Bonus ConvNet MNIST training exploration

```
begin
    using Flux, PlutoUI, Statistics, MLDatasets, Images
    using Flux.Data: DataLoader
    using Flux: @epochs, onehotbatch, onecold, logitcrossentropy, throttle, unsqueeze
    using Random: shuffle, randperm
    using StatsBase: countmap, proportionmap
    using IterTools:ncycle
    using CUDA
end
```

Loading the Dataset

```
0
```

```
CUDA.reclaim()
```

```
(28×28×10000 Array{Float32, 3}:                                                , [7
[:, :, 1] =
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  …  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  …  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 ⋮                         ⋮        ⋱  ⋮                       ⋮
 0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  …  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0

[:, :, 2] =
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …  0.0       0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0       0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0       0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0       0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0       0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …  0.0       0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0       0.0  0.0  0.0  0.0  0.0  0.0
 ⋮                         ⋮           ⋱                          ⋮
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.968627  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.662745  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.458824  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …  0.458824  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.223529  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0       0.0  0.0  0.0  0.0  0.0  0.0

[:, :, 3] =
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 ⋮                         ⋮           ⋱  ⋮                       ⋮
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0

 ...

[:, :, 9998] =
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 ⋮                         ⋮           ⋱  ⋮                       ⋮
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
```

```
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0      0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0      0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0      0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …   0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0      0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 ⋮                        ⋮              ⋱  ⋮                        ⋮
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0      0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0      0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0      0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …   0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0      0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0      0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0

[:, :, 10000] =
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0
 ⋮                        ⋮                 ⋱                        ⋮
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0
```

```
begin
    train_x_org_, train_y_i = MNIST.traindata(Float32);
    test_x_org_, test_y_i = MNIST.testdata(Float32);
end
```

```
28×28×1×10000 CuArray{Float32, 4}:
[:, :, 1, 1] =
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  …  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  …  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 ⋮                        ⋮        ⋱  ⋮                        ⋮
 0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  …  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0

[:, :, 1, 2] =
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …  0.0       0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0       0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0       0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0       0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0       0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …  0.0       0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0       0.0  0.0  0.0  0.0  0.0  0.0
 ⋮                        ⋮              ⋱                        ⋮
```

```
begin
    train_y, test_y = onehotbatch(train_y_i, 0:9)|> gpu, onehotbatch(test_y_i, 0:9)|>
gpu
    train_x_org =  reshape(train_x_org_, 28, 28, 1, :) |> gpu
    test_x_org =  reshape(test_x_org_, 28, 28, 1, :) |> gpu
end
```

Question 1

Defining the Model

```
model =
Chain(Conv((3, 3), 1=>32, relu), Conv((3, 3), 32=>64, relu), MaxPool((2, 2)), Dropout(0.2
```

```
model = Chain(
    Conv((3, 3), 1 => 32, pad=(1, 1), relu),
    Conv((3, 3), 32 => 64, pad=(1, 1), relu),
    MaxPool((2, 2)),
    Dropout(0.25),
    flatten,
    Dense(12544,  128, relu),
    Dropout(0.5),
    Dense(128, 10),
    softmax
) |> gpu
```

Defining the loss and accuracy functions

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

```
function loss(x, y)
    y_predict = model(x)
    return Flux.Losses.crossentropy(y_predict, y)
end
```

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

```
function accuracy(x, y)
    x = Flux.onecold(model(x))
    y = Flux.onecold(y)
    return mean(x .== y)
end
```

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

```
function callBackEvaluation()
    lossVal = loss(test_x_org, test_y)
    accuracyVal = accuracy(test_x_org, test_y)
    @show(lossVal, accuracyVal)
end
```

Defining the Optimizer

```
ADADelta(0.9, IdDict())
```

```
begin
    parameters = Flux.params(model)
    opt = Flux.Optimise.ADADelta()
end
```

