Fashion MNIST training

```
PlotlyBackend()
```

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

```
begin
        using DarkMode
        DarkMode.enable(theme="monokai")
end
```

```
784×10000 CuArray{Float32, 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  0.00784314  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0       0.0  0.0  0.0         0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0       0.0  0.0  0.00392157  0.0  …  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0       0.0  0.0  0.00392157  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 ⋮                                    ⋮   ⋱                 ⋮
 0.0  0.682353  0.0  0.0  0.0         0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.741176  0.0  0.0  0.0         0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.262745  0.0  0.0  0.0         0.0  …  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0       0.0  0.0  0.0         0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0       0.0  0.0  0.0         0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0       0.0  0.0  0.0         0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
```

We can look at few samples. Note that I do `1. .- tensor` to change the background to white

I use the above table to construct a dictionary to make labels to descriptions. Stored in the `class_dic` variable

```
classDict =
Dict(0 ⇒ "T-shirt/top",  4 ⇒ "Coat",  5 ⇒ "Sandal",  6 ⇒ "Shirt",  2 ⇒ "Pullover",
```

Question 1: Defining the model

```
model = Chain(Dense(784, 300, relu), Dense(300, 100, relu), Dense(100, 10), softmax)
```

```
model = Chain(
    Flux.Dense(784, 300, relu),
    Flux.Dense(300, 100, relu),
    Flux.Dense(100, 10),
    softmax
) |> gpu
```

Question 2

Defining Loss Function

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

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

Defining Accuracy Function

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

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

```

Instatiate an ADAM optimizer

```
opt = ADAM(0.001, (0.9, 0.999), IdDict())
```

```
opt = Flux.Optimise.ADAM()
```

Question 3

Defining badIdx Function

We will start by defining a function called getClass; this function is supposed to return the class name of a test datapoint or a test datalabel. We ended up having to define two methods for the same function to handle both test datapoints and labels.

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

```
function getClass(tensorIn, model)
    classIx = Flux.onecold(model(tensorIn), 0:9)
    return classDict[classIx]
end
```

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

```
function getClass(tensorIn::Flux.OneHotArray{UInt32,10,0,1,UInt32})
    classIx = Flux.onecold(tensorIn, 0:9)
    return classDict[classIx]
end
```

We used Our implementation of getClass to construct our badIdx function

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

```
function badIdx(test_x, test_y, model)
    n = size(test_x)[2]
    doesNotMatch = []
    for i in 1:n
        trueClass = getClass(test_y[:,i])
        if getClass(test_x[:,i], model) != trueClass
            push!(doesNotMatch, i)
        end
    end
    return doesNotMatch
end
```

Question 4

Creating the model parameters

```
parameters =
Params([Float32[-0.06551085 0.0715978 … -0.04103929 0.018368045; -0.03336854 0.008918841
```

```
parameters = Flux.params(model)
```

Creating dataloaders for the test and training datasets

```
DataLoader((784×10000 CuArray{Float32, 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  0.00784314   0.0     0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0       0.0  0.0  0.0          0.0     0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0       0.0  0.0  0.00392157   0.0  …  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0       0.0  0.0  0.00392157   0.0     0.0  0.0  0.0  0.0  0.0  0.0
 ⋮                                     ⋮    ⋱                ⋮
 0.0  0.682353  0.0  0.0  0.0          0.0     0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.741176  0.0  0.0  0.0          0.0     0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.262745  0.0  0.0  0.0          0.0  …  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0       0.0  0.0  0.0          0.0     0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0       0.0  0.0  0.0          0.0     0.0  0.0  0.0  0.0  0.0  0.0
 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, train_y), batchsize=50, shuffle=true)
    test_loader = Flux.Data.DataLoader((test_x, test_y), batchsize=50, shuffle=true)
end
```

Defining a callback function, which prints to cunsole the loss and accuracy on the validation dataset when triggered

```
md"Defining a callback function, which prints to cunsole the loss and accuracy on the validation dataset when triggered"
```

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

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

Training the Model and Using ncycle and DataLoader. We specified that the callback funtion is to be triggered every 10 seconds

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

Question 5: Calculating Classes Error Rates

We defined a function called getClassMismatchCount; which uses a given model to classify the validation dataset, and then compares its predictions to the given validation labels.

The function will output the number of misclassified pictures for each class, sorted by from worst to best performing

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

```
function getClassMismatchCount(test_x, test_y, model)
    n = size(test_y)[2]
    classMismatchCount = Dict()
    predictions = Flux.onecold(model(test_x))
    for i in 1:n
        trueClass = getClass(test_y[:,i])
        if classDict[predictions[i]-1] != trueClass
            if haskey(classMismatchCount, trueClass)
                classMismatchCount[trueClass] += 1
                else
                classMismatchCount[trueClass] = 1
            end
        end
    end
    mismatchCountVector = collect(classMismatchCount)
    return sort(mismatchCountVector, by=x -> x[2], rev=true)
end
```

Next we created a function called getClassErrorRates, which uses the output from getClassMismatchCount and devides each class misclassified picture count by the number of datapoints in the class. the function returns the error rate for each class sorted from worst to best performing

```
classCounts =
Dict("T-shirt/top" ⇒ 1000, "Trouser" ⇒ 1000, "Shirt" ⇒ 1000, "Coat" ⇒ 1000, "Sanc
```

```
classCounts = Dict([(classDict[i], count(x -> x == i, test_y_i)) for i in 0:9])
```

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

```
function getClassErrorRates(mismatchCountVector)
    classErrorRates = Dict()
    for pair in mismatchCountVector
        classErrorRates[pair[1]] = pair[2] / classCounts[pair[1]]
    end
    classErrorRatesVector = collect(classErrorRates)
    return sort(classErrorRatesVector, by=x -> x[2], rev=true)
end
```

Finally we defined a function called analyzeMisclassification, this function will tell us for each class and identifies what wrong classes the model had predicted for it, and for each of these wrong classes, the function will calculate the percentage of the respective wrong class compared to the total number of misclassification.

In this way we will understand how the model misclassifies some items of clothing as others.

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

