Model inference using Julia

Julia Tokyo 2019/11/29

S.Terasaki

Intro.

- Twitter
  - (ごまふあざらし)@MathSorcerer
    - Owner of Goma-chan(ゴマちゃん)
- GitHub/Qiita
  - @terasakisatoshi
- A member of Julia-Embedded
  - Maintainer of jlcross

Julia-Embedded

An organization for bringing Julia to embedded processors

Topics

- Introduction to Gomah.jl
  - Chainer2Flux
- Case Study PPN.jl
  - Code will be available soon!

Introduction to Gomah.jl

What is Gomah.jl?

- **According to https://github.com/terasakisatoshi/Gomah.jl**
  - **Gomah.jl is DNN inference library with Julia**
    - **Small project yet useful.**
- **Inspired by:**
  - **My favorite thing (You know :D)**
  - **Menoh(pfnet-research/menoh)**

How to use ?

- **Installation**
  - **Very easy even azarashi**
    - **Python**
      - **Install Chainer and ChainerCV**
    - **Julia**
      - **Install Flux, PyCall**

```
$ julia
julia> ENV["PYTHON"] = Sys.which("python3")
pkg> add https://github.com/terasakisatoshi/Gomah.jl.git
julia> using Gomah # はぁ〜ゴマちゃーん
```

Define Model with PyCall.jl

```
function get_model()
    basemodel = @pydef mutable struct MNIST <: chainer.Chain
        function __init__(self, n_hidden = 100)
            pybuiltin(:super)(MNIST, self).__init__()
            @pywith self.init_scope() begin
                self.l1 = L.Linear(nothing, n_hidden)
                self.l2 = L.Linear(nothing, n_hidden)
                self.l3 = L.Linear(nothing, 10)
            end
        end

        function __call__(self, x)
            h = F.relu(self.l1(x))
            h = F.relu(self.l2(h))
            h = self.l3(h)
            return h
        end
    end

    L.Classifier(basemodel())
end
```

Use @pywith as Context manager

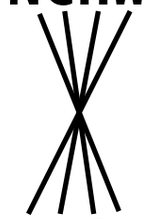
```
function predict()
    model = get_model()
    _, test_set = get_mnist()
    chainer.serializers.load_npz("result/bestmodel.npz", model)

    counter = 0

    @pywith chainer.using_config("train", false) begin
        @pywith chainer.function.no_backprop_mode() begin
            for t in test_set
                img, label = t
                y = model.predictor(np.expand_dims(img, axis = 0))
                predict = np.argmax(np.squeeze(y.array))
                if predict == label
                    counter += 1
                end
            end
        end
    end

    acc = counter / pybuiltin(:len)(test_set)
    println("accuracy for test set = $(100*acc) [%]",)
end
```

Chainer2Flux

- Gomah.jl provides weight converter from Chainer to Flux
- Supported Operators (L:= chainer.links)
  - L.Linear -> Dense
  - L.Convolution2D -> Conv
  - L.DepthwiseConvolution2D -> DepthwiseConv
  - L.BatchNormalization -> BatchNorm
- This feature enable us model inference using Julia/Flux

Idea

- **tensor format (layout of weight)**
  - **Chainer: NCHW**
  - **Flux.jl: WHCN**



```
weight::AbstractArray{T,N}
permutedims(weight, N:-1:1)
```

- **N**: number of images in the batch
- **H**: height of the image
- **W**: width of the image
- **C**: number of channels of the image (ex: 3 for RGB, 1 for grayscale...)

Chainer2Flux(Linear->Flux)

```
function ch2dense(link, σ = Flux.identity)
    W = link.W.array
    b = link.b.array
    Dense(W, b, σ)
end

function test_ch2dense()
    # get instance of Linear
    INSIZE = 10
    OUTSIZE = 20
    BSIZE = 1
    chlinear = L.Linear(in_size = INSIZE, out_size = OUTSIZE)
    dummyX = 128 * np.ones((BSIZE, INSIZE), dtype = np.float32)
    chret = reversedims(chlinear(dummyX).array)
    fldense = ch2dense(chlinear)
    flret = fldense(reversedims(dummyX))
    @test all(isapprox.(flret, chret))
end
```