Loading Data and Training the Model

```
DataLoader((28×28×1×10000 CuArray{Float32, 4}:
[:, :, 1, 1] =
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  …  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  …  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0
 ⋮                             ⋱  ⋮                             ⋮
 0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  …  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0

[:, :, 1, 2] =
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …  0.0       0.0  0.0  0.0  0.0  0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0       0.0  0.0  0.0  0.0  0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0       0.0  0.0  0.0  0.0  0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0       0.0  0.0  0.0  0.0  0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0       0.0  0.0  0.0  0.0  0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …  0.0       0.0  0.0  0.0  0.0  0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0       0.0  0.0  0.0  0.0  0
 ⋮                             ⋱                             ⋮
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.968627  0.0  0.0  0.0  0.0  0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.662745  0.0  0.0  0.0  0.0  0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.458824  0.0  0.0  0.0  0.0  0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …  0.458824  0.0  0.0  0.0  0.0  0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.223529  0.0  0.0  0.0  0.0  0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0       0.0  0.0  0.0  0.0  0

[:, :, 1, 3] =
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …  0.0  0.0  0.0  0.0  0.0  0.0  0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …  0.0  0.0  0.0  0.0  0.0  0.0  0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0
 ⋮                             ⋱  ⋮                             ⋮
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …  0.0  0.0  0.0  0.0  0.0  0.0  0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0

...

[:, :, 1, 9998] =
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …  0.0  0.0  0.0  0.0  0.0  0.0  0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …  0.0  0.0  0.0  0.0  0.0  0.0  0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0
 ⋮                             ⋱  ⋮                             ⋮
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0
```

```
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0      0.0  0.0  0.0  0.0  0.0  0.0  0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0      0.0  0.0  0.0  0.0  0.0  0.0  0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0      0.0  0.0  0.0  0.0  0.0  0.0  0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …   0.0  0.0  0.0  0.0  0.0  0.0  0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0      0.0  0.0  0.0  0.0  0.0  0.0  0
 ⋮                        ⋮              ⋱   ⋮                        ⋮
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0      0.0  0.0  0.0  0.0  0.0  0.0  0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0      0.0  0.0  0.0  0.0  0.0  0.0  0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0      0.0  0.0  0.0  0.0  0.0  0.0  0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …   0.0  0.0  0.0  0.0  0.0  0.0  0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0      0.0  0.0  0.0  0.0  0.0  0.0  0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0      0.0  0.0  0.0  0.0  0.0  0.0  0

[:, :, 1, 10000] =
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …  0.0  0.0  0.0  0.0  0.0  0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0      0.0  0.0  0.0  0.0  0.0  0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0      0.0  0.0  0.0  0.0  0.0  0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0      0.0  0.0  0.0  0.0  0.0  0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0      0.0  0.0  0.0  0.0  0.0  0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …  0.0  0.0  0.0  0.0  0.0  0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0      0.0  0.0  0.0  0.0  0.0  0
 ⋮                        ⋮                 ⋱                        ⋮
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0      0.0  0.0  0.0  0.0  0.0  0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0      0.0  0.0  0.0  0.0  0.0  0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0      0.0  0.0  0.0  0.0  0.0  0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …  0.0  0.0  0.0  0.0  0.0  0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0      0.0  0.0  0.0  0.0  0.0  0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0      0.0  0.0  0.0  0.0  0.0  0
```

```
begin
    train_loader = Flux.Data.DataLoader((train_x_org, train_y), batchsize=50, shuffle=true)
    test_loader = Flux.Data.DataLoader((test_x_org, test_y), batchsize=50, shuffle=true)
end
```

```
Flux.Optimise.train!(
    loss,
    parameters,
    ncycle(train_loader, 10),
    opt,
    cb=Flux.throttle(callBackEvaluation, 10)
)
```

Model Accuracy

```
0.9877
```

```
accuracy(test_x_org, test_y)
```

Comment:

The accuracy of our model in julia is very close to the provided tensorflow example. the difference of less than .3% in accuracy is negligible.