```
function analyzeMisclassification(test_x, test_y, model)
    n = size(test_y)[2]
    classMismatch = Dict()
```

```
    classMismatchCount = Dict()
    predictions = Flux.onecold(model(test_x))
    for i in 1:n
        trueClass = getClass(test_y[:,i])
        predictedClass = classDict[predictions[i]-1]
        if predictedClass != trueClass
            if ~haskey(classMismatch, trueClass)
                classMismatch[trueClass] = Dict()
                classMismatchCount[trueClass] = 1
                else
                classMismatchCount[trueClass] += 1
            end
            if haskey(classMismatch[trueClass], predictedClass)
                classMismatch[trueClass][predictedClass] += 1
                else
                classMismatch[trueClass][predictedClass] = 1
            end
        end
    end
    for (key, _) in classMismatch
        for (key₂, value) in classMismatchCount
            if haskey(classMismatch[key], key₂)
                classMismatch[key][key₂] /= classMismatchCount[key]
            end
        end
        classMismatch[key] = collect(classMismatch[key])
        classMismatch[key] = sort(classMismatch[key], by=x -> x[2], rev=true)
    end
    return classMismatch
end
```

Testing getClass

```
prediction = "Ankle boot"
```

```
prediction = getClass(test_x[:,1], model)
```

```
actual = "Ankle boot"
```

```
actual = getClass(test_y[:,1])
```

Calculating the final accuracy of the MLP network

```
finalAccuracy = 0.1252
```

```
finalAccuracy = accuracy(test_x, test_y)
```

running badIdx on the resuts of the MLP

```
badIndices = Any[
                  1:  13
                  2:  18
                  3:  24
                  4:  26
                  5:  41
                  6:  43
                  7:  44
                  8:  50
                  9:  52
                  10:  58
                  11:  67
                  12:  68
                  13:  69
                  14:  90
                  15:  99
                  16:  128
                  17:  146
                  18:  148
                  19:  151
                  20:  152
                     more
                  1143:  9942
                  1144:  9947
                  1145:  9948
                  1146:  9954
                  1147:  9956
                  1148:  9962
                  1149:  9973
                  1150:  9978
                  1151:  9980
                  1152:  9992
                ]
```

```
badIndices = badIdx(test_x, test_y, model)
```

```
classMisMatchCounts = Pair{Any, Any}[
                                  1:   "Shirt" => 353
                                  2:   "Coat" => 282
                                  3:   "Pullover" => 152
                                  4:   "T-shirt/top" => 108
                                  5:   "Dress" => 71
                                  6:   "Sneaker" => 70
                                  7:   "Trouser" => 39
                                  8:   "Sandal" => 30
                                  9:   "Ankle boot" => 30
                                  10:   "Bag" => 17
                                ]
```

```
classMisMatchCounts = getClassMismatchCount(test_x, test_y, model)
```

Running getClassMismatchCount on the results of the MLP.

We see that the top 3 misclassified classes are in order:

1. Shirt at 35.3% accuracy
2. Coat at 28.2% accuracy
3. Pullover at 15.2% accuracy

```
classMisMatchRates = Pair{Any, Any}[
    "Shirt" ⇒ 0.353
    "Coat" ⇒ 0.282
    "Pullover" ⇒ 0.152
    "T-shirt/top" ⇒ 0.108
    "Dress" ⇒ 0.071
    "Sneaker" ⇒ 0.07
    "Trouser" ⇒ 0.039
    "Sandal" ⇒ 0.03
    "Ankle boot" ⇒ 0.03
     "Bag" ⇒ 0.017
]
```

```
classMisMatchRates = getClassErrorRates(classMisMatchCounts)
```

Question 6: Analyzing Class Error Rates

We ran analyzeMisclassification on the results of the MLP to understand the mistakes that the model made

```
classMisMatchAnalysis =
Dict{Any, Any}(
                    "Sneaker" => Pair{Any, Any}[
                                                 1:   "Ankle boot" => 0.757143
                                                 2:   "Sandal" => 0.242857
                                                ]
                    "Shirt" => Pair{Any, Any}[
                                               1:   "T-shirt/top" => 0.447592
                                               2:   "Pullover" => 0.303116
                                               3:   "Coat" => 0.133144
                                               4:   "Dress" => 0.0736544
                                               5:   "Bag" => 0.0424929
                                              ]
                    "Trouser" => Pair{Any, Any}[
                                                 1:   "Dress" => 0.589744
                                                 2:   "T-shirt/top" => 0.205128
                                                 3:   "Pullover" => 0.0769231
                                                 4:   "Coat" => 0.0512821
                                                 5:   "Bag" => 0.0512821
                                                 6:   "Shirt" => 0.025641
                                                ]
                    "Coat" => Pair{Any, Any}[
                                              1:   "Pullover" => 0.510638
                                              2:   "Shirt" => 0.234043
                                              3:   "Dress" => 0.198582
                                              4:   "Bag" => 0.0531915
                                              5:   "T-shirt/top" => 0.0035461
                                             ]
                    "T-shirt/top" => Pair{Any, Any}[
                                                     1:   "Shirt" => 0.555556
                                                     2:   "Dress" => 0.25
                                                     3:   "Pullover" => 0.0925926
                                                     4:   "Bag" => 0.0740741
                                                     5:   "Sandal" => 0.0185185
                                                     6:   "Coat" => 0.00925926
                                                    ]
                    "Sandal" => Pair{Any, Any}[
                                                1:   "Sneaker" => 0.533333
                                                2:   "Ankle boot" => 0.466667
                                               ]
                    "Bag" => Pair{Any, Any}[
                                             1:   "T-shirt/top" => 0.294118
                                             2:   "Sandal" => 0.235294
                                             3:   "Dress" => 0.235294
                                             4:   "Sneaker" => 0.117647
                                             5:   "Shirt" => 0.0588235
                                             6:   "Coat" => 0.0588235
                                            ]
                    "Dress" => Pair{Any, Any}[
                                               1:   "T-shirt/top" => 0.338028
                                               2:   "Shirt" => 0.211268
                                               3:   "Coat" => 0.183099
                                               4:   "Pullover" => 0.140845
                                               5:   "Bag" => 0.0985915
                                               6:   "Trouser" => 0.028169
                                              ]
                    "Pullover" => Pair{Any, Any}[
                                                  1:   "Coat" => 0.447368
                                                  2:   "Shirt" => 0.276316
                                                  3:   "T-shirt/top" => 0.171053
                                                  4:   "Dress" => 0.0723684
                                                  5:   "Bag" => 0.0328947
                                                 ]
```