NumPy <-> Julia

```
In [5]: import numpy as np

In [6]: t=np.arange(1,24+1).reshape(2,3,4)

In [7]: t
Out[7]:
array([[[ 1,  2,  3,  4],
        [ 5,  6,  7,  8],
        [ 9, 10, 11, 12]],

       [[13, 14, 15, 16],
        [17, 18, 19, 20],
        [21, 22, 23, 24]]])

In [8]: t[0,1,2]
Out[8]: 7
```

```
julia> t=reshape(1:24,4,3,2);

julia> t
4×3×2 reshape(::UnitRange{Int64}, 4, 3, 2) w
type Int64:
[:, :, 1] =
 1  5   9
 2  6  10
 3  7  11
 4  8  12

[:, :, 2] =
 13  17  21
 14  18  22
 15  19  23
 16  20  24

julia> t[2+1,1+1,0+1]
7
```

Chainer2Flux(Convolution2D->Conv)

```
function reversedims(a::AbstractArray{T,N}) where {T<:AbstractFloat,N}
    permutedims(a, N:-1:1)
end

function ch2conv(link, σ = Flux.identity)
    # get weight W and bias b
    W = reversedims(link.W.array)
    # flip kernel data
    W = W[end:-1:1, end:-1:1, :, :]
    if !isa(link.b, Nothing)
        b = reversedims(link.b.array)
    else
        b = zeros(DTYPE, size(W)[4])
    end
    pad = link.pad
    s = link.stride
    Conv(W, b, σ, pad = pad, stride = s)
end
```

```
chconv = Gomah.L.Convolution2D(
    in_channels = INCH,
    out_channels = OUTCH,
    ksize = KSIZE,
    pad = pad,
    stride = s)
# pass dummy data to create computation graph
dummyX = np.ones(
     (BSIZE, INCH, inH, inW),
     dtype = np.float32
)
chret = reversedims(chconv(dummyX).array)
flconv = ch2conv(chconv)
flret = flconv(ones(DTYPE, inW, inH, INCH,BSIZE))
```

