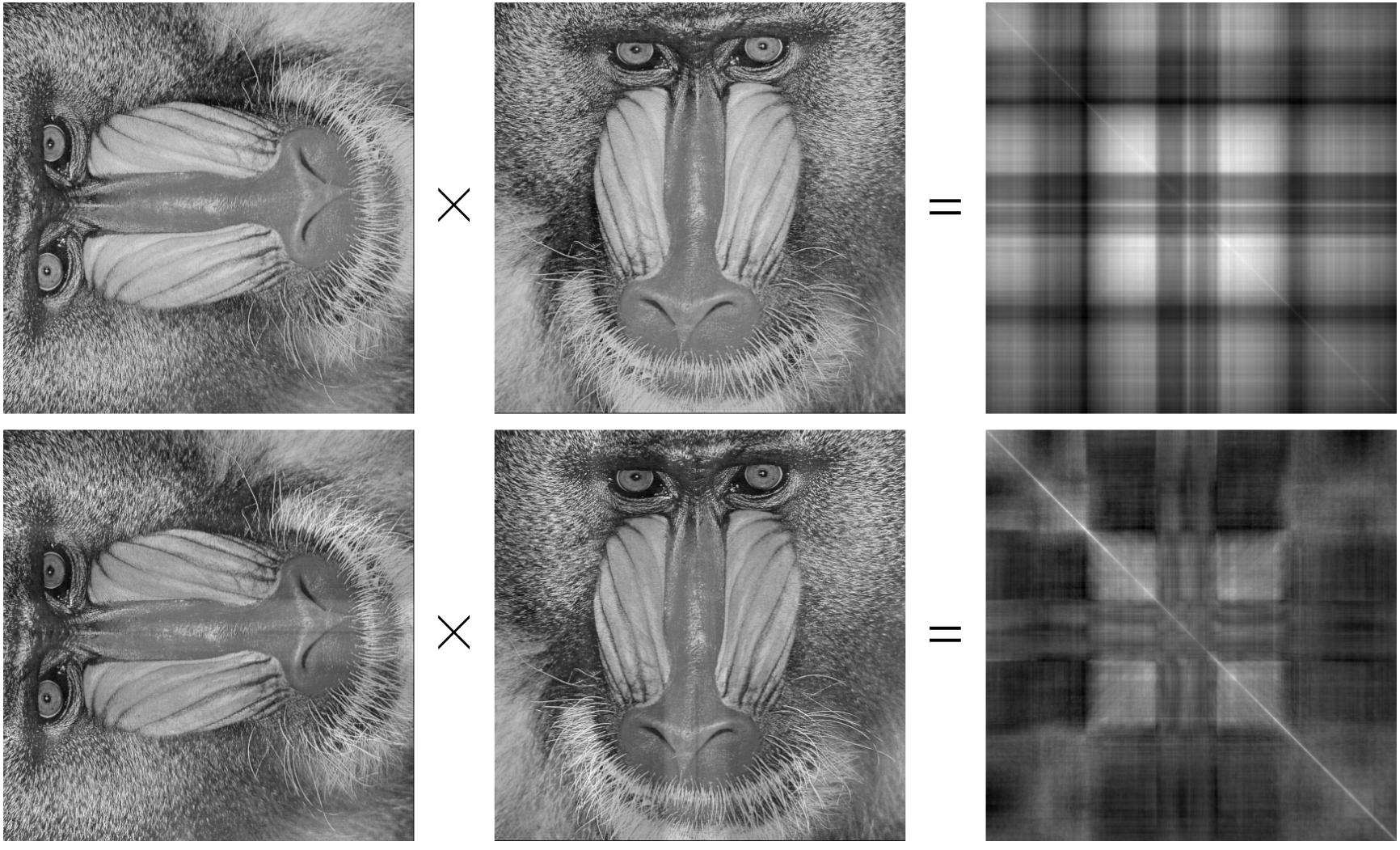
For example, it plays the key role in Transformers
(the leading class of machine learning models).

It would be great to understand it better.

Let's do something crazy:
take monochrome images and multiply them as matrices!