Question 2

Shuffling the each image pixel

it is worth noting that all images are being shuffled the same exact way. The pix_perm variable is randomized but once it is compiled it is essentially a constant map of where pixels should be that when applied to an image it would do the same kind of computation.

```
pix_perm=reshape(CartesianIndices((28,28))[randperm(28*28)], 28,28);
```

```
28×28×1×60000 CuArray{Float32, 4}:
[:, :, 1, 1] =
 0.0       0.0       0.0       0.992157  …  0.0  0.0       0.0705882  0.0
 0.0       0.0       0.0       0.192157     0.0  0.0       0.0        0.0
 0.0       0.0       0.0       0.0          0.0  0.305882  0.0        0.0
 0.0       0.0       0.992157  0.0          0.0  0.0       0.419608   0.545098
 0.0       0.0       0.0       0.0          0.0  0.0       0.0        0.0
 0.0       0.0       0.0       0.0       …  0.0  0.0       0.0        0.0
 0.0       0.0       0.992157  0.0          0.0  0.0       0.0        0.0
 ⋮                                        ⋱       ⋮
 0.0       0.945098  0.0       0.0          0.0  0.0       0.0        0.0
 0.141176  0.67451   0.0       0.215686     0.0  0.0       0.0        0.0
 0.0       0.0       0.0       0.0          0.0  0.0       0.0        0.0
 0.509804  0.0       0.0       0.0       …  0.0  0.713726  0.0        0.0
 0.0       0.0       0.0       0.0          0.0  0.0       0.0        0.0
 0.0       0.0       0.0       0.0          0.0  0.0       0.0        0.992157

[:, :, 1, 2] =
 0.0       0.0       0.0        0.988235  …  0.0       0.0       0.109804  0.0
 0.0       0.0       0.0980392  0.0          0.0       0.0       0.0       0.0
 0.0       0.0       0.0        0.0          0.0       0.219608  0.0       0.0
 0.0       0.196078  0.988235   0.0          0.0       0.0       0.2       0.913725
 0.0       0.0       0.647059   0.0          0.0       0.439216  0.0       0.0
 0.0       0.0       0.0        0.0       …  0.729412  0.0       0.0       0.0
 0.0       0.0       0.0        0.0          0.0       0.0       0.0       0.988235
 ⋮                                        ⋱            ⋮
```

```
begin
    train_x_ = similar(train_x_org_);
    for i=1:size(train_x_,3)
        train_x_[:,:,i] .= train_x_org_[:,:,i][pix_perm]
    end
```

```
    train_x = reshape(train_x_, 28, 28, 1, :) |> gpu
end
```

```
28×28×1×10000 CuArray{Float32, 4}:
[:, :, 1, 1] =
 0.0       0.0  0.0       0.0      …  0.0       0.0       0.0       0.996078
 0.0       0.0  0.0       0.72549     0.0       0.0       0.0       0.0
 0.0       0.0  0.0       0.0         0.0       0.0       0.992157  0.0
 0.0       0.0  0.776471  0.0         0.0       0.0       0.639216  0.0
 0.0       0.0  0.0       0.0         0.0       0.0       0.0       0.0
 0.0       0.0  0.0       0.0      …  0.0       0.0       0.0       0.0
 0.0       0.0  0.996078  0.0         0.0       0.0       0.0       0.0
 ⋮                                 ⋱            ⋮
 0.898039  0.0  0.0       0.0         0.0       0.0       0.0       0.0
 0.0       0.0  0.0       0.0         0.0       0.0       0.0       0.0
 0.0       0.0  0.0       0.0         0.0       0.0       0.0       0.0
 0.984314  0.0  0.0       0.0      …  0.996078  0.776471  0.0       0.0
 0.0       0.0  0.996078  0.0         0.32549   0.0       0.0       0.0
 0.666667  0.0  0.156863  0.0         0.0       0.0       0.996078  0.858824

[:, :, 1, 2] =
 0.0       0.0       0.0       0.0        …  0.223529  0.0       0.0  0.0
 0.0       0.588235  0.784314  0.301961      0.0       0.0       0.0  0.0
 0.0       0.0       0.203922  0.0           0.0       0.65098   0.0  0.0
 0.0       0.0       0.0       0.0           0.0       0.0       0.0  0.0
 0.0       0.0       0.0       0.0980392     0.0       0.305882  0.0  0.0
 0.0       0.0       0.0       0.0        …  0.0       0.0       0.0  0.0
 0.0       0.0       0.0       0.0           0.0       0.0       0.0  0.0
 ⋮                                         ⋱            ⋮
```

```
begin
    test_x_ = similar(test_x_org_);
    for i=1:size(test_x_,3)
        test_x_[:,:,i] .= test_x_org_[:,:,i][pix_perm]
    end
    test_x = reshape(test_x_, 28, 28, 1, :) |> gpu
end
```

Defining a New model to be trained with the shuffled images

```
model_shuf =
Chain(Conv((3, 3), 1=>32, relu), Conv((3, 3), 32=>64, relu), MaxPool((2, 2)), Dropout(0.2
```

```
model_shuf = Chain(
    Conv((3, 3), 1 => 32, pad=(1, 1), relu),
    Conv((3, 3), 32 => 64, pad=(1, 1), relu),
    MaxPool((2, 2)),
    Dropout(0.25),
    flatten,
    Dense(12544,  128, relu),
    Dropout(0.5),
    Dense(128, 10),
    softmax
) |> gpu
```

