Image Convolutions

```julia
begin
    using Statistics
    using Images
    using FFTW
    using Plots
    using DSP
    using ImageFiltering
    using PlutoUI
    using OffsetArrays
end
```

```
shrink_image (generic function with 2 methods)
```

```julia
function shrink_image(image, ratio=5)
    (height, width) = size(image)
    new_height = height ÷ ratio - 1
    new_width = width ÷ ratio - 1
    list = [
            mean(image[
                ratio * i:ratio * (i + 1),
                ratio * j:ratio * (j + 1),
            ])
            for j in 1:new_width
            for i in 1:new_height
    ]
    reshape(list, new_height, new_width)
end
```

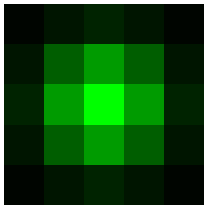
```julia
begin
    url = "https://upload.wikimedia.org/wikipedia/en/thumb/0/03/TheOreoCat.jpeg/900px-TheOreoCat.jpeg"
    download(url, "cat_in_a_hat.jpg")
    large_image = load("cat_in_a_hat.jpg")
    image = shrink_image(large_image, 7)
end
```

```
kernel = 5×5 OffsetArray(::Array{Float64,2}, -2:2, -2:2) with eltype Float64 with indice
         s -2:2×-2:2:
          0.002969016743950497  0.013306209891013651  …  0.002969016743950497
          0.013306209891013651  0.059634295436180124     0.013306209891013651
          0.02193823127971464   0.09832033134884574      0.02193823127971464
          0.013306209891013651  0.059634295436180124     0.013306209891013651
          0.002969016743950497  0.013306209891013651     0.002969016743950497
```

```julia
kernel = Kernel.gaussian((1, 1))
```

```
show_colored_kernel (generic function with 1 method)
```

```julia
function show_colored_kernel(kernel)
    to_rgb(x) = RGB(max(-x, 0), max(x,0), 0)
    to_rgb.(kernel) / maximum(abs.(kernel))
end
```



```julia
show_colored_kernel(kernel)
```

```
clamp_at_boundary (generic function with 1 method)
```

```julia
function clamp_at_boundary(M, i, j)
    return M[
        clamp(i, 1, size(M, 1)),
        clamp(j, 1, size(M, 2)),
    ]
end
```

```
3
```

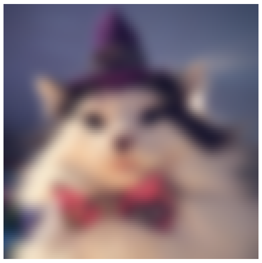
```julia
begin
    I = [1 2 3; 8 4 9]
    size(I, 2)
end
```

```
convolve (generic function with 2 methods)
```

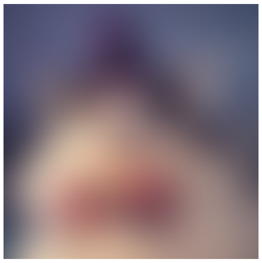
```julia
function convolve(M, kernel, M_index_function=clamp_at_boundary)
    height = size(kernel, 1)
    width = size(kernel, 2)

    half_height = height ÷ 2
    half_width = width ÷ 2

    new_image = similar(M)

    # (i, j) loop over the original image
    @inbounds for i in 1:size(M, 1)
        for j in 1:size(M, 2)
            # (k, l) loop over the neighbouring pixels
            new_image[i, j] = sum([
                        kernel[k, l] * M_index_function(M, i - k, j - l)
                        for k in -half_height:-half_height + height - 1
                        for l in -half_width:-half_width + width - 1
                    ])
        end
    end
    return new_image
end


```

```
6×6 Array{Float64,2}:
 114.85918756263384   140.0070399060573   …  112.4211026781003    83.82610630511002
  79.51984223925064   123.32722079915104      90.88366234552912   99.24989422673832
 114.99244728908782    76.50841808351247     139.35295170297752  127.39724235874077
  31.89652540904718    54.75988866400508      95.68258786210048   41.65183605959037
  13.397858245628077   33.99550908324552      64.24391644622891  111.03540092735439
  37.663498128234615   80.35542838914873  …  111.26154651019945   36.18171748110745
```

```
begin
    K = OffsetArray(gaussian((3,3), 0.25), -1:1, -1:1)
    U = rand(1.0:100.0, 6, 6);
    convolve(U, K)
end
```



```
convolve(image, Kernel.gaussian((3, 3)))
```