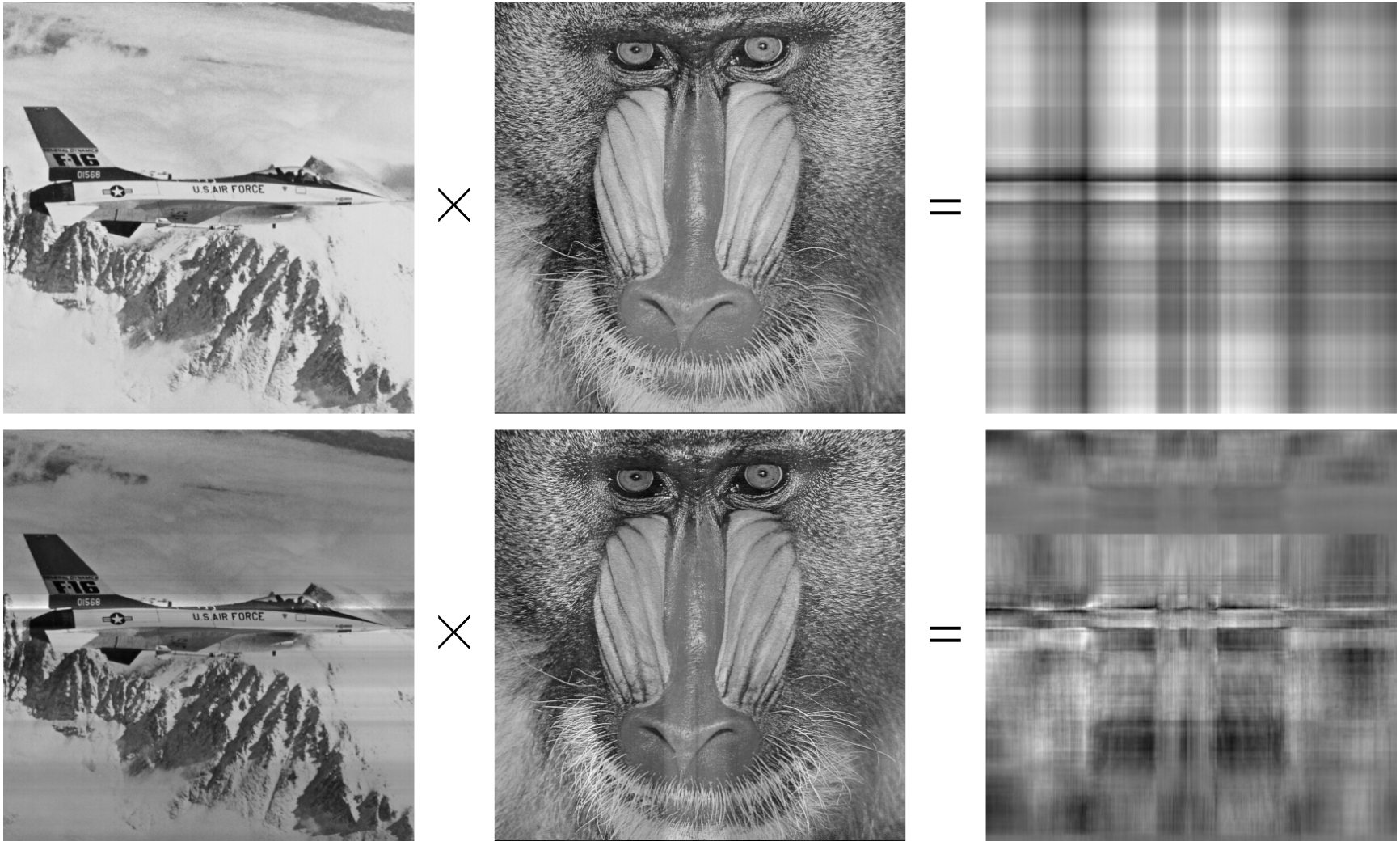
That is, multiply matrices of pixel values.

The results are visually interesting and worth further study.



In Transformers people sometimes **softmax** rows of the left matrix: $\text{Attention}(Q, K, V) = \text{softmax}(cKQ^T)V$ from "Attention Is All You Need" (2017).

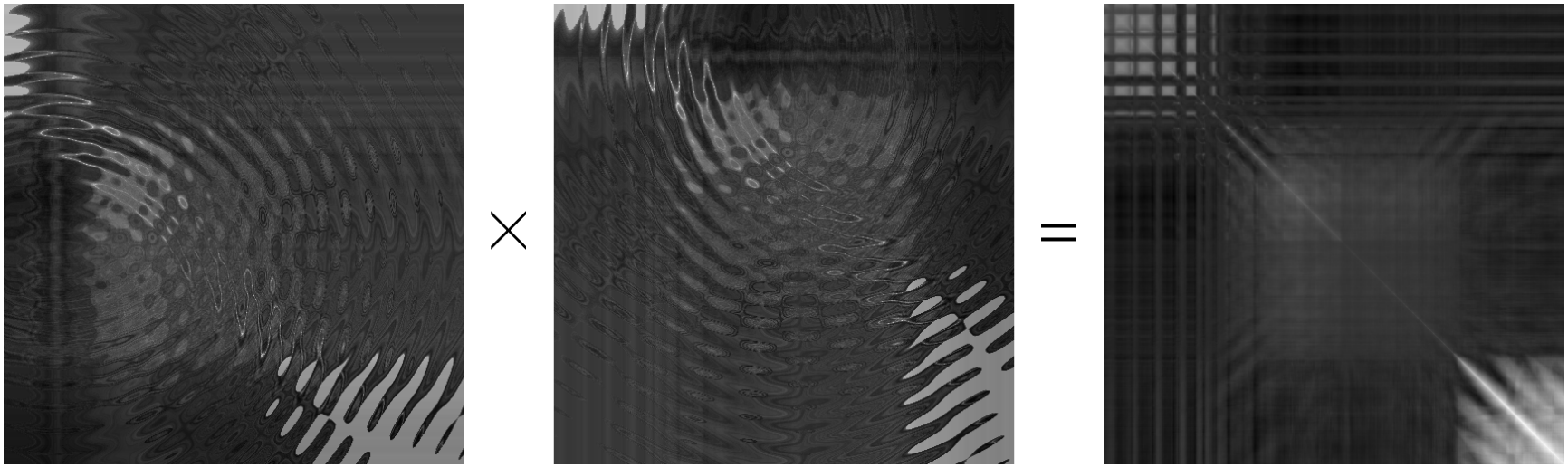
In the second example we **softmax** rows of the left matrix **and** columns of the right matrix resulting in products with richer, more fine-grained structure.