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

```
function accuracyShuf(x, y)
    x = Flux.onecold(model_shuf(x))
    y = Flux.onecold(y)
    return mean(x .== y)
end
```

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

```
function lossShuf(x, y)
    y_predict = model_shuf(x)
    return Flux.Losses.crossentropy(y_predict, y)
end
```

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

```
function callBackEvaluationShuf()
    lossVal = lossShuf(test_x, test_y)
    accuracyVal = accuracyShuf(test_x, test_y)
    @show(lossVal, accuracyVal)
end
```

```
ADADelta(0.9, IdDict())
```

```
begin
    parameters_shuf = Flux.params(model_shuf)
    opt_shuf = Flux.Optimise.ADADelta()
end
```

```
DataLoader((28×28×1×10000 CuArray{Float32, 4}:
[:, :, 1, 1] =
 0.0       0.0  0.0       0.0       …  0.0       0.0       0.0       0.99607
 0.0       0.0  0.0       0.72549      0.0       0.0       0.0       0.0
 0.0       0.0  0.0       0.0          0.0       0.0       0.992157  0.0
 0.0       0.0  0.776471  0.0          0.0       0.0       0.639216  0.0
 0.0       0.0  0.0       0.0          0.0       0.0       0.0       0.0
 0.0       0.0  0.0       0.0       …  0.0       0.0       0.0       0.0
 0.0       0.0  0.996078  0.0          0.0       0.0       0.0       0.0
 ⋮                                  ⋱            ⋮
 0.898039  0.0  0.0       0.0          0.0       0.0       0.0       0.0
 0.0       0.0  0.0       0.0          0.0       0.0       0.0       0.0
 0.0       0.0  0.0       0.0          0.0       0.0       0.0       0.0
 0.984314  0.0  0.0       0.0       …  0.996078  0.776471  0.0       0.0
 0.0       0.0  0.996078  0.0          0.32549   0.0       0.0       0.0
 0.666667  0.0  0.156863  0.0          0.0       0.0       0.996078  0.85882

[:, :, 1, 2] =
 0.0       0.0       0.0       0.0         …  0.223529  0.0       0.0  0.0
 0.0       0.588235  0.784314  0.301961        0.0       0.0       0.0  0.0
 0.0       0.0       0.203922  0.0             0.0       0.65098   0.0  0.0
 0.0       0.0       0.0       0.0             0.0       0.0       0.0  0.0
 0.0       0.0       0.0       0.0980392       0.0       0.305882  0.0  0.0
 0.0       0.0       0.0       0.0         …  0.0       0.0       0.0  0.0
 0.0       0.0       0.0       0.0             0.0       0.0       0.0  0.0
 ⋮                                         ⋱            ⋮
 0.0       0.913725  0.992157  0.0             0.0       0.0       0.0  0.0
 0.992157  0.0       0.0       0.0             0.0       0.0       0.0  0.0
 0.0       0.0       0.0       0.0             0.0       0.0       0.0  0.0
 0.0       0.0       0.0       0.0         …  0.992157  0.992157  0.0  0.0
 0.0       0.0       0.0       0.0             0.0       0.0       0.0  0.0
 0.0       0.0       0.0       0.0             0.0       0.0       0.0  0.992

[:, :, 1, 3] =
 0.0  0.0       0.0  0.0       0.0         …  0.0       0.0       0.0       0
 0.0  0.0       0.0  0.0       0.0            0.0       0.0       0.0       0
 0.0  0.0       0.0  0.0       0.0            0.0       0.0       0.0       0
 0.0  0.0       0.0  0.0       0.0            0.0       0.0       0.0       0
 0.0  0.0       0.0  0.0       0.803922       0.0       0.0       0.0       0
 0.0  0.0       0.0  0.592157  0.0         …  0.0       0.0       0.0       0
 0.0  0.0       0.0  0.0       0.0            0.0       0.0       0.0       0
 ⋮                                        ⋱                      ⋮