```
classMisMatchAnalysis = analyzeMisclassification(test_x, test_y, model)
```

Here we defined some functions to do plots that we need to visualize the model results

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

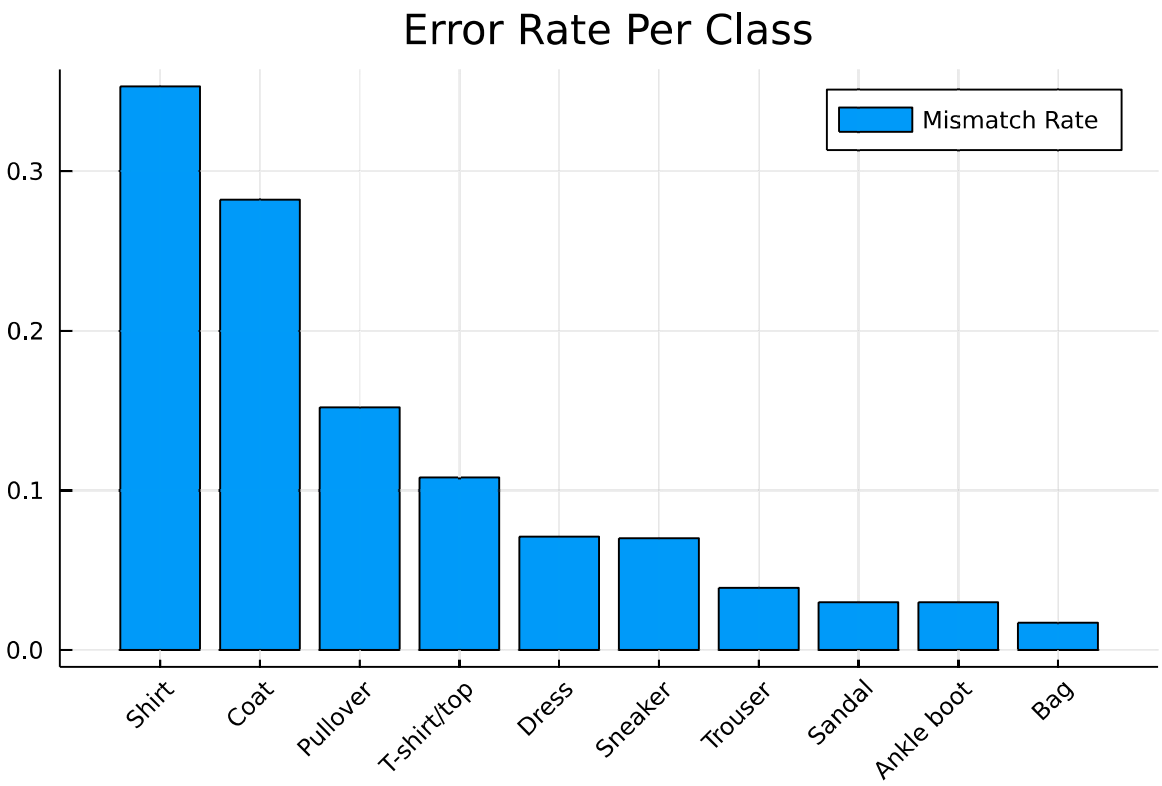
```
begin
    function PlotPair(pairs, label, title)
        names = []
        values= []
        for pair in pairs
            push!(names, pair[1])
            push!(values, pair[2])
        end
        p = bar(values, xticks=(1:10, names), label=label, title=title, xrotation=45)
        return p
    end
end
```

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