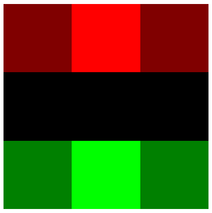
```
convolve(image, Kernel.gaussian((10, 10)))
```

```
sharpen_kernel = 3×3 OffsetArray(::Array{Float64,2}, -1:1, -1:1) with eltype Float64 wit
                 h indices -1:1×-1:1:
                  -0.5  -1.0  -0.5
                  -1.0   7.0  -1.0
                  -0.5  -1.0  -0.5
```

```
sharpen_kernel = centered([
    -0.5 -1.0 -0.5
    -1.0  7.0 -1.0
    -0.5 -1.0 -0.5
])
```

```
edge_detection_kernel_horizontal = 3×3 OffsetArray(::Array{Float64,2}, -1:1, -1:1) with
                                    eltype Float64 with indices -1:1×-1:1:
                                    -0.125  -0.25  -0.125
                                     0.0     0.0    0.0
                                     0.125   0.25   0.125
```

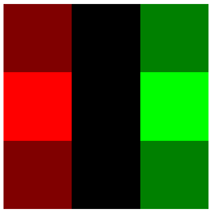
```
edge_detection_kernel_horizontal = Kernel.sobel()[1]
```



```
show_colored_kernel(edge_detection_kernel_horizontal)
```

```
edge_detection_kernel_vertical = 3×3 OffsetArray(::Array{Float64,2}, -1:1, -1:1) with el
                                 type Float64 with indices -1:1×-1:1:
                                  -0.125  0.0  0.125
                                  -0.25   0.0  0.25
                                  -0.125  0.0  0.125
```

```
edge_detection_kernel_vertical = Kernel.sobel()[2]
```

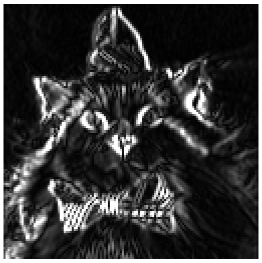


```
show_colored_kernel(edge_detection_kernel_vertical)
```

```
0.0
```

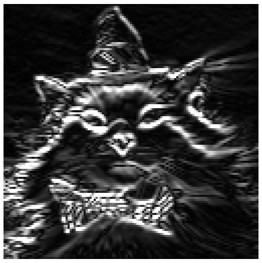
```
sum(edge_detection_kernel_vertical)
```

```
edge_enhanced_vertical =
```

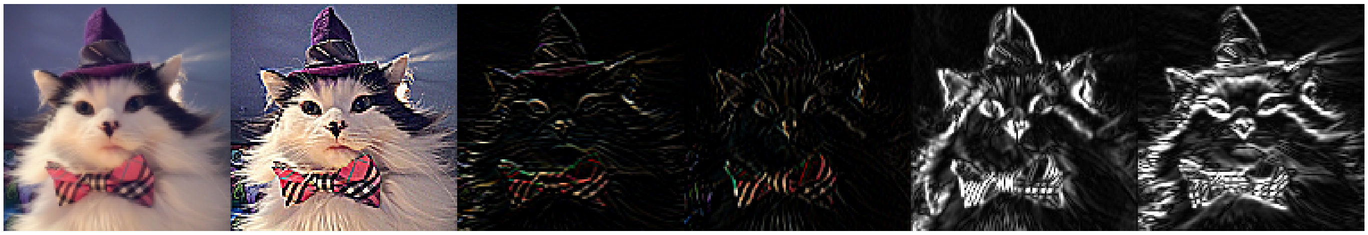


```
edge_enhanced_vertical = 3 * Gray.(abs.(convolve(image,
edge_detection_kernel_vertical)))
```

```
edge_enhanced_horizontal =
```



```
edge_enhanced_horizontal = 3 * Gray.(abs.(convolve(image,
edge_detection_kernel_horizontal)))
```



```
[image convolve(image, sharpen_kernel) convolve(convolve(image, sharpen_kernel),
edge_detection_kernel_horizontal) convolve(convolve(image, sharpen_kernel),
edge_detection_kernel_vertical) edge_enhanced_vertical edge_enhanced_horizontal]
```

