Exercise 2: Create a network from scratch and train

Create new deep networks for image classification and regression tasks by defining the network architecture and training the network from scratch.

After defining the network architecture, you can define training parameters using the `trainingOptions` function. You can then train the network using the `trainnet` function. Use the trained network to predict class labels or numeric responses. If the `trainingOptions` function does not provide the training options that you need for your task, or custom output layers do not support the loss functions that you need, then you can define a custom training loop.

Design network layers

Create new deep networks for tasks such as image classification and regression by defining the network architecture from scratch. Build networks using MATLAB or interactively using Deep Network Designer.

For most tasks, you can use built-in layers. If there is not a built-in layer that you need for your task, then you can define your own custom layer. You can define custom layers with learnable and state parameters.

```
layers = [
    imageInputLayer([8 8 1])
    %featureInputLayer(1,Normalization="zscore")

    convolution2dLayer(3,8,Padding="same")
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer(2,Stride=2)

    convolution2dLayer(3,16,Padding="same")
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer(2,Stride=2)

    convolution2dLayer(3,32,Padding="same")
    batchNormalizationLayer
    reluLayer

    fullyConnectedLayer(4)
    softmaxLayer];
```

Split data into training and testing portions

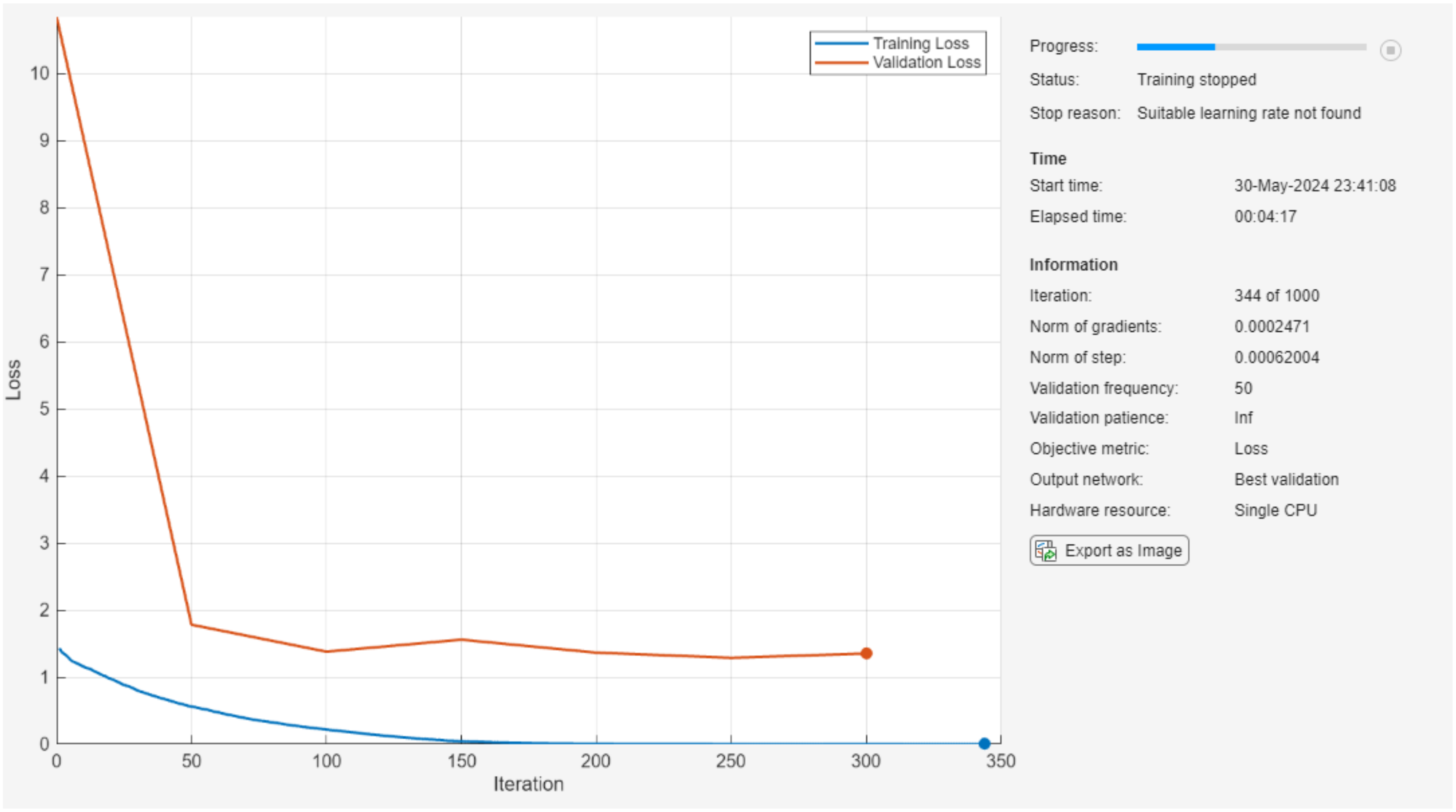
```
%% set up the dataset entries in workspace
[xTrain, label_Train, xTest, label_Test] = trainWithTabularData(dataTable);
```

