Chest X-Ray Images (Pneumonia)

My first kernel in Julia using the data from **this** Kaggle challenge

Some Hyperparameters

```
begin
    img_size = (224, 224)
    batchsize = 1
end;
```

Load training data

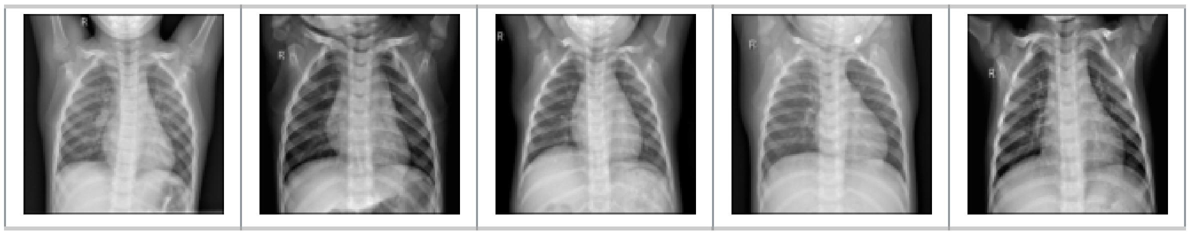
```
using Images    , FileIO    , Colors    , Flux    , Metalhead
```

```
begin
    # TODO: currently only loading first 5
    Xtrain_normal = load.(readdir("data/train/NORMAL/", join=true)[1:5])
    Xtrain_normal = [imresize(img, img_size) for img in Xtrain_normal]
    ytrain_normal = repeat([0], length(Xtrain_normal))

    Xtrain_pneumonia = load.(readdir("data/train/PNEUMONIA/", join=true)[1:5])
    Xtrain_pneumonia = [imresize(img, img_size) for img in Xtrain_pneumonia]
    ytrain_pneumonia = repeat([1], length(Xtrain_pneumonia))
end;
```

Show them to me

Display first 5 X-Rays of **normal** patients



(a vector displayed as a row to save space) and first 5 of **pneumonia** patients

(a vector displayed as a row to save space)

Merge both classes and create a DataLoader

```
begin
    Xtrain = append!([], Xtrain_normal, Xtrain_pneumonia)
    ytrain = append!([], ytrain_normal, ytrain_pneumonia)

    # xs = [colorview(RGB, img, img, img) for img in Xtrain] # grayscale -> rgb

    xs = [Flux.unsqueeze(Float64.(img), 3) for img in Xtrain]
    ys = Flux.onehotbatch(ytrain, 0:1)

    train_loader = Flux.DataLoader((data=xs, label=ys),
                                   batchsize=batchsize, shuffle=true)
end;
```

```
(224, 224, 1)
```

```
begin
    img = xs[1]
    img |> size
end
```