```
begin
    function PlotDict(classMisMatchAnalysis)
        nFigures = length(classMisMatchAnalysis)
        l = @layout [grid(ceil(nFigures/2) |> Int, 2)]
        bars = []
        for (key, value) in classMisMatchAnalysis
            names = []
            values= []
            for pair in value
                push!(names, pair[1])
                push!(values, pair[2])
            end
            bar_ = bar(values, xticks=(1:10, names), label="mismatch rate",
title=key, xrotation=45)
            push!(bars, bar_)
        end
        p = plot(bars... ,layout = (ceil(nFigures/2) |> Int, 2), size=(1000,1500))
        return p
    end
end
```

in this plot we visualized the error rate for each class. We notice that Shirt, Coat and T-shirt/top are the top three misclassified classes.

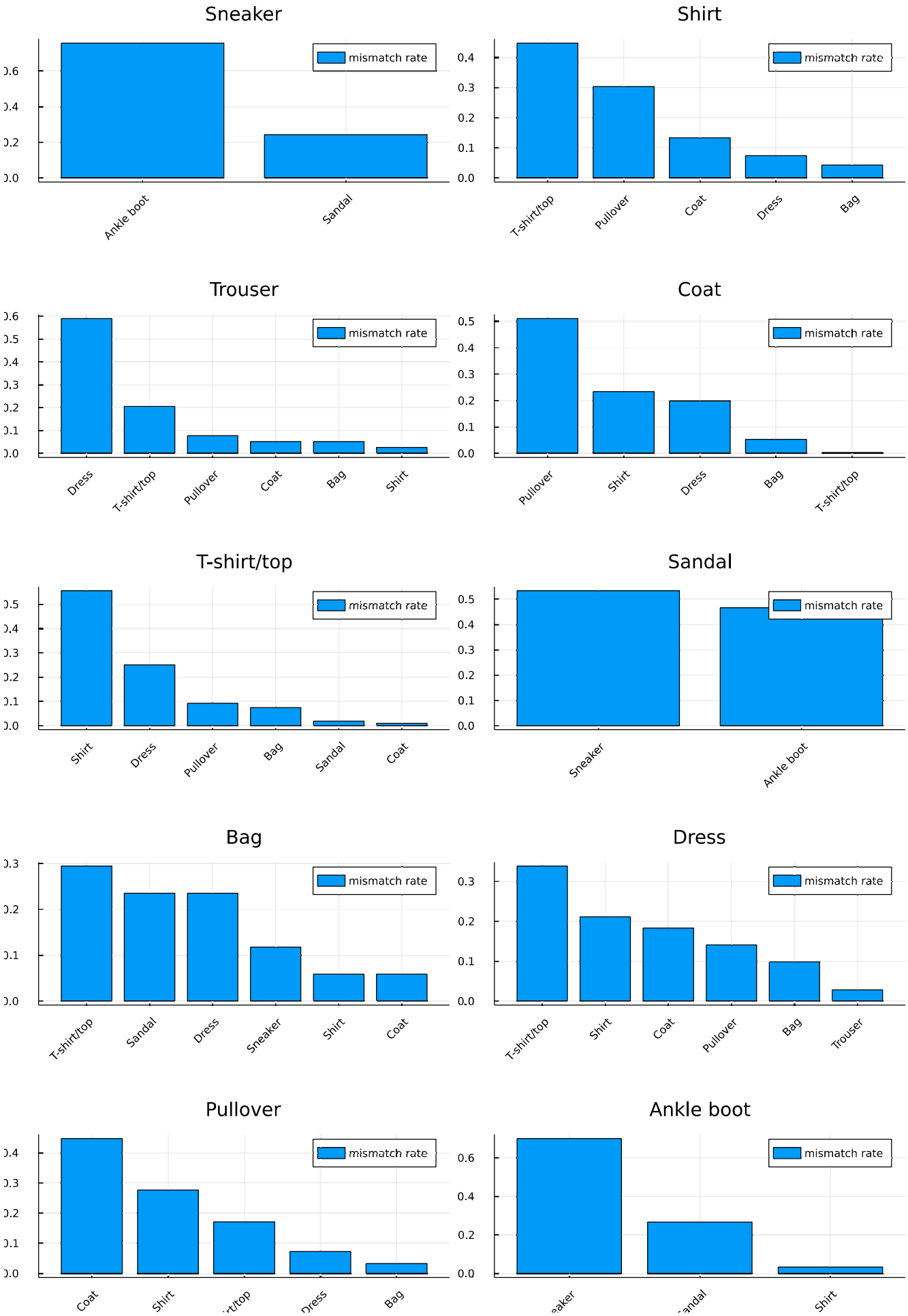
We also notice that articles of clothing that are worn on the upper body e.g. Shirt, Coat, T-shirt/top, pullover and dress, have much higher error rates compared to other classes, suggesting that they are often misclassified as each other. We will analyze that further shortly.




```
PlotPair(classMisMatchRates, "Mismatch Rate", "Error Rate Per Class")
```

In this plot we are looking at each individual class in the validation dataset, and identifying what percentage of its misclassification instances were maped to what wrong class.

For example, We see that the misclassified sneakes, were identified as either sandals or ankle boots by the model.. etc.


Sneaker
mismatch rate
Ankle boot
Sandal
Shirt
mismatch rate
T-shirt/top
Pullover
Coat
Dress
Bag
Trouser
mismatch rate
Dress
T-shirt/top
Pullover
Coat
Bag
Shirt
Coat
mismatch rate
Pullover
Shirt
Dress
Bag
T-shirt/top
T-shirt/top
mismatch rate
Shirt
Dress
Pullover
Bag
Sandal
Coat
Sandal
mismatch rate
Sneaker
Ankle boot
Bag
mismatch rate
T-shirt/top
Sandal
Dress
Sneaker
Shirt
Coat
Dress
mismatch rate
T-shirt/top
Shirt
Coat
Pullover
Bag
Trouser
Pullover
mismatch rate
Coat
Shirt
Dress
Bag
Ankle boot
mismatch rate
Shirt

```
PlotDict(classMisMatchAnalysis)
```

Question 6 (repeated): Building a Custom Training Loop

```
md"# Question 6 (repeated): Building a Custom Training Loop"
```

We made an identical MLP model to train with the custom training loop

```
model_cust =
Chain(Dense(784, 300, relu), Dense(300, 100, relu), Dense(100, 10), softmax)
```

```
model_cust = Chain(
    Flux.Dense(784, 300, relu), # Flattened input Image -> 300 Nodes Hidden Layer 1
    Flux.Dense(300, 100, relu), # 300 Nodes Hidden Layer 1-> 100 Nodes Hidden Layer 2
    Flux.Dense(100, 10), # 100 Nodes Hidden Layer 2-> 10 Nodes Output
    softmax # Softmax for Classification
) |> gpu
```

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

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

Defining the custom training loop. This loop will train a model for 10 epochs and will output the validation loss and the accuracy after each epoch.