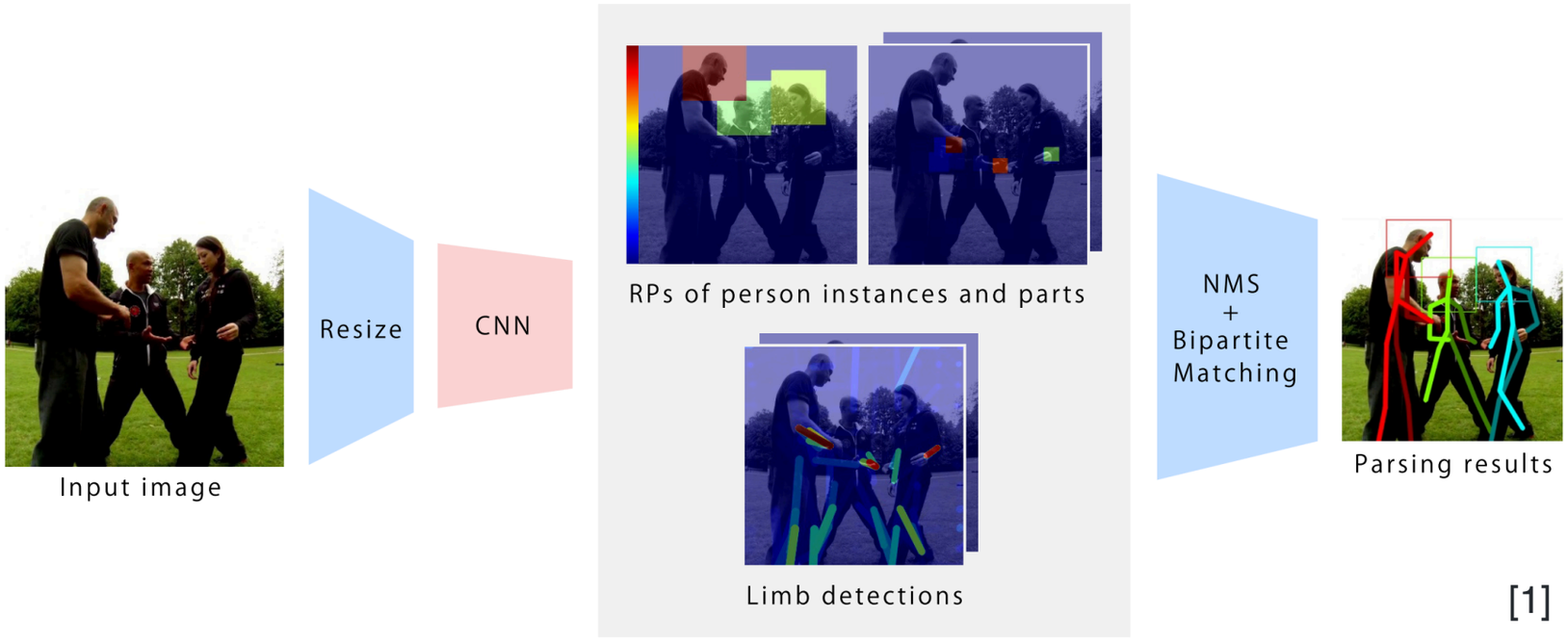
Ambitions Azarashi

- **Oh yeah! Is there any chance I can make DNN application using Julia?**
- **I searched on Google and I found a ray of hope.**
- **Key Technology:**
  - **Capture: VideoIO.jl**
  - **Process: Images.jl**
  - **Run DNN inference: PPN.jl/Gomah.jl**
  - **Draw: ImageDraw.jl**
  - **Visualize it: Makie.jl**

Case Study

Deep Learning based Pose Estimation

Pose Estimation



[1]

- We adapt Pose Proposal Networks architecture that enables us to detects multi-person 2D poses in real time.

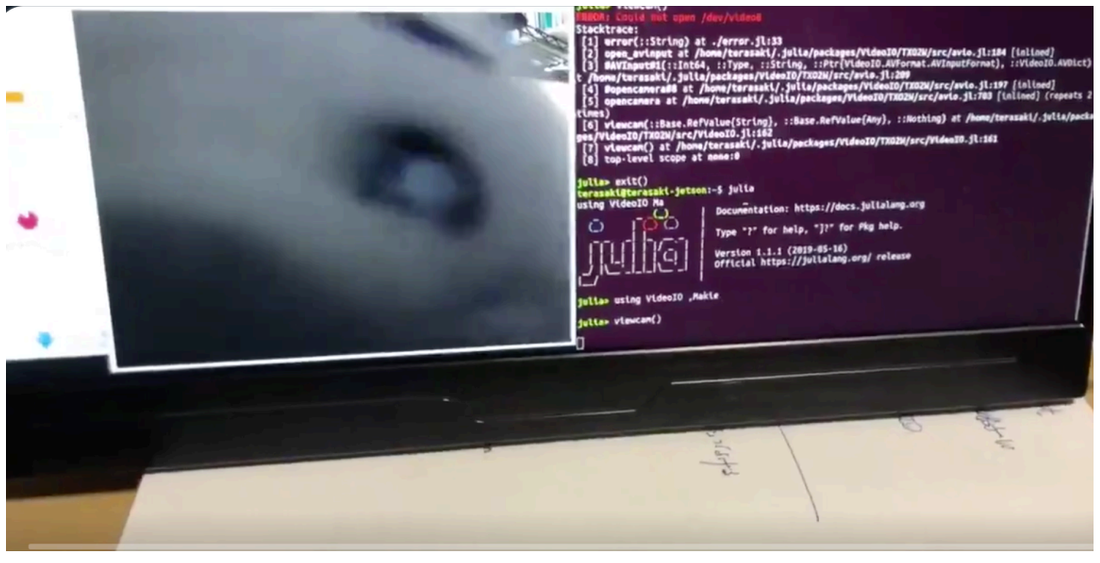
- We use pre-trained Chainer model provided by idein/chainer-pose-proposal-net, convert it to Flux via Gomah.jl and finally create application that utilizes it.

[1] Figure is taken from: Sekii, Taiki. “Pose Proposal Networks.” ECCV (2018)

Capture

- **VideoIO.jl is killer package for our application !!**
  - **Capture image without using Python/OpenCV**
  - **Just run:**