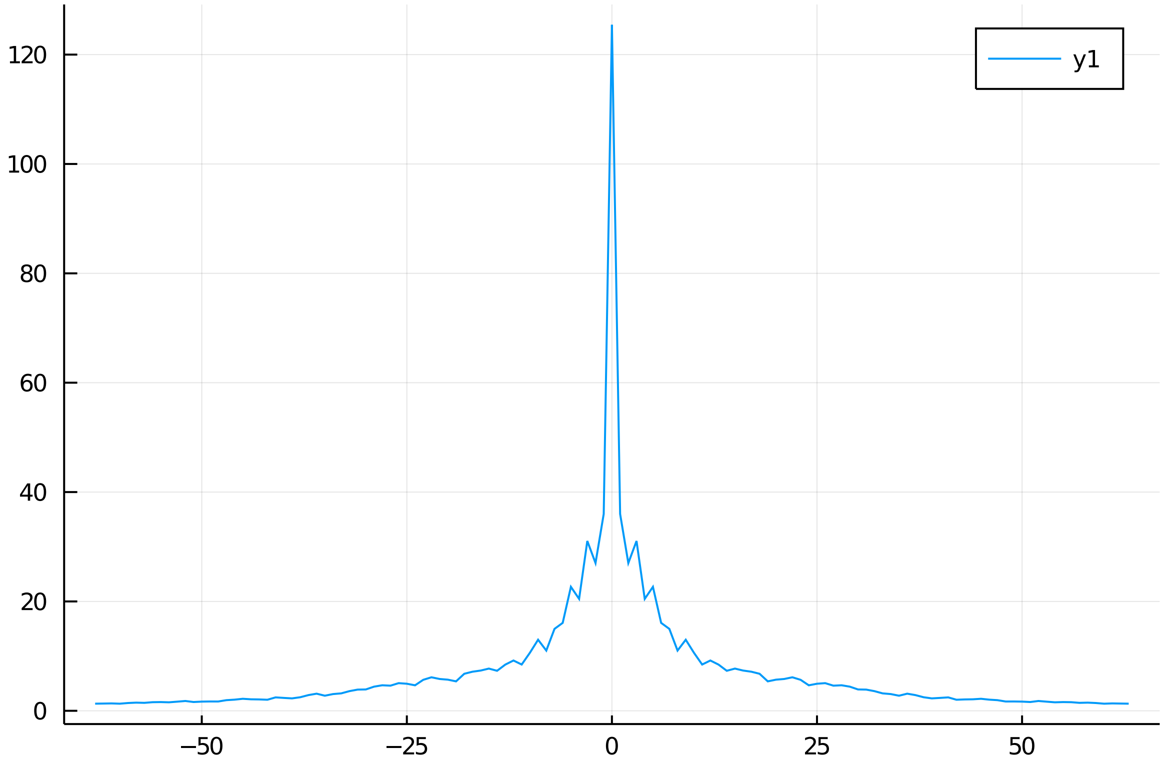
```
1.0
```

```julia
sum(sharpen_kernel)
```