```
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

    opt_cust = Flux.Optimise.ADAM()
    params_cust = Flux.params(model_cust)

    for _ in 1:10
        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)
        end
        @show loss_cust(test_x, test_y) , accuracy_cust(test_x, test_y, model_cust)
    end
end
```

training the second MLP with the custom training loop and getting its final accuracy

```
customTrainingLoop(model_cust, train_loader, test_x, test_y)
```

```
0.8839
```

```
accuracy_cust(test_x, test_y, model_cust)
```

Question 7: Creating a CNN Model

For the CNN model we have to reshape the dataset before we can start training the model

```
train_x_conv =
28×28×1×60000 CuArray{Float32, 4}:
[:, :, 1, 1] =
 0.0  0.0  0.0  0.0         0.0         …  0.0       0.00784314  0.0  0.0  0.0
 0.0  0.0  0.0  0.0         0.0            0.0       0.0         0.0  0.0  0.0
 0.0  0.0  0.0  0.0         0.0            0.290196  0.0         0.0  0.0  0.0
 0.0  0.0  0.0  0.0         0.0            0.741176  0.0         0.0  0.0  0.0
 0.0  0.0  0.0  0.0         0.0            0.831373  0.258824    0.0  0.0  0.0
 0.0  0.0  0.0  0.0         0.0         …  0.74902   0.784314    0.0  0.0  0.0
 0.0  0.0  0.0  0.0         0.0            0.686275  0.870588    0.0  0.0  0.0
 ⋮                                      ⋱                       ⋮
 0.0  0.0  0.0  0.0         0.00392157     0.737255  0.65098     0.0  0.0  0.0
 0.0  0.0  0.0  0.0         0.0117647      0.760784  0.658824    0.0  0.0  0.0
 0.0  0.0  0.0  0.0         0.0156863      0.752941  0.388235    0.0  0.0  0.0
 0.0  0.0  0.0  0.00392157  0.0         …  0.847059  0.227451    0.0  0.0  0.0
 0.0  0.0  0.0  0.00392157  0.0            0.666667  0.0         0.0  0.0  0.0
 0.0  0.0  0.0  0.0         0.0117647      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         0.00392157  0.0            0.0         0.0  0.0         0.0
 0.0         0.0         0.0            0.0         0.0  0.0         0.0
 0.00392157  0.0         0.054902    …  0.00392157  0.0  0.0         0.00392157
 0.0         0.0         0.690196       0.0         0.0  0.0         0.0
```

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

```
train_loader_conv =
DataLoader((28×28×1×60000 CuArray{Float32, 4}:
[:, :, 1, 1] =
 0.0  0.0  0.0  0.0         0.0         …  0.0       0.00784314  0.0  0.0
 0.0  0.0  0.0  0.0         0.0            0.0       0.0         0.0  0.0
 0.0  0.0  0.0  0.0         0.0            0.290196  0.0         0.0  0.0
 0.0  0.0  0.0  0.0         0.0            0.741176  0.0         0.0  0.0
 0.0  0.0  0.0  0.0         0.0            0.831373  0.258824    0.0  0.0
 0.0  0.0  0.0  0.0         0.0         …  0.74902   0.784314    0.0  0.0
 0.0  0.0  0.0  0.0         0.0            0.686275  0.870588    0.0  0.0
 ⋮                                      ⋱                       ⋮
 0.0  0.0  0.0  0.0         0.00392157     0.737255  0.65098     0.0  0.0
 0.0  0.0  0.0  0.0         0.0117647      0.760784  0.658824    0.0  0.0
 0.0  0.0  0.0  0.0         0.0156863      0.752941  0.388235    0.0  0.0
 0.0  0.0  0.0  0.00392157  0.0         …  0.847059  0.227451    0.0  0.0
 0.0  0.0  0.0  0.00392157  0.0            0.666667  0.0         0.0  0.0
 0.0  0.0  0.0  0.0         0.0117647      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         0.00392157  0.0          0.0         0.0  0.0         0.0
 0.0         0.0         0.0          0.0         0.0  0.0         0.0
 0.00392157  0.0         0.054902  …  0.00392157  0.0  0.0         0.003921
 0.0         0.0         0.690196     0.0         0.0  0.0         0.0
 ⋮                                 ⋱              ⋮
 0.0         0.176471    0.878431     0.0         0.0  0.0         0.0
 0.0         0.0         0.643137     0.00392157  0.0  0.00784314  0.0
 0.0         0.0         0.0          0.0         0.0  0.0         0.0
 0.0         0.0         0.0       …  0.0         0.0  0.0         0.0
 0.0         0.0         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, 59998] =
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …  0.0  0.0  0.0  0.0  0.0  0.0  0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …  0.0  0.0  0.0  0.0  0.0  0.0  0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0
 ⋮                             ⋮          ⋱  ⋮                       ⋮
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0
```

```
0.0  0.0  0.0        0.0        0.0          0.0  0.0  0.0  0.0         0.
0.0  0.0  0.0        0.0        0.0          0.0  0.0  0.0  0.0         0.
0.0  0.0  0.0        0.0        0.0          0.0  0.0  0.0  0.0         0.
0.0  0.0  0.0        0.0509804  0.270588     0.0  0.0  0.0  0.0         0.
0.0  0.0  0.0392157  0.376471   0.27451   …  0.0  0.0  0.0  0.0         0.
0.0  0.0  0.32549    0.270588   0.196078     0.0  0.0  0.0  0.0666667   0.
⋮                                        ⋱            ⋮
0.0  0.0  0.0431373  0.407843   0.290196     0.0  0.0  0.0  0.0         0.
0.0  0.0  0.0        0.0509804  0.305882     0.0  0.0  0.0  0.00392157  0.
0.0  0.0  0.0        0.0        0.0          0.0  0.0  0.0  0.0         0.
0.0  0.0  0.0        0.0        0.0       …  0.0  0.0  0.0  0.0         0.
0.0  0.0  0.0        0.0        0.0          0.0  0.0  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, 60000] =
0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …  0.0  0.0  0.0  0.0  0.0  0.0  0
0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0
0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0
0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0
0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0
0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …  0.0  0.0  0.0  0.0  0.0  0.0  0
0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0
⋮                        ⋮              ⋱  ⋮                        ⋮
0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0
0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0
0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0
0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …  0.0  0.0  0.0  0.0  0.0  0.0  0
0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0
0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0
```