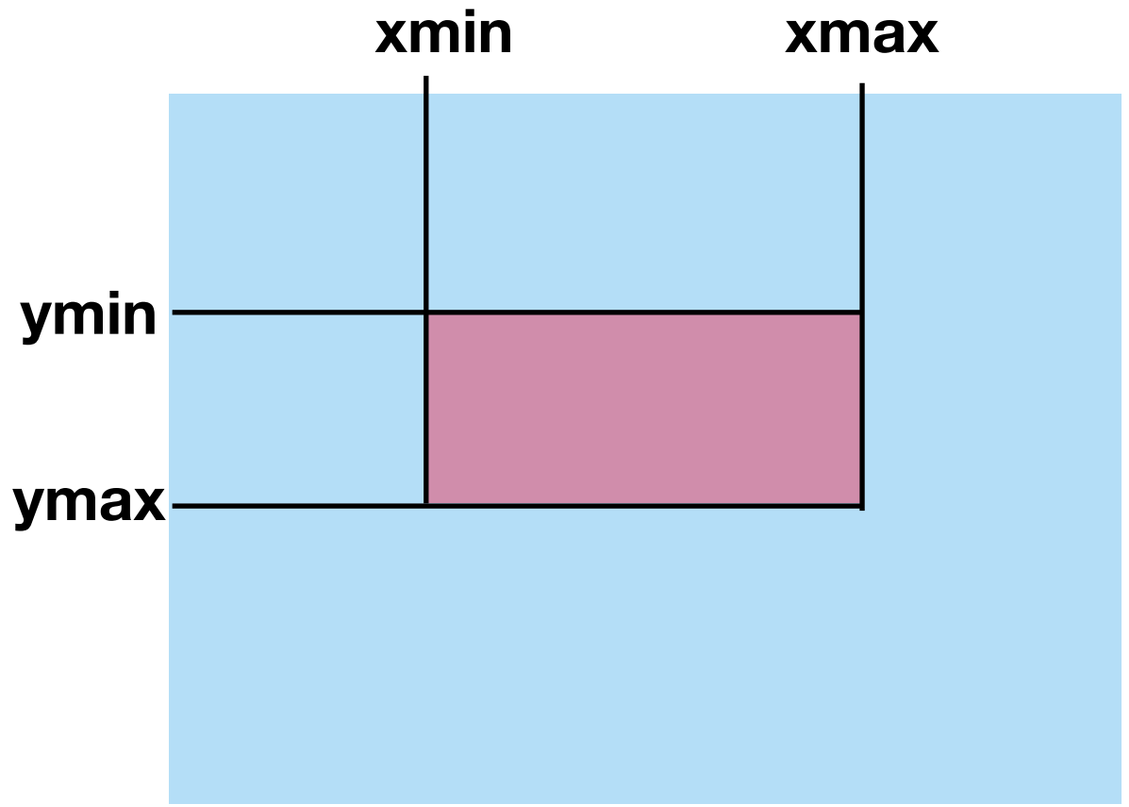
```
using VideoIO
f = opencamera()
while !eof(f)
     img = read(f)
     # do something
end
```



Process

```
using Images

function inference(img)
    inp = reshape(img, 1, size(img)…)
    inp = reversedims(inp) # NCHW -> WHCN
    feature = reversedims(flux_model(inp))
    humans = estimate(feature)
    imgshow = colorview(RGB{N0f8}, N0f8.(img / 256f0))
    draw_humans(imgshow, humans)
    imgshow
end

img = read(opencamera())
imgshow = begin
    img |>
    channelview |>
    rawview .|>
    Float32 |>
    inference
end
```