Trying Fourier Transforms

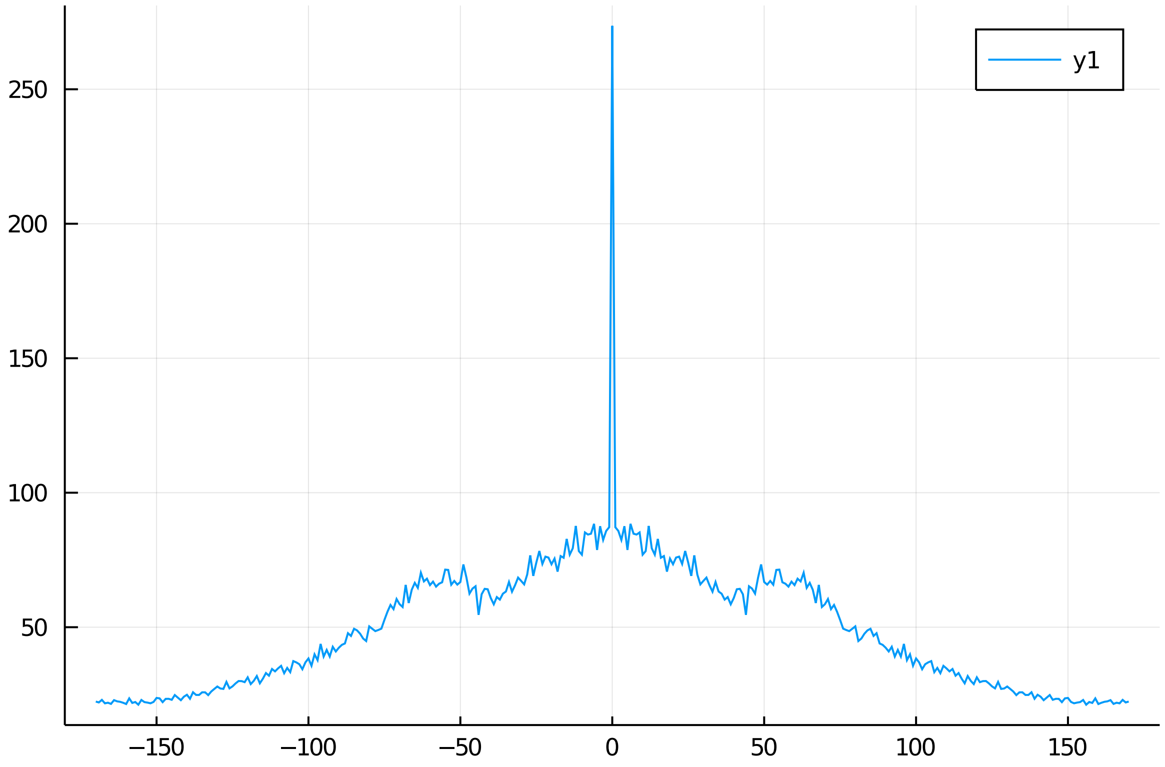
```
plot_1d_fourier_spectrum (generic function with 2 methods)
```

```julia
begin
    function rgb_to_float(color)
        return mean([color.r, color.g, color.b])
    end

    function fourier_spectrum_magnitudes(img)
        grey_values = rgb_to_float.(img)
        spectrum = fftshift(fft(grey_values))
        return abs.(spectrum)
    end

    function plot_1d_fourier_spectrum(img, dims=1)
        spectrum = fourier_spectrum_magnitudes(img)
        plot(centered(mean(spectrum, dims=1)[1:end]))
    end
end
```

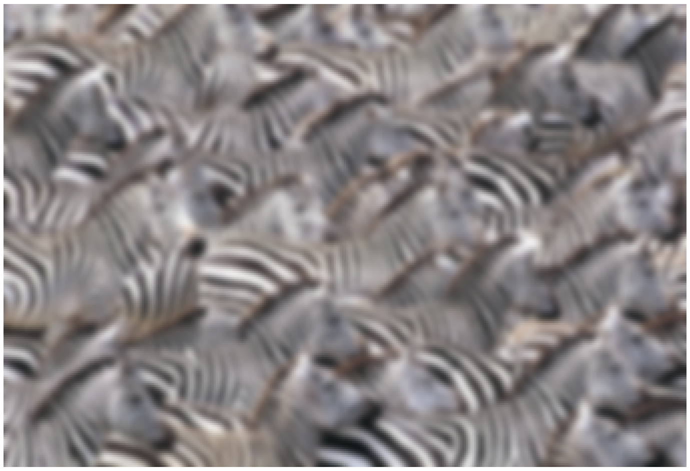


```julia
plot_1d_fourier_spectrum(image)
```