 0.0  0.223529  0.0  0.0       0.0            0.0       0.0       0.0       0
 0.0  0.0       0.0  0.0       0.321569       0.0       0.0       0.0       0
 0.0  0.0       0.0  0.0       0.0            0.0       0.0       0.0       0
 0.0  0.0       0.0  0.0       0.0         …  0.0       0.792157  0.329412  0
 0.0  0.0       0.0  0.0       0.0            0.0       0.0       0.0       0
 0.0  0.0       0.0  0.0       0.0            0.529412  0.0       0.0       0

...

[:, :, 1, 9998] =
 0.0       0.0       0.0       0.0       …  0.0       0.0       0.0       0
 0.0       0.0       0.14902   0.0          0.0       0.0       0.0       0
 0.0       0.0       0.0       0.0          0.0       0.0       0.690196  0
 0.0       0.0       0.290196  0.0          0.0       0.0       0.847059  0
 0.0       0.0       0.996078  0.054902     0.0       0.901961  0.0       0
 0.0       0.0       0.0       0.690196  …  0.0       0.0       0.0       0
 0.0       0.0       0.996078  0.0          0.0       0.0       0.0       0
 ⋮                                        ⋱            ⋮
 0.996078  0.682353  0.996078  0.0          0.0       0.0       0.0       0
 0.0       0.0       0.0       0.0          0.0       0.482353  0.0       0
```

```
 0.0       0.0       0.0        0.0  0.0          0.0       0.0  0.701961
 0.0       0.996078  0.0392157  0.0  0.0          0.4       0.0  0.0
 0.0       0.0       0.0        0.0  0.996078     0.0       0.0  0.0
 0.0       0.0       0.0        0.0  0.0       …  0.0       0.0  0.0
 0.0       0.0       0.0        0.0  0.0          0.0       0.0  0.0
 ⋮                                              ⋱            ⋮
 0.298039  0.972549  0.996078   0.0  0.996078     0.0       0.0  0.0
 0.0       0.0       0.0        0.0  0.0          0.0       0.0  0.0
 0.0       0.0       0.0        0.0  0.996078     0.0       0.0  0.0
 0.996078  0.32549   0.0        0.0  0.0       …  0.623529  0.0  0.54902
 0.0       0.0       0.0        0.0  0.0          0.866667  0.0  0.0
 0.0       0.0       0.0        0.0  0.0          0.0       0.0  0.0

[:, :, 1, 10000] =
 0.0       0.458824   0.0       0.0       0.0       …  0.0       0.0
 0.0       0.0196078  0.811765  0.0       0.0          0.0       0.0
 0.0       0.0        0.0       0.0       0.0          0.992157  0.0
 0.0       0.0        0.992157  0.0       0.0          0.0       0.294118
 0.0       0.0        0.980392  0.992157  0.992157     0.94902   0.0
 0.0       0.0        0.0       0.992157  0.0       …  0.0       0.0
 0.0       0.992157   0.992157  0.0       0.0          0.0       0.0
 ⋮                                                    ⋱  ⋮
 0.0       0.0        0.25098   0.0       0.0          0.0       0.0
 0.172549  0.0        0.831373  0.0       0.776471     0.992157  0.0196078
 0.0       0.0        0.0       0.0       0.996078     0.0       0.0
 0.337255  0.992157   0.0       0.490196  0.0       …  0.960784  0.0
 0.0       0.0        0.0       0.0       0.0          0.0       0.0
 0.0       0.0        0.0       0.0       0.0          0.0       0.0
```

```
begin
    train_loader_shuf = Flux.Data.DataLoader((train_x, train_y), batchsize=50, shuffle=true)
    test_loader_shuf = Flux.Data.DataLoader((test_x, test_y), batchsize=50, shuffle=true)
end
```

```
Flux.Optimise.train!(
    lossShuf,
    parameters_shuf,
    ncycle(train_loader_shuf, 10),
    opt_shuf,
    cb=Flux.throttle(callBackEvaluationShuf, 10)
)
```