Draw

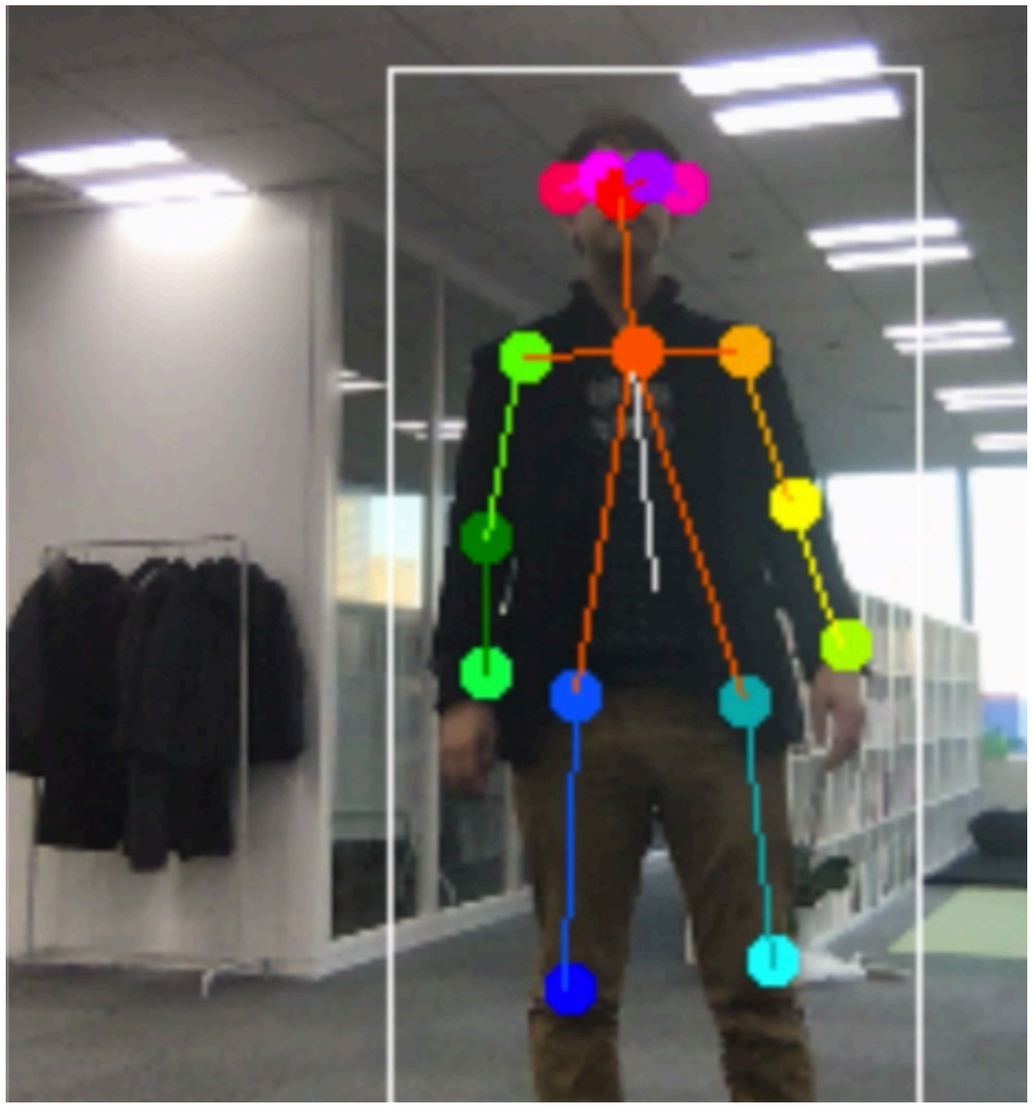



```
using ImageDraw: Polygon
function draw_bbox(img, ymin, xmin, ymax, xmax)
    vert = [
        CartesianIndex(ymin, xmin),
        CartesianIndex(ymin, xmax),
        CartesianIndex(ymax, xmax),
        CartesianIndex(ymax, xmin),
    ]
    draw!(img, Polygon(vert))
end
```

Visualize

```
# Reference VideoIO.jl

using Makie

function play(f)
    sz = min(f.height, f.width)
    scene = Makie.Scene(resolution = (sz, sz))
    buf = read(f)
    dispW, dispH = size(disp)
    makieimg = Makie.image!(
        scene,
        1:sz,
        1:sz,
        disp,
        show_axis = false,
        scale_plot = false,
    )[end]

    Makie.rotate!(scene, -π / 2)

    display(scene)

    while !eof(f) && isopen(scene)
        read!(f, buf)
        makieimg[3] = buf
    end
end
```

Result



NumPy <-> Julia

```
julia> reversedims(t)=permutedims(t,ndims(t):-1:1)
reversedims (generic function with 1 method)

julia> function pyreshape(t,pysz)
           t=reversedims(t)
           t=reshape(t,reverse(pysz))
           t=reversedims(t)
           return t
       end
pyreshape (generic function with 1 method)

julia> t=1:24
1:24

julia> t=pyreshape(t,(2,3,4))
2×3×4 Array{Int64,3}:
[:, :, 1] =
  1   5   9
 13  17  21

[:, :, 2] =
  2   6  10
 14  18  22

[:, :, 3] =
  3   7  11
 15  19  23

[:, :, 4] =
  4   8  12
 16  20  24
```

```
julia> using PyCall


julia> py"""
       import numpy as np
       arr=np.arange(1,24+1).reshape(2,3,4)
       """

julia> pyarr=py"arr"
2×3×4 Array{Int64,3}:
[:, :, 1] =
  1   5   9
 13  17  21

[:, :, 2] =
  2   6  10
 14  18  22

[:, :, 3] =
  3   7  11
 15  19  23

[:, :, 4] =
  4   8  12
 16  20  24
```

Conclusion

- **We've introduced Gomah.jl**
- **We've shown we can create DNN application with Julia**

KyuKyuKyu=3Kyu