In Transformers people sometimes **softmax** rows of the left matrix: $\text{Attention}(Q, K, V) = \text{softmax}(cKQ^T)V$ from "Attention Is All You Need" (2017).

In the second example we **softmax** rows of the left matrix **and** columns of the right matrix resulting in products with richer, more fine-grained structure.

We can compose matrix product of images $(X, Y) \to X * Y$ with other image transformations, e.g. $F(Z_1, \ldots, Z_n) * G(Z_1, \ldots, Z_n)$:



The way to think about this is as follows:

- take computational elements used in Transformers
- combine them as primitives in a more flexible fashion to make small machines
- add more primitives as necessary.

We can use outstanding flexibility of differentiable programming in **Julia Flux** and solve machine learning problems involving these new flexible machines.

We give an example of finding an alternative solution to an inverse problem.

In what follows, we replace one-dimensional

```
softmax(x) = exp.(x) ./ (sum(exp.(x))
```

with

```
x -> f.(x) ./ (sum(f.(x))
```

where

```
f(y) = y + 1
```

which works fine in our case.

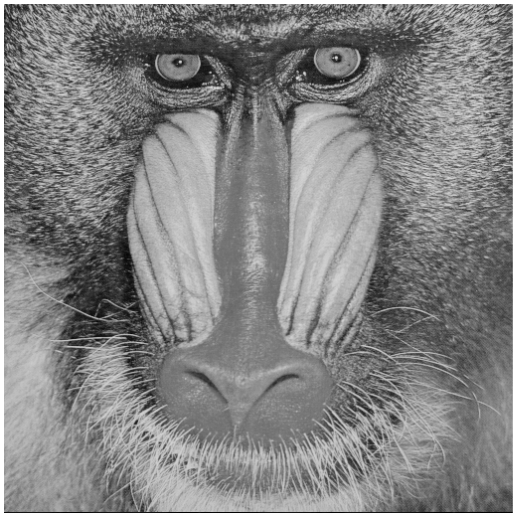
Define a matrix transformation `A -> value(A)` as follows:

```
import LinearAlgebra:  transpose, norm

function normalize_image(img)
    img1 = img .- minimum(img)
    return (1/maximum(img1))*img1
end

t_product(x) = normalize_image(transpose(x)*x)

norm_columns(f, x) = f.(x) ./ (sum(f.(x), dims=1))
norm_image_columns(f, x) = normalize_image(norm_columns(f, x))

value(A) = t_product(norm_image_columns(x -> x+1, A))
# this is used instead of true softmax, which would be
# A -> t_product(norm_image_columns(exp, A))
```

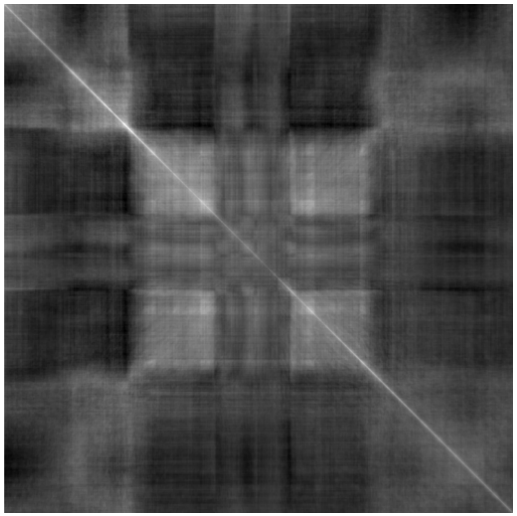
Define the loss function:

```
loss(A, B) = norm(value(A) - B)
```

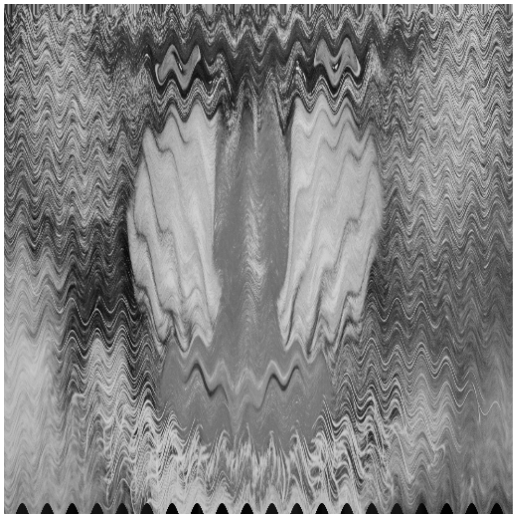
`A -> value(A)` applied to the original mandrill:



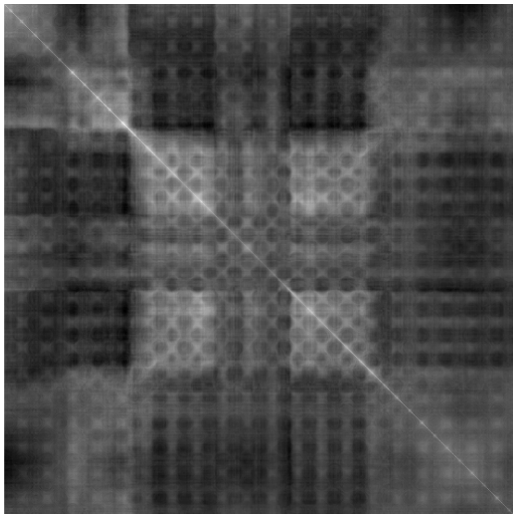
->



`A -> value(A)` applied to `oscillatory_warp(mandrill)`:



->



Let's pretend that we forgot `oscillatory_warp(mandrill)` and try to solve the following inverse problem:

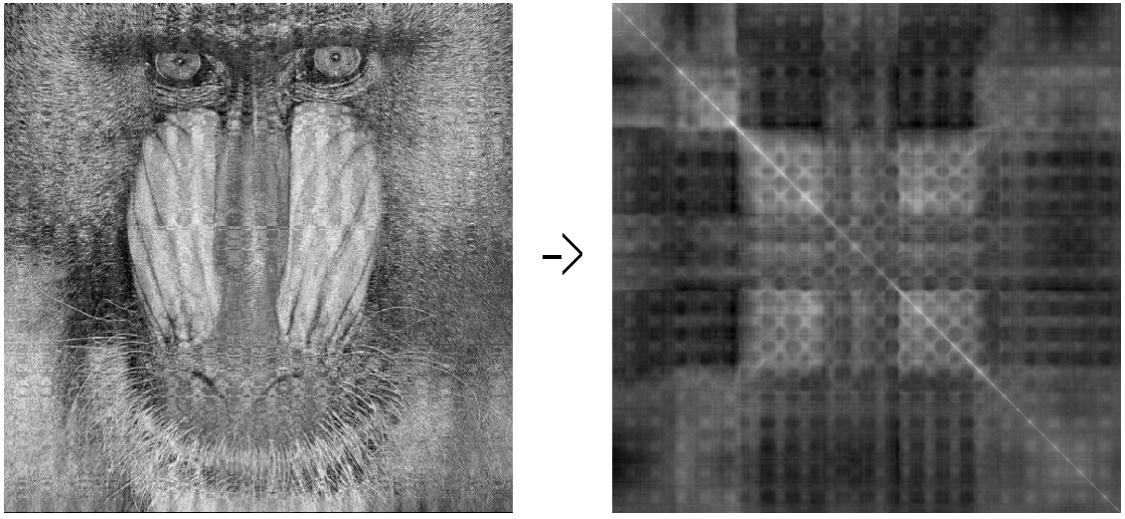
```
value(A) = value(oscillatory_warp(mandrill))
```

Initialization: start with `A` equal to the original mandrill image.

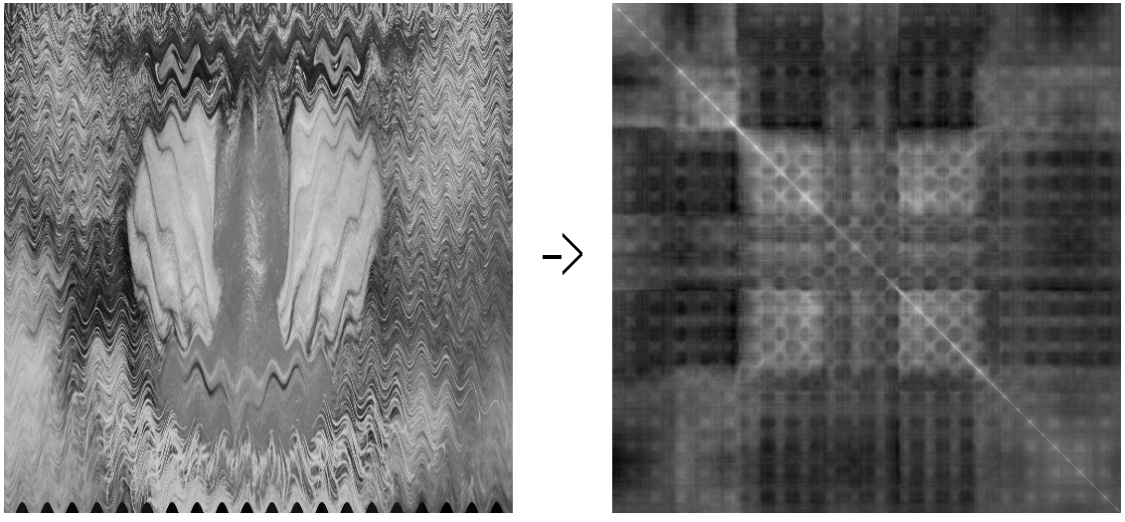
Take the gradient of `loss(A,value(oscillatory_warp(mandrill)))` with respect to all pixel values of `A` taking advantage of **Julia Flux**, and perform gradient descent with respect to `A` aiming to minimize this loss.

We found a different solution, not the original oscillatory warp:

`A -> value(A)` applied to the learned image:

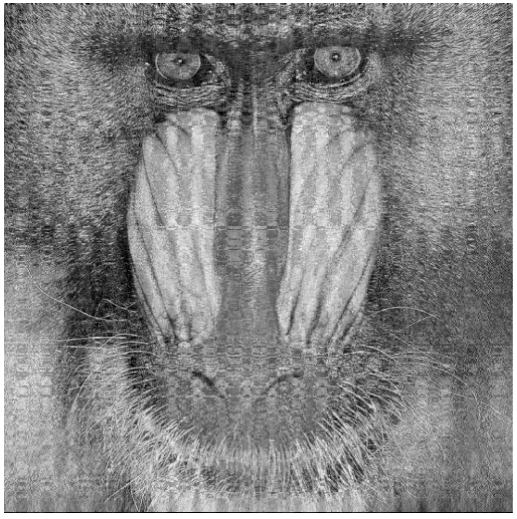


`A -> value(A)` applied to `oscillatory_warp(mandrill)`:

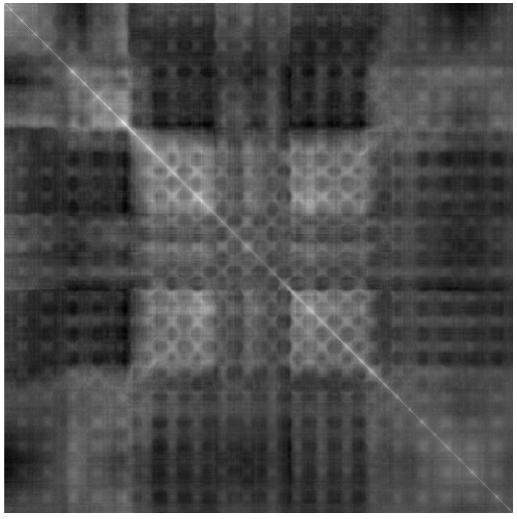


Both solutions exhibit "vertical stripes" structure:

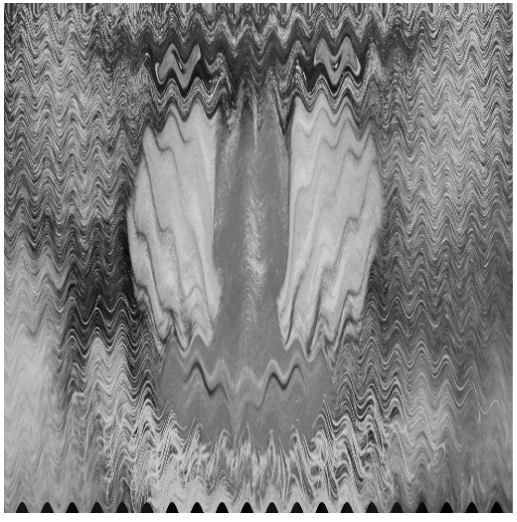
`A -> value(A)` applied to the learned image:



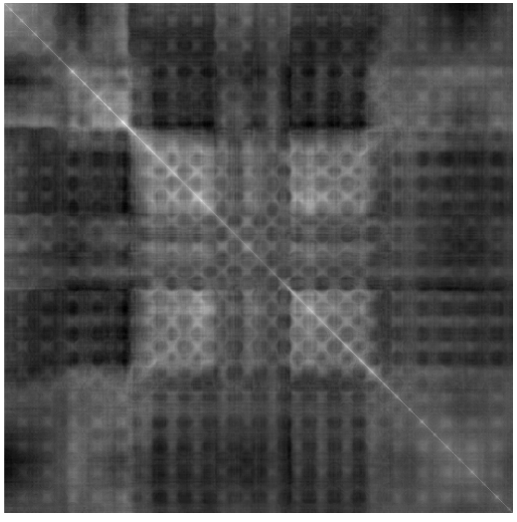
->



`A -> value(A)` applied to `oscillatory_warp(mandrill)`:

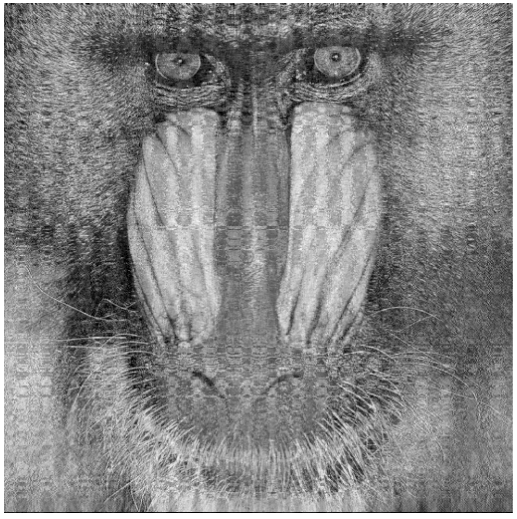


->

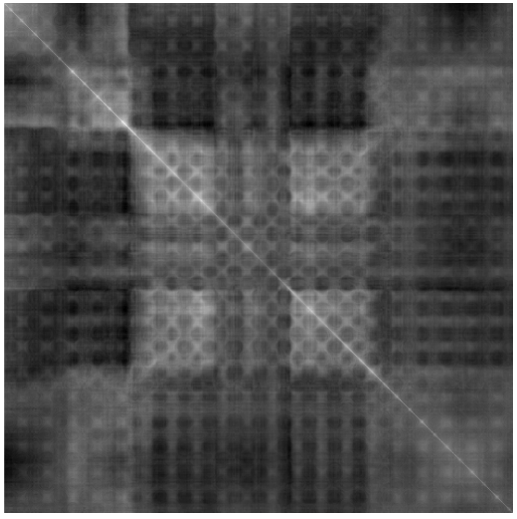


Taking gradients with respect to images is inspired by DeepDream:

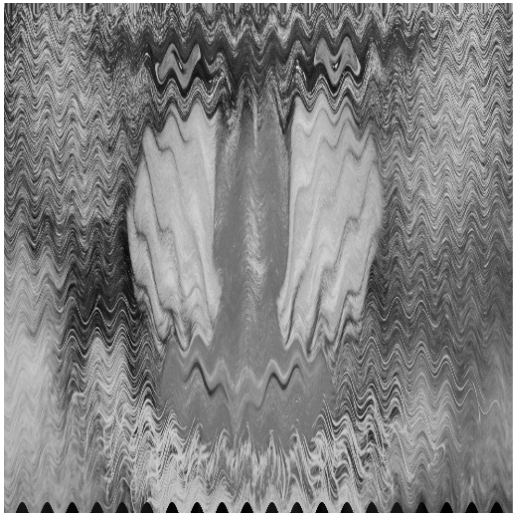
`A -> value(A)` applied to the learned image:



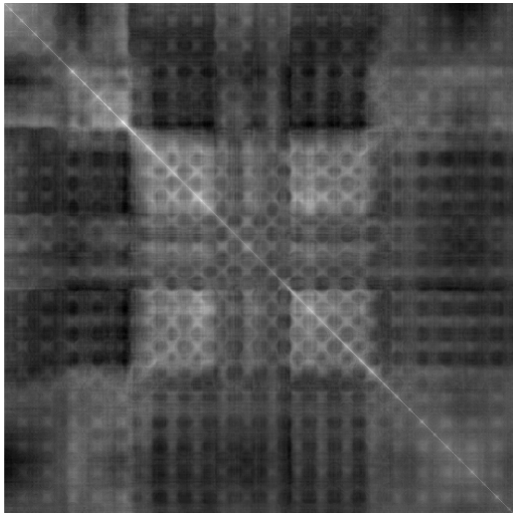
->



`A -> value(A)` applied to `oscillatory_warp(mandrill)`:

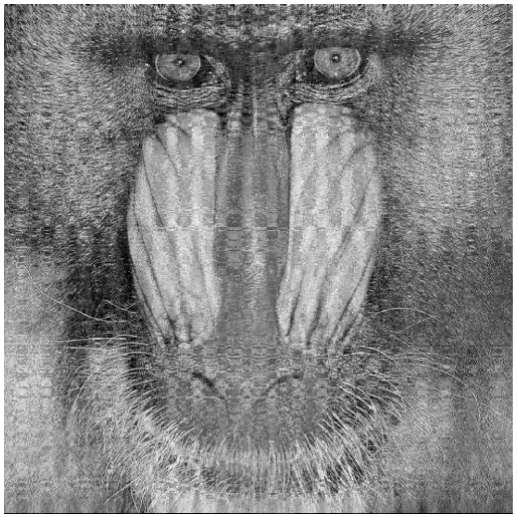
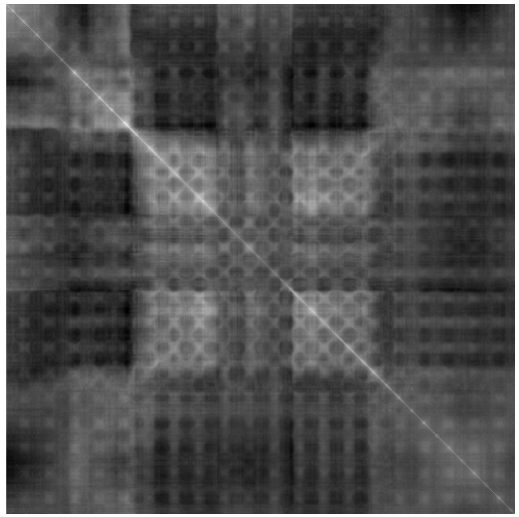


->

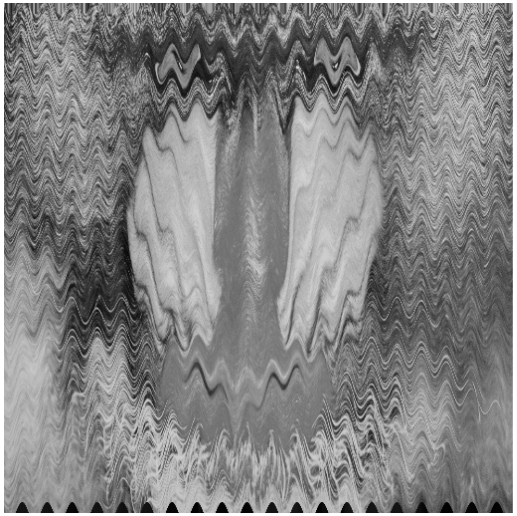
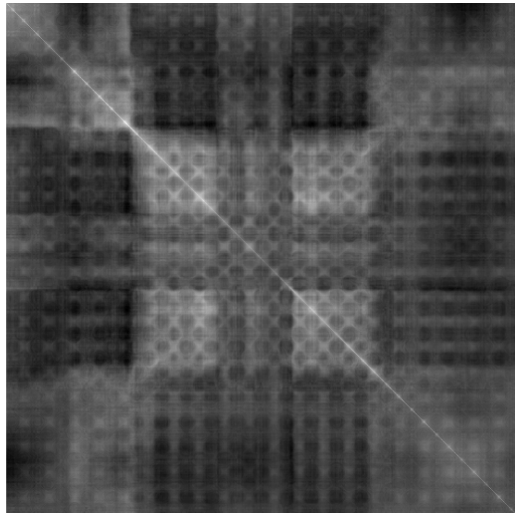


I hope this material will be useful to you.