Set up training options

```
% Start with simple options
options = trainingOptions("lbfgs", ...
    ExecutionEnvironment="cpu", ...
    Plots="training-progress", ...
    ValidationData={xTest, label_Test}, ...
    Verbose=false);
```

Train and validate easily

```
%% train one network
% Use augmented dataset for training
net = trainnet(augimds,layers,"crossentropy",options);
```



Progress:
Status: Training stopped
Stop reason: Suitable learning rate not found

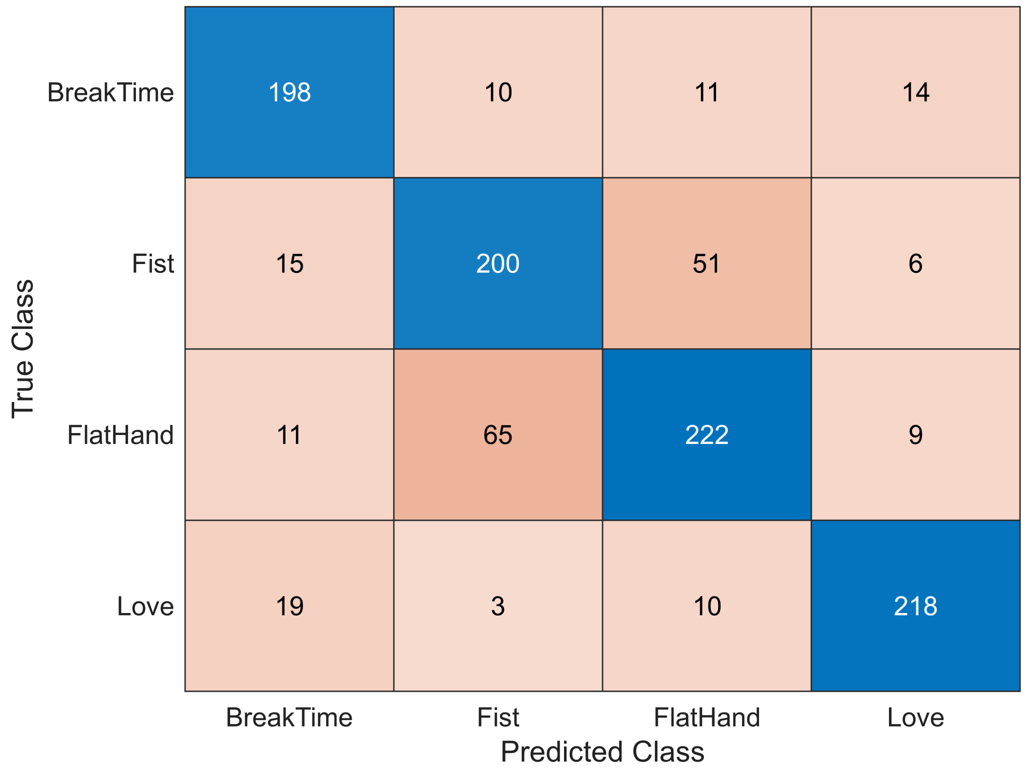
Time
Start time: 30-May-2024 23:41:08
Elapsed time: 00:04:17