```
train_loader_conv = Flux.Data.DataLoader((train_x_conv, train_y), batchsize=50, shuffle=true)
```

```
test_x_conv =
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     0.0470588  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.262745   0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  …  0.415686   0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.643137   0.0  0.0  0.0  0.0  0.0  0.0
 ⋮                       ⋮        ⋱                          ⋮
 0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.737255   0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.686275   0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.635294   0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  …  0.619608   0.0  0.0  0.0  0.0  0.0  0.0
 0.0  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     0.0431373  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           0.737255  0.509804  0.298039  0.192157
 0.0  0.0  0.0       0.4       0.913725   …  0.941176  0.960784  1.0       0.803922
 0.0  0.0  0.537255  1.0       0.945098      0.980392  0.94902   0.976471  0.772549
```

```
test_x_conv = reshape(test_x_, 28, 28, 1, :)  |> gpu
```

Here we defined a CNN model as described in the assignment questions

```
model_conv =
Chain(Conv((5, 5), 1=>6, relu), MaxPool((2, 2)), Conv((5, 5), 6=>16, relu), MaxPool((2, 2
```

```
model_conv = Chain(
    Conv((5, 5), 1=>6, pad=SamePad(), relu),
    MaxPool((2, 2)),
    Conv((5, 5), 6=>16, pad=SamePad(), relu),
    MaxPool((2, 2)),
    flatten,
    Dense(784, 10),
    softmax,
) |> gpu
```

We trained the CNN model with our custom training loop. And tested its final accuracy

```
customTrainingLoop(model_conv, train_loader_conv, test_x_conv, test_y)
```

```
0.9
```

```
accuracy_cust(test_x_conv, test_y, model_conv)
```

Question 8: Improving the CNN Model Through Architectural Changes

```
model_conv_2 =
Chain(Conv((3, 3), 1=>16, relu), MaxPool((2, 2)), Conv((3, 3), 16=>32, relu), MaxPool((2,
```

```
model_conv_2 = Chain(
    # First convolution, operating upon a 28x28 image
    Conv((3, 3), 1 => 16, pad=(1, 1), relu),
    MaxPool((2, 2)),

    # Second convolution, operating upon a 14x14 image
    Conv((3, 3), 16 => 32, pad=(1, 1), relu),
    MaxPool((2, 2)),

    # Third convolution, operating upon a 7x7 image
    Conv((3, 3), 32 => 32, pad=(1, 1), relu),
    MaxPool((2, 2)),

    # 'Flux.flatten' will make the pictures (3, 3, 32, N)
    flatten,
    Dense(288,  128, relu),
    Dense(128, 64, relu),
    Dense(64, 10),
    softmax
    ) |> gpu
```

We traind the new CNN model with our custom trainig loop

```
customTrainingLoop(model_conv_2, train_loader_conv, test_x_conv, test_y)
```

We can see that this model performs better at more than 90% accuracy

```
0.9122
```

```
accuracy_cust(test_x_conv, test_y, model_conv_2)
```

We will do the same analysis we did on the results of the first MLP model

```
classMisMatchCountsConv = Pair{Any, Any}[
                                          1:   "Shirt" => 250
                                          2:   "T-shirt/top" => 157
                                          3:   "Coat" => 141
                                          4:   "Pullover" => 126
                                          5:   "Dress" => 83
                                          6:   "Ankle boot" => 45
                                          7:   "Sneaker" => 23
                                          8:   "Trouser" => 18
                                          9:   "Sandal" => 18
                                          10:   "Bag" => 17
                                        ]
```

```
classMisMatchCountsConv = getClassMismatchCount(test_x_conv, test_y, model_conv_2)
```

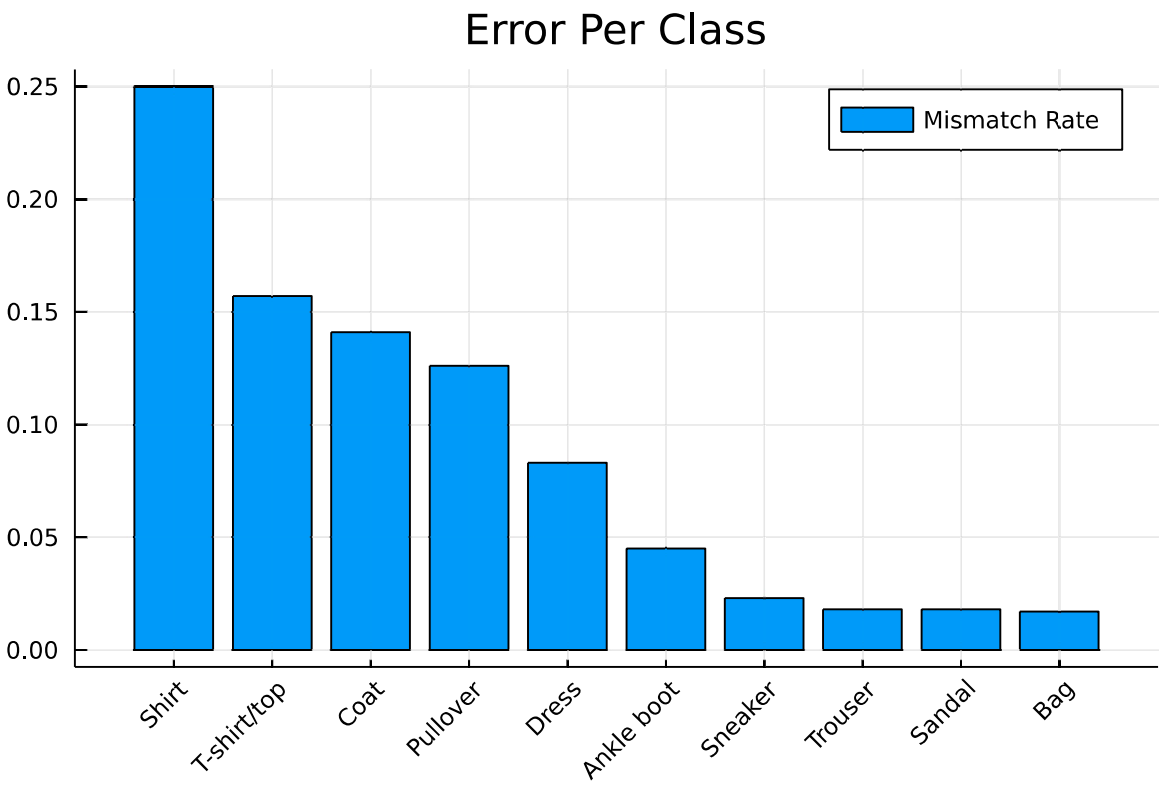
We again see that the top misclassified classes are all upperbody clothing items. however the top three misclassified items here are different:

1. Shirt at 25% error Rate.

2. T-shirt/top at 15.7% error Rate.
3. Coat 14.1% error Rate.