`A -> value(A)` applied to the learned image:

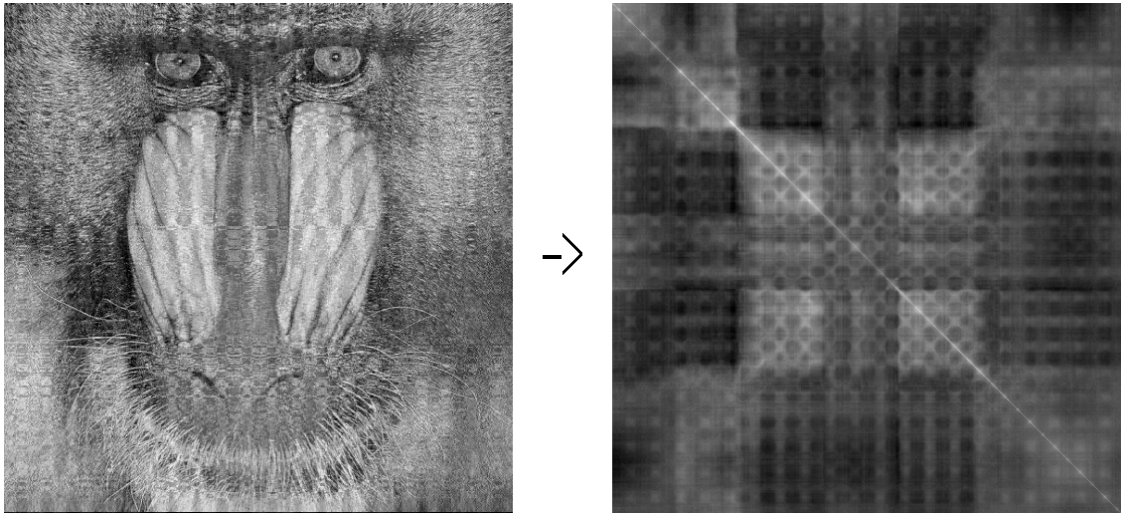



`A -> value(A)` applied to `oscillatory_warp(mandrill)`:

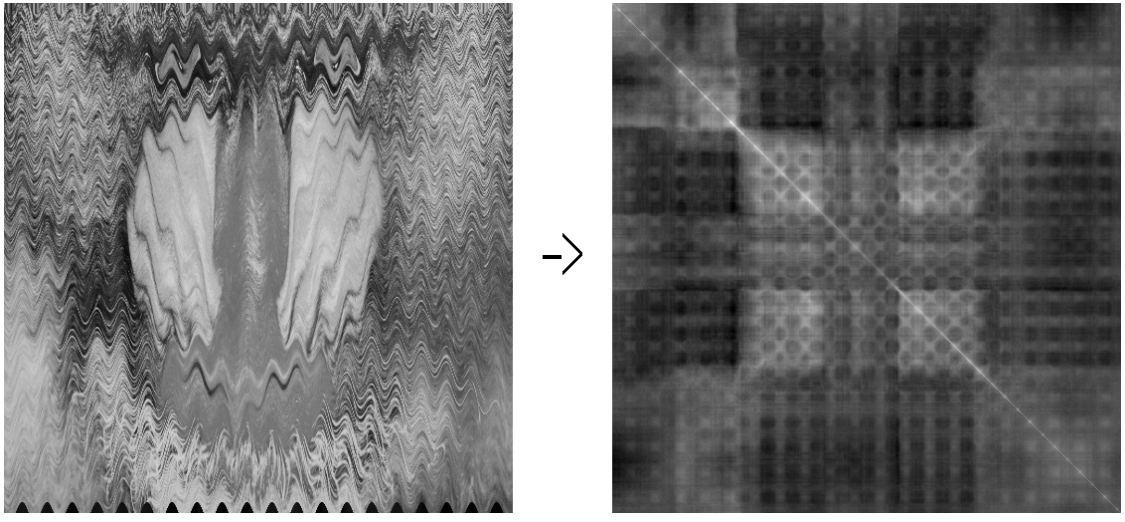



Further materials: `https://github.com/anhinga/JuliaCon2021-poster`

`A -> value(A)` applied to the learned image:

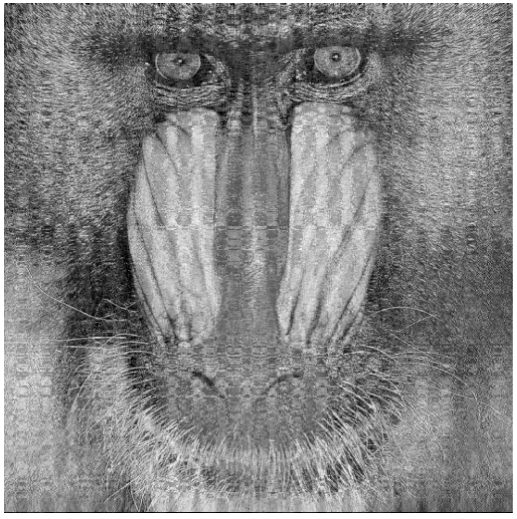


`A -> value(A)` applied to `oscillatory_warp(mandrill)`:

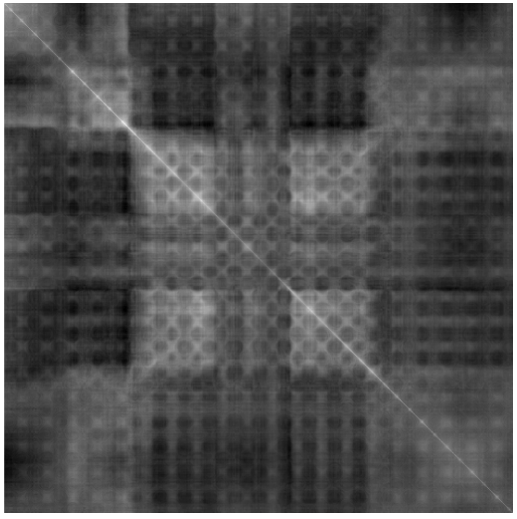


I am looking for collaborators.