Information
Iteration: 344 of 1000
Norm of gradients: 0.0002471
Norm of step: 0.00062004
Validation frequency: 50
Validation patience: Inf
Objective metric: Loss
Output network: Best validation
Hardware resource: Single CPU

```
save("ex2TrainedNet.mat", "net");
```

Test the network inference

```
%% test the trained network
scoresTest = minibatchpredict(net,xTest);
YTest = onehotdecode(scoresTest,classNames,2);
confusionchart(label_Test,YTest);
```

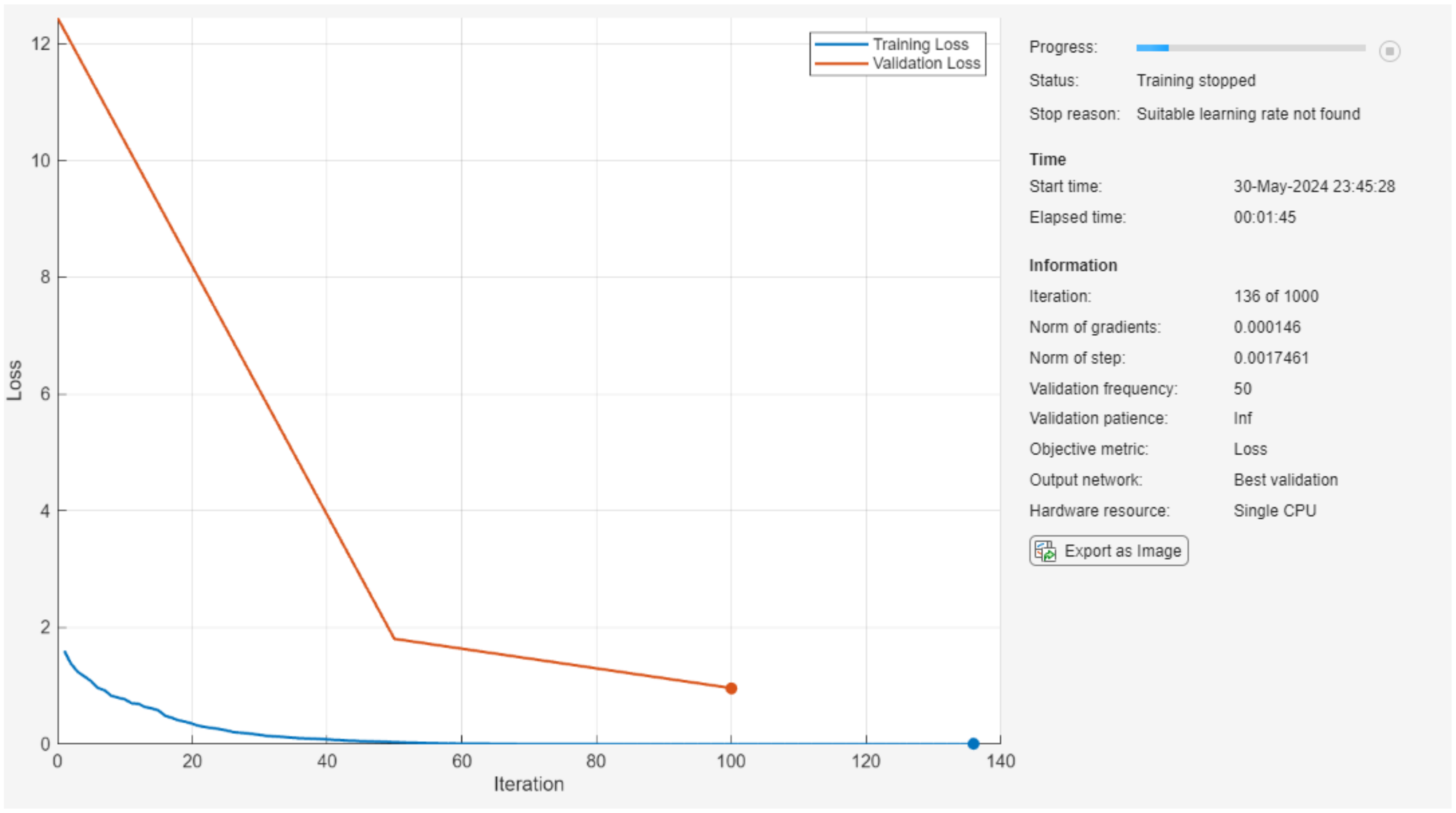


```
accuracy = mean(YTest == label_Test)
```

```
accuracy = 0.7891
```