Model Accuracy

```
0.9719
```

```
accuracyShuf(test_x, test_y)
```

Observation and Comments

In this experiment, initialization is very important. when pix_perm compiles it sets a random transofrmation that will be applied to the entire dataset. and it stands to reason that sometimes this transformation is going to be more forgiving than otherwise.

In all of our experiments we have had accuracies in the high ninties. the model trained on the shuffled images was only marginally less accurate than the model trained on the original images.

We know that CNN architecture are sensitice to variability and diversity of pixel intensity within and between local regions. It is a `spacially aware` model. in this case we did shuffle our images but we shuffled them all the same way, still giving the model a stable `pattern` to learn for each class.

Had we shuffled each individual picture randomly (compiled pix_perm in a for loop before every shuffle for example) the accuracy would have suffered greatly.

Question 3

Defining a model with no dropout

```
ADADelta(0.9, IdDict())
```

```
begin
    model_no_dropout = Chain(
        Conv((3, 3), 1 => 32, pad=(1, 1), relu),
        Conv((3, 3), 32 => 64, pad=(1, 1), relu),
        MaxPool((2, 2)),
        flatten,
        Dense(12544,  128, relu),
        Dense(128, 10),
        softmax
    ) |> gpu

    function accuracy_no_dropout(x, y)
        x = Flux.onecold(model_no_dropout(x))
        y = Flux.onecold(y)
        return mean(x .== y)
    end

    function loss_no_dropout(x, y)
        y_predict = model_no_dropout(x)
        return Flux.Losses.crossentropy(y_predict, y)
    end
```

```
    parameters_no_dropout = Flux.params(model_no_dropout)
    opt_no_dropout = Flux.Optimise.ADADelta()
end
```

This is the modek that we made in question 1 (optimal dropout)

```
ADADelta(0.9, IdDict())
```

```
begin
    model_dropout = Chain(
        Conv((3, 3), 1 => 32, pad=(1, 1), relu),
        Conv((3, 3), 32 => 64, pad=(1, 1), relu),
        MaxPool((2, 2)),
        Dropout(0.25),
        flatten,
        Dense(12544, 128, relu),
        Dropout(0.5),
        Dense(128, 10),
        softmax
    ) |> gpu

    function accuracy_dropout(x, y)
        x = Flux.onecold(model_dropout(x))
        y = Flux.onecold(y)
        return mean(x .== y)
    end

    function loss_dropout(x, y)
        y_predict = model_dropout(x)
        return Flux.Losses.crossentropy(y_predict, y)
    end

    parameters_dropout = Flux.params(model_dropout)
    opt_dropout = Flux.Optimise.ADADelta()
end
```

Defining a model with high dropout

```
ADADelta(0.9, IdDict())
```

```
begin
    model_high_dropout = Chain(
        Conv((3, 3), 1 => 32, pad=(1, 1), relu),
        Conv((3, 3), 32 => 64, pad=(1, 1), relu),
        MaxPool((2, 2)),
        Dropout(0.6),
        flatten,
        Dense(12544, 128, relu),
        Dropout(0.8),
        Dense(128, 10),
        softmax
```

```
    ) |> gpu

    function accuracy_high_dropout(x, y)
        x = Flux.onecold(model_dropout(x))
        y = Flux.onecold(y)
        return mean(x .== y)
    end

    function loss_high_dropout(x, y)
        y_predict = model_dropout(x)
        return Flux.Losses.crossentropy(y_predict, y)
    end

    parameters_high_dropout = Flux.params(model_dropout)
    opt_high_dropout = Flux.Optimise.ADADelta()
end
```

customTrainingLoop (generic function with 1 method)

```
function customTrainingLoop(model_cust, trainLoader, test_x, test_y)
    function loss_cust(x, y)
        y_predict = model_cust(x)
        return Flux.Losses.crossentropy(y_predict, y)
    end

    function accuracy_cust(x, y)
        x = Flux.onecold(model_cust(x))
        y = Flux.onecold(y)
        return mean(x .== y)
    end

    opt_cust = Flux.Optimise.ADAM()
    params_cust = Flux.params(model_cust)
    losses = []
    accuracies = []
    for (train_point, test_point) in trainLoader
        grads = gradient(params_cust) do
            loss_cust(train_point, test_point)
        end
        Flux.Optimise.update!(opt_cust, params_cust, grads)
        append!(losses, loss_cust(test_x, test_y))
        append!(accuracies, accuracy_cust(test_x, test_y))
        @show accuracies
        if accuracies[end] > 0.95
            break
        end
    end
    return  losses, accuracies
end
```

We will train the hhree models

Each model will be trained for a single epoch and record the loss and the accuracy for each model

```
([2.26851, 2.20679, 2.13904, 2.06445, 1.97705, 1.87709, 1.75624, 1.61533, 1.48362,
```

```
dropout_losses, dropout_accuracies = customTrainingLoop(model_dropout, train_loader,
```

```
    test_x_org, test_y)
```

```
([2.28677, 2.27866, 2.25622, 2.23074, 2.2038, 2.17881, 2.16029, 2.14272, 2.12386,
```

```
high_dropout_losses, high_dropout_accuracies = customTrainingLoop(model_high_dropout, train_loader, test_x_org, test_y)
```