```
classMisMatchRatesConv = Pair{Any, Any}[
    "Shirt" ⇒ 0.25
    "T-shirt/top" ⇒ 0.157
    "Coat" ⇒ 0.141
    "Pullover" ⇒ 0.126
    "Dress" ⇒ 0.083
    "Ankle boot" ⇒ 0.045
    "Sneaker" ⇒ 0.023
    "Trouser" ⇒ 0.018
    "Sandal" ⇒ 0.018
     "Bag" ⇒ 0.017
]
```

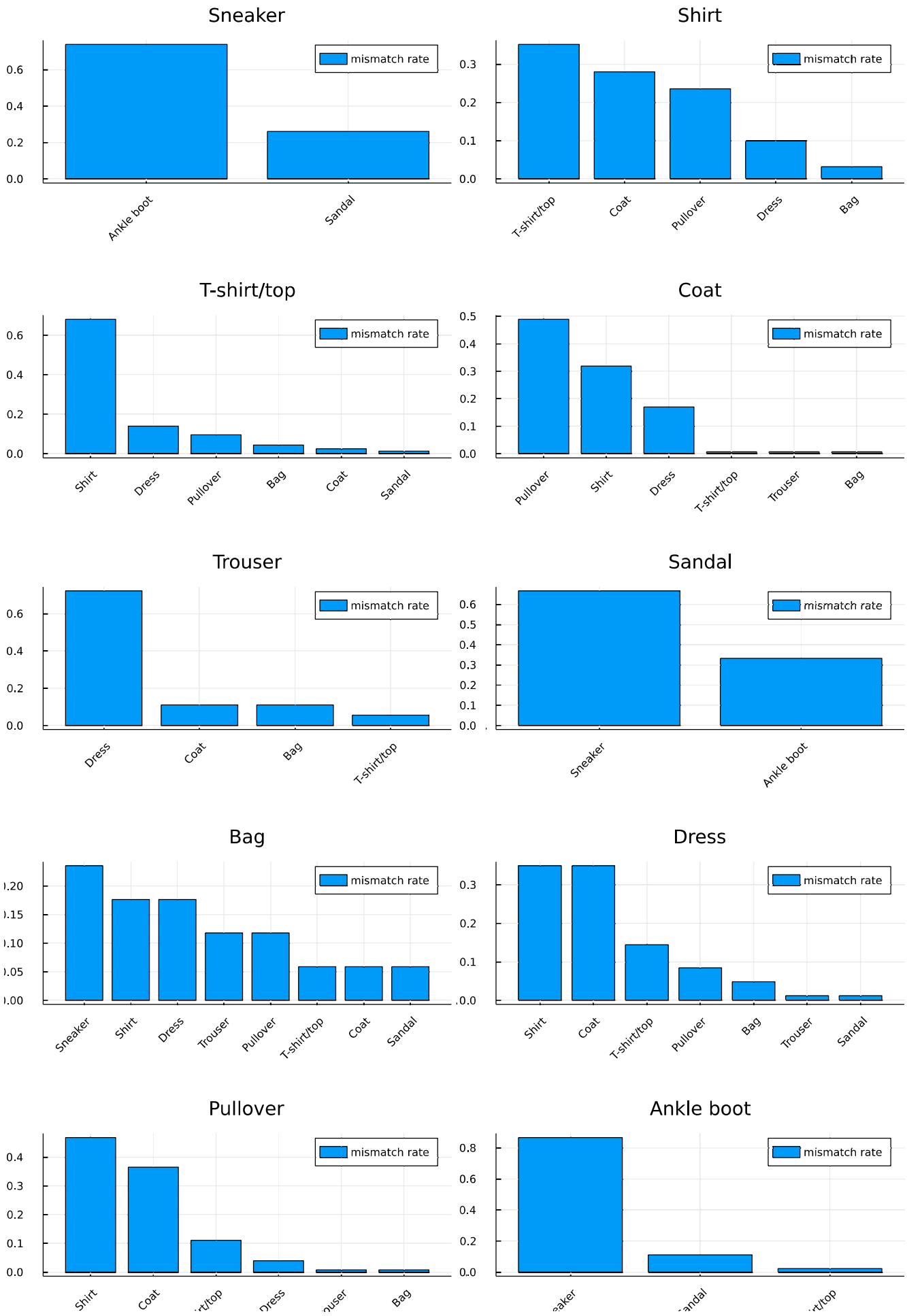
```
classMisMatchRatesConv = getClassErrorRates(classMisMatchCountsConv)
```