Define first model

```
begin
    const vgg_config = Dict(:A => [(64,1), (128,1), (256,2), (512,2), (512,2)],
                            :B => [(64,2), (128,2), (256,2), (512,2), (512,2)],
                            :D => [(64,2), (128,2), (256,3), (512,3), (512,3)],
                            :E => [(64,2), (128,2), (256,4), (512,4), (512,4)])

    model = VGG(img_size; config = vgg_config[:D],
                          inchannels = 1,
                          batchnorm = false,
                          nclasses = 2,
                          fcsize = 4096,
                          dropout = 0.5)
end;
```

```
DimensionMismatch("Rank of x and w must match! (1 vs. 4)")

 1. var"#DenseConvDims#7"(::Base.Iterators.Pairs{Symbol, Any, NTuple{4, Symbol},
    NamedTuple{(:stride, :padding, :dilation, :groups), Tuple{Tuple{Int64, Int64},
    Tuple{Int64, Int64}, Tuple{Int64, Int64}, Int64}}}, ::Type{NNlib.DenseConvDims},
    ::Vector{Array{Float64, 3}}, ::Array{Float32, 4}) @ DenseConvDims.jl:58
 2. chain_rrule_kw @ chainrules.jl:203 [inlined]
 3. macro expansion @ interface2.jl:0 [inlined]
 4. _pullback(::Zygote.Context, ::Core.var"#Type##kw", ::NamedTuple{(:stride, :padding,
    :dilation, :groups), Tuple{Tuple{Int64, Int64}, Tuple{Int64, Int64}, Tuple{Int64,
    Int64}, Int64}}, ::Type{NNlib.DenseConvDims}, ::Vector{Array{Float64, 3}},
    ::Array{Float32, 4}) @ interface2.jl:9
 5. _pullback @ conv.jl:162 [inlined]
 6. _pullback(::Zygote.Context, ::Flux.Conv{2, 2, typeof(NNlib.relu), Array{Float32,
    4}, Vector{Float32}}, ::Vector{Array{Float64, 3}}) @ interface2.jl:0
 7. _pullback @ basic.jl:47 [inlined]
 8. _pullback(::Zygote.Context, ::typeof(Flux.applychain), ::Tuple{Flux.Conv{2, 2,
    typeof(NNlib.relu), Array{Float32, 4}, Vector{Float32}}, Flux.Conv{2, 2,
    typeof(NNlib.relu), Array{Float32, 4}, Vector{Float32}}, Flux.MaxPool{2, 4},
    Flux.Conv{2, 2, typeof(NNlib.relu), Array{Float32, 4}, Vector{Float32}},
    Flux.Conv{2, 2, typeof(NNlib.relu), Array{Float32, 4}, Vector{Float32}},
    Flux.MaxPool{2, 4}, Flux.Conv{2, 2, typeof(NNlib.relu), Array{Float32, 4},
    Vector{Float32}}, Flux.Conv{2, 2, typeof(NNlib.relu), Array{Float32, 4},
    Vector{Float32}}, Flux.Conv{2, 2, typeof(NNlib.relu), Array{Float32, 4},
    Vector{Float32}}, Flux.MaxPool{2, 4}, Flux.Conv{2, 2, typeof(NNlib.relu),
    Array{Float32, 4}, Vector{Float32}}, Flux.Conv{2, 2, typeof(NNlib.relu),
    Array{Float32, 4}, Vector{Float32}}, Flux.Conv{2, 2, typeof(NNlib.relu),
    Array{Float32, 4}, Vector{Float32}}, Flux.MaxPool{2, 4}, Flux.Conv{2, 2,
    typeof(NNlib.relu), Array{Float32, 4}, Vector{Float32}}, Flux.Conv{2, 2,
    typeof(NNlib.relu), Array{Float32, 4}, Vector{Float32}}, Flux.Conv{2, 2,
    typeof(NNlib.relu), Array{Float32, 4}, Vector{Float32}}, Flux.MaxPool{2, 4}},
    ::Vector{Array{Float64, 3}}) @ interface2.jl:0
 9. _pullback @ basic.jl:49 [inlined]
10. _pullback(::Zygote.Context, ::Flux.Chain{Tuple{Flux.Conv{2, 2, typeof(NNlib.relu),
    Array{Float32, 4}, Vector{Float32}}, Flux.Conv{2, 2, typeof(NNlib.relu),
    Array{Float32, 4}, Vector{Float32}}, Flux.MaxPool{2, 4}, Flux.Conv{2, 2,
    typeof(NNlib.relu), Array{Float32, 4}, Vector{Float32}}, Flux.Conv{2, 2,
    typeof(NNlib.relu), Array{Float32, 4}, Vector{Float32}}, Flux.MaxPool{2, 4},
    Flux.Conv{2, 2, typeof(NNlib.relu), Array{Float32, 4}, Vector{Float32}},
    Flux.Conv{2, 2, typeof(NNlib.relu), Array{Float32, 4}, Vector{Float32}},
    Flux.Conv{2, 2, typeof(NNlib.relu), Array{Float32, 4}, Vector{Float32}},
    Flux.MaxPool{2, 4}, Flux.Conv{2, 2, typeof(NNlib.relu), Array{Float32, 4},
    Vector{Float32}}, Flux.Conv{2, 2, typeof(NNlib.relu), Array{Float32, 4},
    Vector{Float32}}, Flux.Conv{2, 2, typeof(NNlib.relu), Array{Float32, 4},
    Vector{Float32}}, Flux.MaxPool{2, 4}, Flux.Conv{2, 2, typeof(NNlib.relu),
    Array{Float32, 4}, Vector{Float32}}, Flux.Conv{2, 2, typeof(NNlib.relu),
    Array{Float32, 4}, Vector{Float32}}, Flux.Conv{2, 2, typeof(NNlib.relu),
    Array{Float32, 4}, Vector{Float32}}, Flux.MaxPool{2, 4}}}, ::Vector{Array{Float64,
    3}}) @ interface2.jl:0