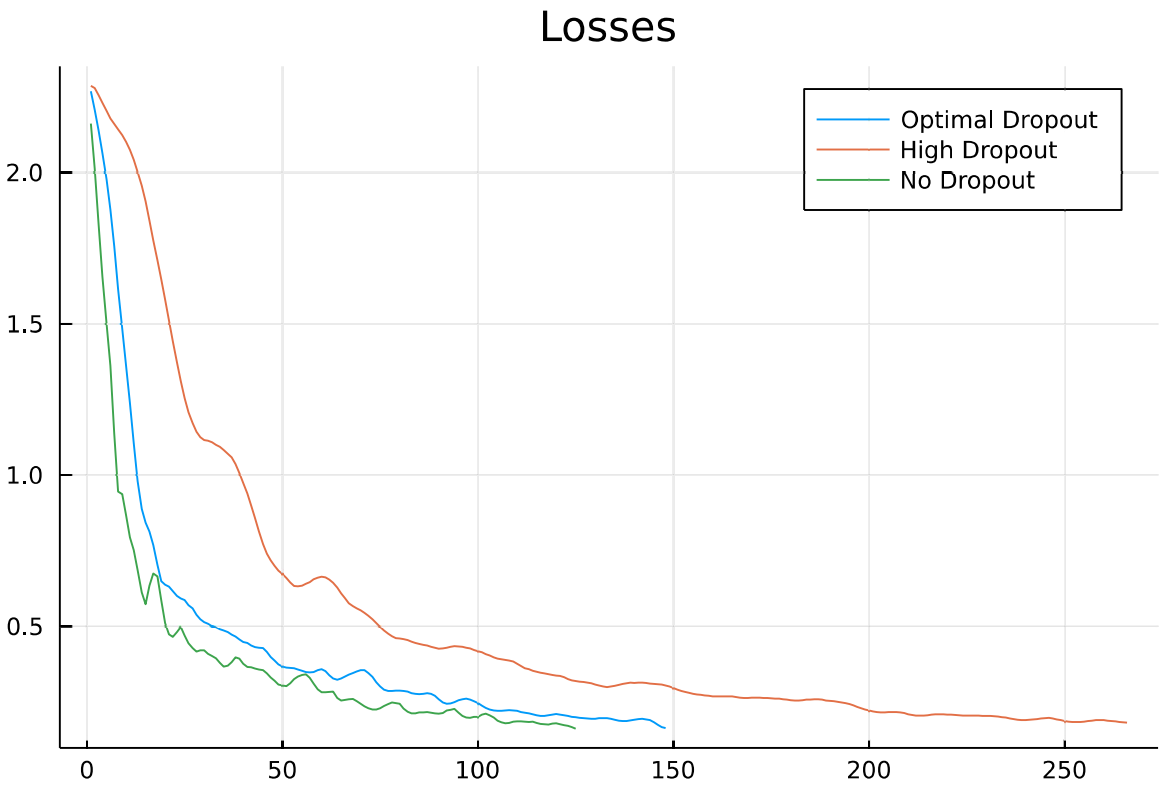
```
([2.16151, 2.01287, 1.83101, 1.65656, 1.50071, 1.36136, 1.13703, 0.945228, 0.93651
```

```
no_dropout_losses, no_dropout_accuracies = customTrainingLoop(model_no_dropout, train_loader, test_x_org, test_y)
```