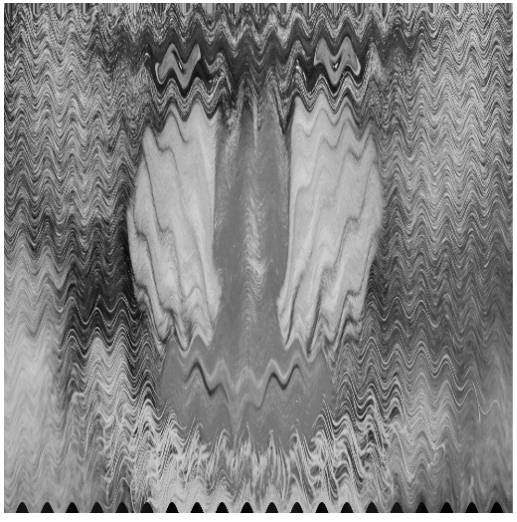
`A -> value(A)` applied to the learned image:



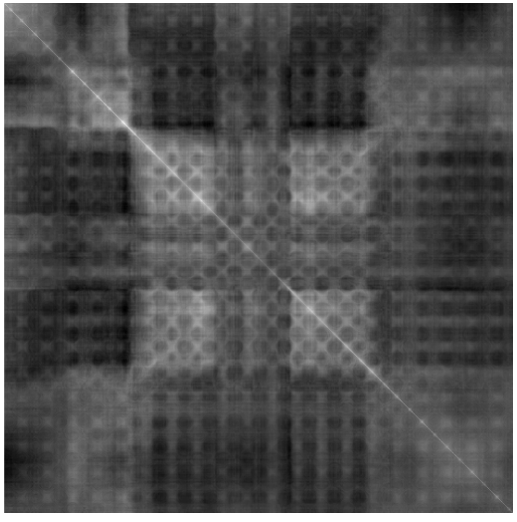
->



`A -> value(A)` applied to `oscillatory_warp(mandrill)`:

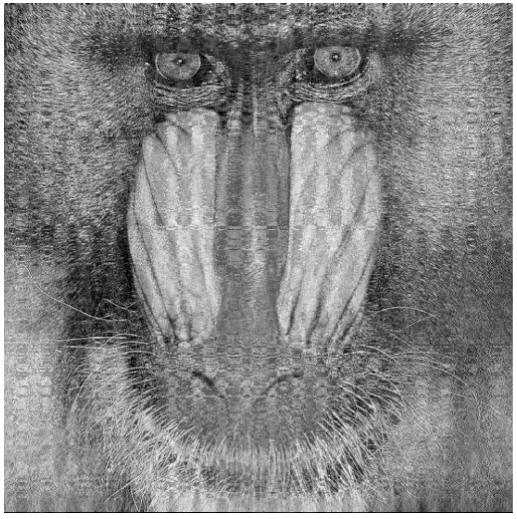


->

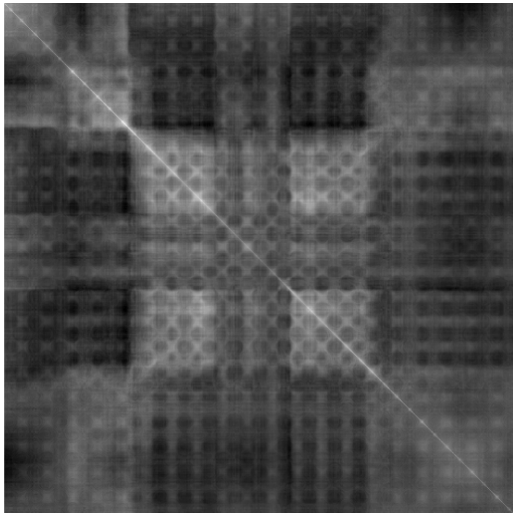


I am looking for collaborators. Thank you!!!

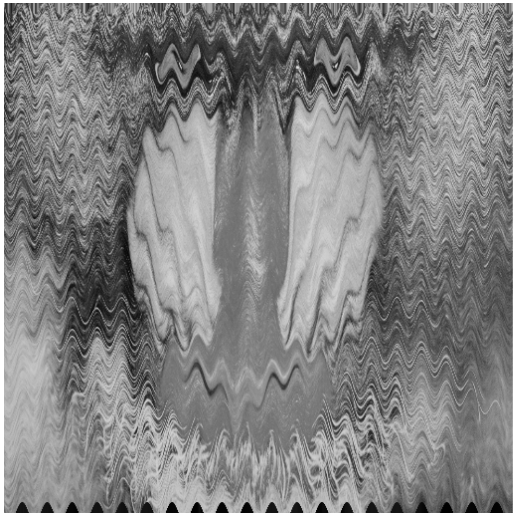
`A -> value(A)` applied to the learned image:



->



`A -> value(A)` applied to `oscillatory_warp(mandrill)`:



->