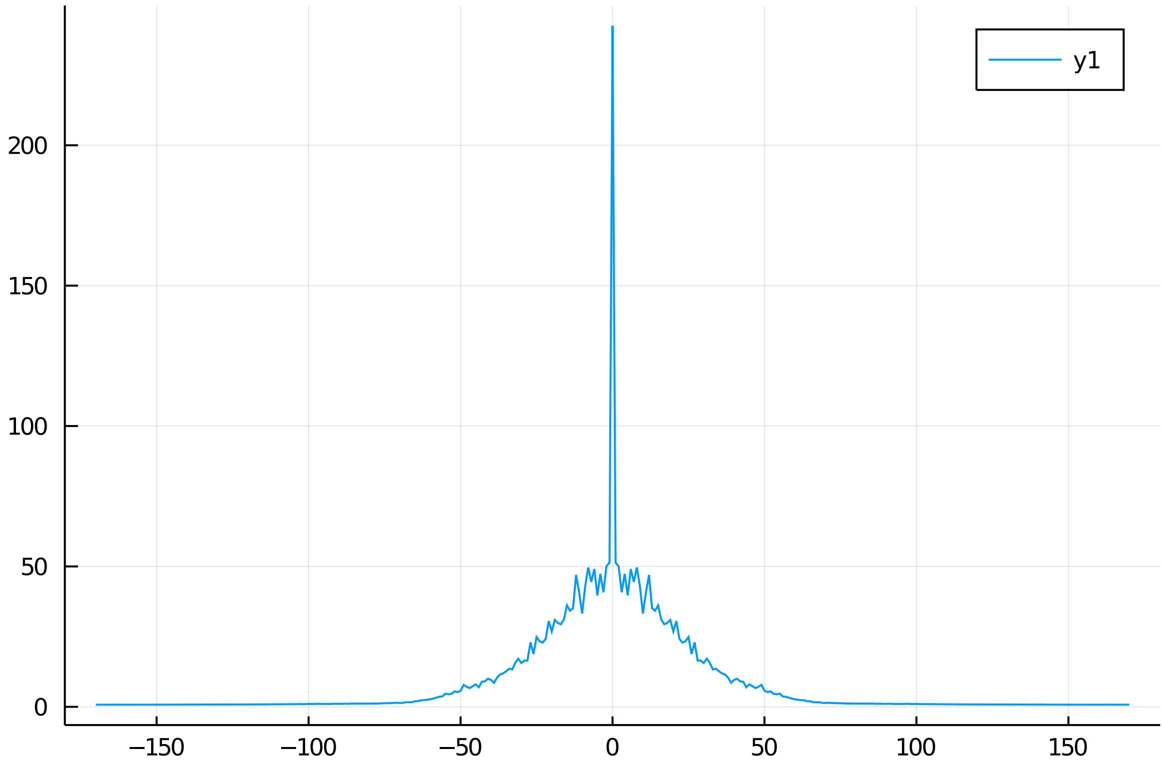
```
begin
    herd_zebras_url =
"https://i.pinimg.com/originals/3c/66/74/3c6674c2c869cccdd741379fe593294d.jpg"
    download(herd_zebras_url, "herd_zebras.jpg")
    large_zebras = load("herd_zebras.jpg")
    shrink_zebras = shrink_image(large_zebras, 7)
end
```



```
plot_1d_fourier_spectrum(shrink_zebras)
```



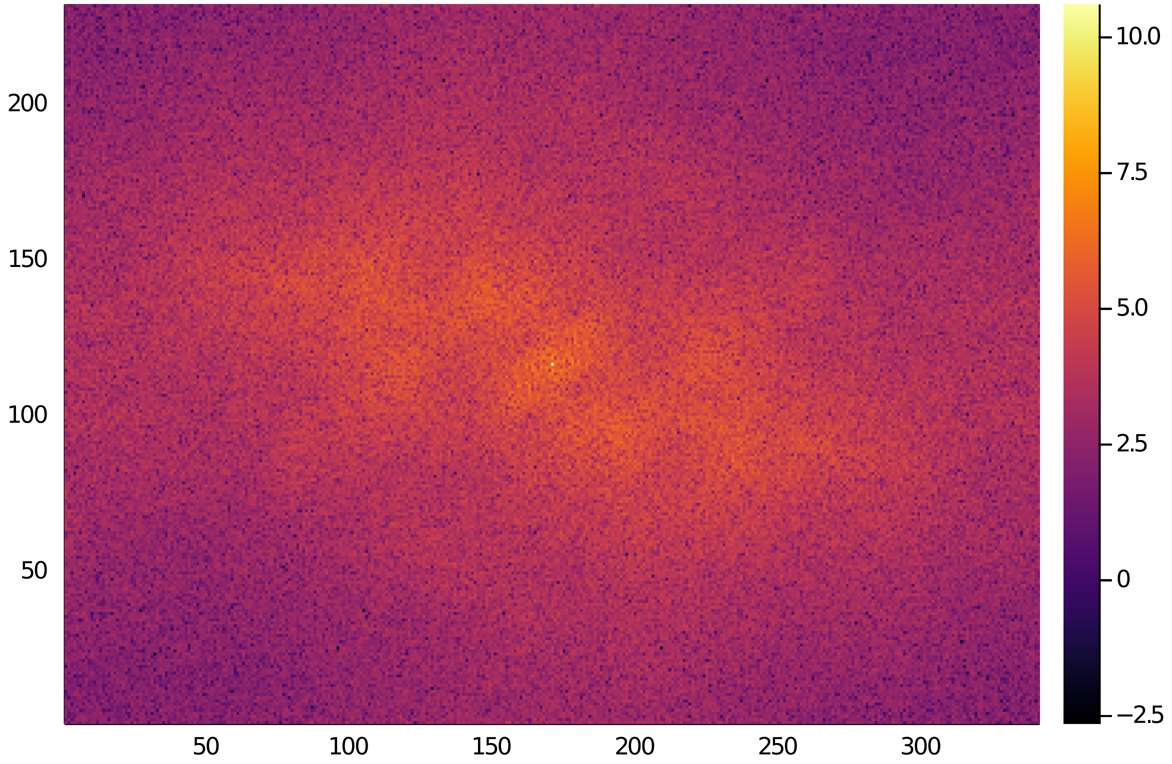
```
begin
    gauss_kernel = Kernel.gaussian((2, 2))
    conv_image = convolve(shrink_zebras, gauss_kernel)
end