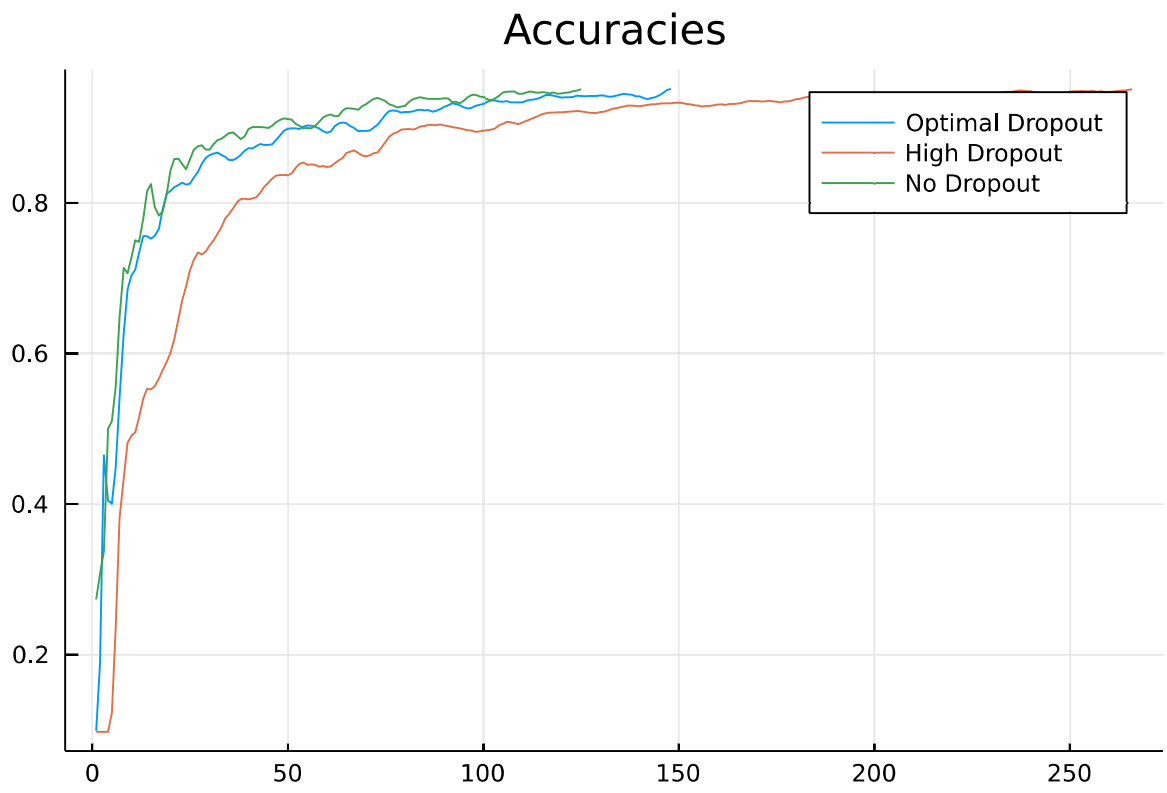
```
using Plots
```

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

```
function list_comp(list_)
    return [x for x in 1:length(list_)]
end
```




```
begin
    plot(list_comp(dropout_losses), dropout_losses, label="Optimal Dropout", title="Losses")
    plot!(list_comp(high_dropout_losses), high_dropout_losses, label="High Dropout")
    plot!(list_comp(no_dropout_losses), no_dropout_losses, label="No Dropout")
end
```




```
begin
    plot(list_comp(dropout_accuracies), dropout_accuracies, label="Optimal Dropout", title="Accuracies")
    plot!(list_comp(high_dropout_accuracies), high_dropout_accuracies, label="High Dropout")
    plot!(list_comp(no_dropout_accuracies), no_dropout_accuracies, label="No Dropout")
end
```

Comments

To truly measure whether our model is over fitting, we would have to withhold a portion of the dataset for validation, some data that the model never sees during training to test the fully trained mode. An Overfit model will perform poorly on the validation data despite having a very high training accuracy. We couldn't acheive this in this experiment due to constraints related to my machine's gpu memory and the fact that Pluto seems to run the whole sheet and overload the memory everytime a change is made.

Therefore we will try to estimate the model's tendency to overfit by how fast the model training accuracy improves. we will use early stoping to compare the losses and accuracies of the three models.

We notice that the model with no dropout at all was the fastest to reach 95% accuracy after just about 120 batches (of 50) of the pictures. This model is likely to overfit during the rest of the training period because of how often it will get exposed to similar data. tje model might suffer from `over-`

`reliance` on a few of its inputs.

We can trust the dropout models to generalize better because they git rid of this `over-reliance`. Since not all of the models inputs are present at all time during training, the model does not become over-reliant on its inputs and as a result can generalize better.

Assignment Questions

Question 1 (10+ pts)

Replicate the Conv net described in **here** at `In [20]` in Julia. Do you get similar results? Comment!

Question 2 (5+ pts)

After shuffeling the pixels around, did the results change? Comment.

Question 2 (5+ pts)

Would the Conv net *overfits* faster or slower without the dropout layers? Demonstrate with results and comment. Get even more points with experimention and analysis of differet dropout rates.