```
PlotPair(classMisMatchRatesConv, "Mismatch Rate", "Error Per Class")
```

```
classErrorAnalysisConv2 =
Dict{Any, Any}(
                    "Sneaker" => Pair{Any, Any}[
                                                 1:   "Ankle boot" => 0.73913
                                                 2:   "Sandal" => 0.26087
                                                ]
                    "Shirt" => Pair{Any, Any}[
                                               1:   "T-shirt/top" => 0.352
                                               2:   "Coat" => 0.28
                                               3:   "Pullover" => 0.236
                                               4:   "Dress" => 0.1
                                               5:   "Bag" => 0.032
                                              ]
                    "T-shirt/top" => Pair{Any, Any}[
                                                     1:   "Shirt" => 0.681529
                                                     2:   "Dress" => 0.140127
                                                     3:   "Pullover" => 0.0955414
                                                     4:   "Bag" => 0.044586
                                                     5:   "Coat" => 0.0254777
                                                     6:   "Sandal" => 0.0127389
                                                    ]
                    "Coat" => Pair{Any, Any}[
                                              1:   "Pullover" => 0.489362
                                              2:   "Shirt" => 0.319149
                                              3:   "Dress" => 0.170213
                                              4:   "T-shirt/top" => 0.0070922
                                              5:   "Trouser" => 0.0070922
                                              6:   "Bag" => 0.0070922
                                             ]
                    "Trouser" => Pair{Any, Any}[
                                                 1:   "Dress" => 0.722222
                                                 2:   "Coat" => 0.111111
                                                 3:   "Bag" => 0.111111
                                                 4:   "T-shirt/top" => 0.0555556
                                                ]
                    "Sandal" => Pair{Any, Any}[
                                                1:   "Sneaker" => 0.666667
                                                2:   "Ankle boot" => 0.333333
                                               ]
                    "Bag" => Pair{Any, Any}[
                                             1:   "Sneaker" => 0.235294
                                             2:   "Shirt" => 0.176471
                                             3:   "Dress" => 0.176471
                                             4:   "Trouser" => 0.117647
                                             5:   "Pullover" => 0.117647
                                             6:   "T-shirt/top" => 0.0588235
                                             7:   "Coat" => 0.0588235
                                             8:   "Sandal" => 0.0588235
                                            ]
                    "Dress" => Pair{Any, Any}[
                                               1:   "Shirt" => 0.349398
                                               2:   "Coat" => 0.349398
                                               3:   "T-shirt/top" => 0.144578
                                               4:   "Pullover" => 0.0843373
                                               5:   "Bag" => 0.0481928
                                               6:   "Trouser" => 0.0120482
                                               7:   "Sandal" => 0.0120482
                                              ]
                    "Pullover" => Pair{Any, Any}[
                                                  1:   "Shirt" => 0.468254
                                                  2:   "Coat" => 0.365079
                                                  3:   "T-shirt/top" => 0.111111
```

```
classErrorAnalysisConv2 = analyzeMisclassification(test_x_conv, test_y, model_conv_2)
                )
```


Sneaker
mismatch rate
Ankle boot
Sandal
Shirt
mismatch rate
T-shirt/top
Coat
Pullover
Dress
Bag
T-shirt/top
mismatch rate
Shirt
Dress
Pullover
Bag
Coat
Sandal
Coat
mismatch rate
Pullover
Shirt
Dress
T-shirt/top
Trouser
Bag
Trouser
mismatch rate
Dress
Coat
Bag
T-shirt/top
Sandal
mismatch rate
Sneaker
Ankle boot
Bag
mismatch rate
Sneaker
Shirt
Dress
Trouser
Pullover
T-shirt/top
Coat
Sandal
Dress
mismatch rate
Shirt
Coat
T-shirt/top
Pullover
Bag
Trouser
Sandal
Pullover
mismatch rate
Shirt
Coat
Bag
Ankle boot
mismatch rate

```
PlotDict(classErrorAnalysisConv2)
```

Question 9

yes, the top 3 mmisclassified claasses changed across the models we trained, they also changed if we trained the same model

Assignment Questions

Question 1 (4 pts)

Using Flux `Chain` build a Dense MLP NN, with following layers in sequence 784, 300, 100, 10. Run the output of the lastad layer through a `softmax` function.

Question 2 (2 pts)

a) Define a loss funtion based on `crossentropty`

b) Instatiate an `ADAM` optimizer

c) Define an `accuracy` function that could runn over the whole dataset (Hint: make use of `onecold` and `mean`)

Question 3 (3 pts)

Define a funciton `badIdx` that filter for the indices where the model failes to classify correctly.

Question 4 (3 pts)

Make use of `nycle` and `DataLoader` and `Flux.train!` to train the model with `batchsize` of 50 and 10 epochs (Hint: see). Use a callback function, `cb` to the `accuracy` and `loss` every 10 seconds.

Question 5 (4 pts)

Which top 3 classes did the Dense MLP NN model struggle with the most? Are ther error rates uniform across the classes? If the repeat the training, it the top three misclassified classes

3×3×20×1

Question 6 (4 pts)

Which top 3 classes did the Dense MLP NN model struggle with the most? Are ther error rates uniform across the classes? If the repeat the training, it the top three misclassified classes

Question 6 (3 pts)

Build you own *custom training loop* with same parameters. Check that it works just like in question 4.

Question 7 (5 pts)

Construct a convolutional model with following architecture

```
1. Conv with a (5,5) kernel mapping to **6** feature maps, with `relu`,  same paddi
g
2. 2x2 Max pool
3. Conv with a (5,5) kernel mapping to **16** feature maps, with `relu`,  same padd
ig
4. 2x2 Max pool
5. Appropriatly sized Dense layer with 10 outputs
6. A `softmax` layer
```

Do the training using a *custom training loop*

Question 8 (3 pts)

Do changes in the convnet architecture to get a better accuracy.

Question 9 (3 pts)

Did the top 3 classes in Question 6 change with the use of the networks in Questions 8 or 9?