11. _pullback @ basic.jl:47 [inlined]
12. _pullback(::Zygote.Context, ::typeof(Flux.applychain),
    ::Tuple{Flux.Chain{Tuple{Flux.Conv{2, 2, typeof(NNlib.relu), Array{Float32, 4},
    Vector{Float32}}, Flux.Conv{2, 2, typeof(NNlib.relu), Array{Float32, 4},
    Vector{Float32}}, Flux.MaxPool{2, 4}, Flux.Conv{2, 2, typeof(NNlib.relu),
    Array{Float32, 4}, Vector{Float32}}, Flux.Conv{2, 2, typeof(NNlib.relu),
    Array{Float32, 4}, Vector{Float32}}, Flux.MaxPool{2, 4}, Flux.Conv{2, 2,
    typeof(NNlib.relu), Array{Float32, 4}, Vector{Float32}}, Flux.Conv{2, 2,
    typeof(NNlib.relu), Array{Float32, 4}, Vector{Float32}}, Flux.Conv{2, 2,
    typeof(NNlib.relu), Array{Float32, 4}, Vector{Float32}}, Flux.MaxPool{2, 4},
    Flux.Conv{2, 2, typeof(NNlib.relu), Array{Float32, 4}, Vector{Float32}},
    Flux.Conv{2, 2, typeof(NNlib.relu), Array{Float32, 4}, Vector{Float32}},
    Flux.Conv{2, 2, typeof(NNlib.relu), Array{Float32, 4}, Vector{Float32}},
    Flux.MaxPool{2, 4}, Flux.Conv{2, 2, typeof(NNlib.relu), Array{Float32, 4},
    Vector{Float32}}, Flux.Conv{2, 2, typeof(NNlib.relu), Array{Float32, 4},
    Vector{Float32}}, Flux.Conv{2, 2, typeof(NNlib.relu), Array{Float32, 4},
    Vector{Float32}}, Flux.MaxPool{2, 4}}}, Flux.Chain{Tuple{typeof(Flux.flatten),
    Flux.Dense{typeof(NNlib.relu), Matrix{Float32}, Vector{Float32}},
    Flux.Dropout{Float64, Colon}, Flux.Dense{typeof(NNlib.relu), Matrix{Float32},
    Vector{Float32}}, Flux.Dropout{Float64, Colon}, Flux.Dense{typeof(identity),
    Matrix{Float32}, Vector{Float32}}}}}}, ::Vector{Array{Float64,
    3}}) @ interface2.jl:0
13. _pullback @ basic.jl:49 [inlined]
14. _pullback(::Zygote.Context, ::Flux.Chain{Tuple{Flux.Chain{Tuple{Flux.Conv{2, 2,
    typeof(NNlib.relu), Array{Float32, 4}, Vector{Float32}}, Flux.Conv{2, 2,
    typeof(NNlib.relu), Array{Float32, 4}, Vector{Float32}}, Flux.MaxPool{2, 4},
    Flux.Conv{2, 2, typeof(NNlib.relu), Array{Float32, 4}, Vector{Float32}},
    Flux.Conv{2, 2, typeof(NNlib.relu), Array{Float32, 4}, Vector{Float32}},
    Flux.MaxPool{2, 4}, Flux.Conv{2, 2, typeof(NNlib.relu), Array{Float32, 4},
    Vector{Float32}}, Flux.Conv{2, 2, typeof(NNlib.relu), Array{Float32, 4},
    Vector{Float32}}, Flux.Conv{2, 2, typeof(NNlib.relu), Array{Float32, 4},
    Vector{Float32}}, Flux.MaxPool{2, 4}, Flux.Conv{2, 2, typeof(NNlib.relu),
    Array{Float32, 4}, Vector{Float32}}, Flux.Conv{2, 2, typeof(NNlib.relu),
    Array{Float32, 4}, Vector{Float32}}, Flux.Conv{2, 2, typeof(NNlib.relu),
    Array{Float32, 4}, Vector{Float32}}, Flux.MaxPool{2, 4}, Flux.Conv{2, 2,
    typeof(NNlib.relu), Array{Float32, 4}, Vector{Float32}}, Flux.Conv{2, 2,
    typeof(NNlib.relu), Array{Float32, 4}, Vector{Float32}}, Flux.Conv{2, 2,
    typeof(NNlib.relu), Array{Float32, 4}, Vector{Float32}}, Flux.MaxPool{2, 4}}},
    Flux.Chain{Tuple{typeof(Flux.flatten), Flux.Dense{typeof(NNlib.relu),
    Matrix{Float32}, Vector{Float32}}, Flux.Dropout{Float64, Colon},
    Flux.Dense{typeof(NNlib.relu), Matrix{Float32}, Vector{Float32}},
    Flux.Dropout{Float64, Colon}, Flux.Dense{typeof(identity), Matrix{Float32},
    Vector{Float32}}}}}}, ::Vector{Array{Float64, 3}}) @ interface2.jl:0
15. _pullback @ vgg.jl:141 [inlined]
16. _pullback(::Zygote.Context, ::Metalhead.VGG, ::Vector{Array{Float64,
    3}}) @ interface2.jl:0
17. _pullback @ Local: 4 [inlined]
18. _pullback(::Zygote.Context, ::typeof(Main.workspace2.loss), ::Vector{Array{Float64,
    3}}, ::Flux.OneHotArray{UInt32, 2, 1, 2, Vector{UInt32}},
    ::Metalhead.VGG) @ interface2.jl:0
19. _pullback @ Local: 6 [inlined]
20. _pullback(::Zygote.Context, ::Main.workspace2.var"#1#2"{Flux.OneHotArray{UInt32, 2,
    1, 2, Vector{UInt32}}, Vector{Array{Float64, 3}}}) @ interface2.jl:0
21. pullback(::Function, ::Zygote.Params) @ interface.jl:338
22. gradient(::Function, ::Zygote.Params) @ interface.jl:75
23. top-level scope @ Local: 6
```

```julia
begin
    opt = ADAM()
    ps = Flux.params(model)
    loss(x, y, m) = Flux.Losses.logitcrossentropy(m(x), y)
    for (x, y) in train_loader
        gs = gradient(() -> loss(x, y, model), ps)
        Flux.update!(opt, ps, gs)
    end
end
```