```



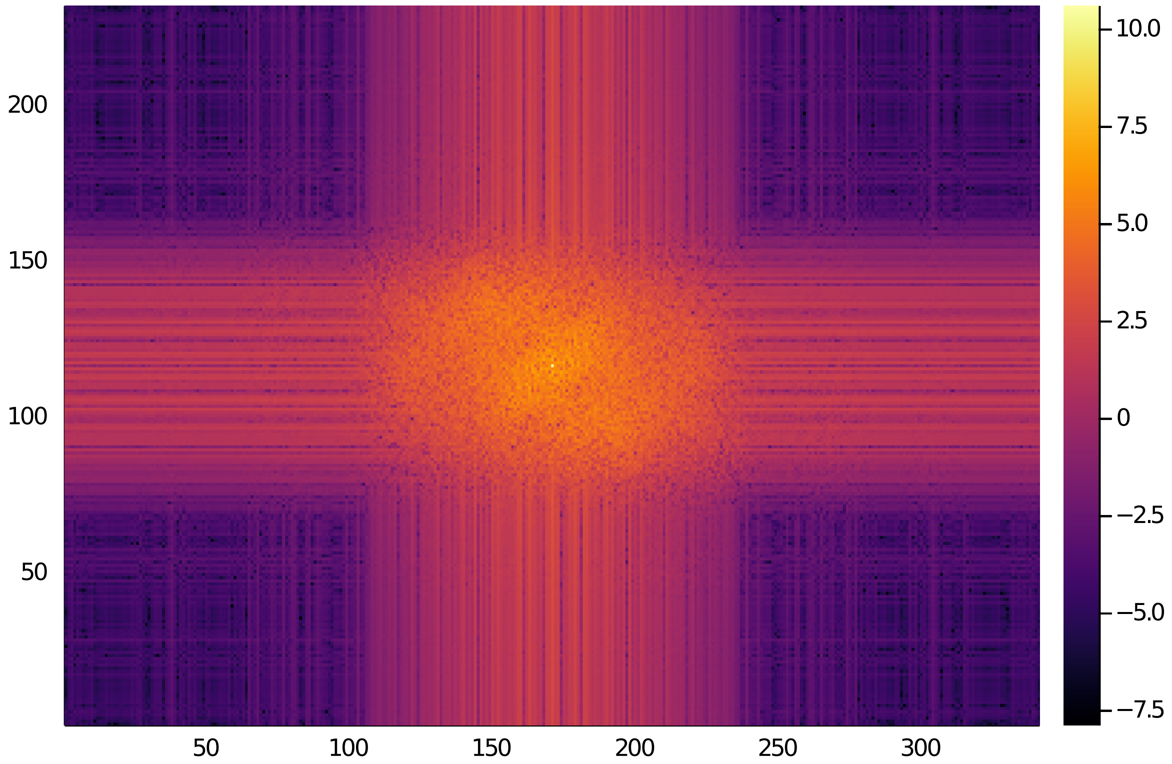
```
plot_1d_fourier_spectrum(conv_image)
```

```
heatmap_2d_fourier_spectrum (generic function with 1 method)
```

```
begin
    function heatmap_2d_fourier_spectrum(img)
        heatmap(log.(fourier_spectrum_magnitudes(img)))
    end

    function heatmap_2d_fourier_spectrum(img)
        heatmap(log.(fourier_spectrum_magnitudes(img)))
    end
end
```



```
heatmap_2d_fourier_spectrum(shrink_zebras)
```



```
heatmap_2d_fourier_spectrum(conv_image)
```

```
Enter cell code...
```