Topics to consider: What if we trained with the original data, without augmentation?

```
% Can also use raw data to train
net_raw = trainnet(xTrain,label_Train,layers,"crossentropy",options);
```



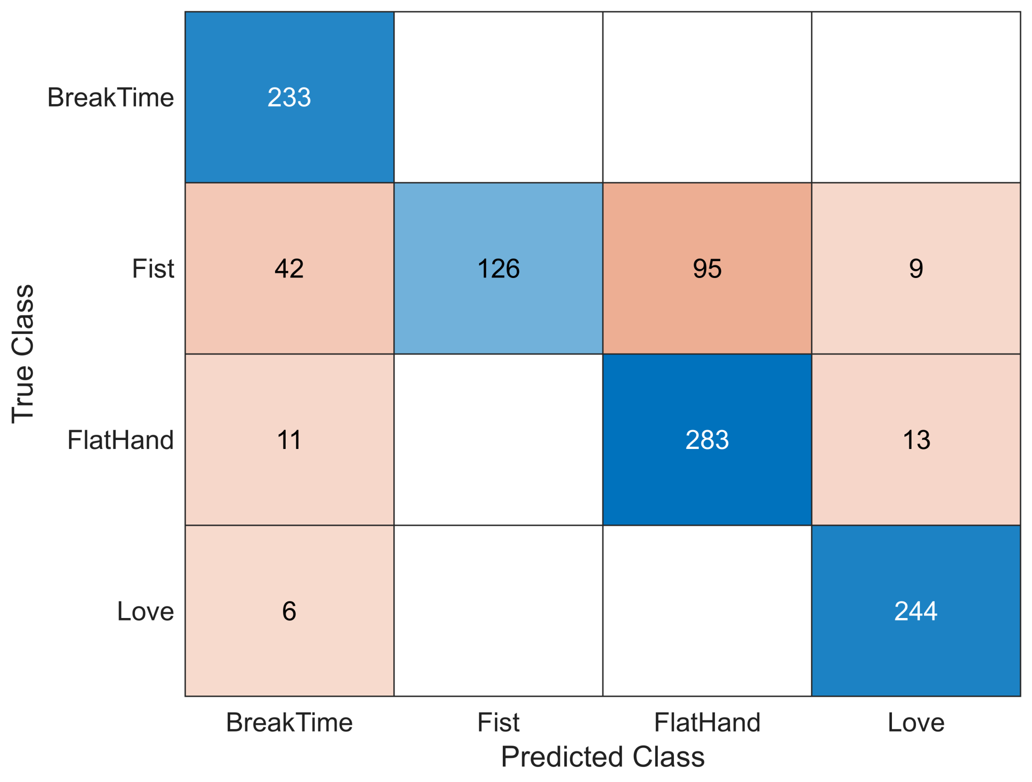
Progress:
Status: Training stopped
Stop reason: Suitable learning rate not found

Time
Start time: 30-May-2024 23:45:28
Elapsed time: 00:01:45

Information
Iteration: 136 of 1000
Norm of gradients: 0.000146
Norm of step: 0.0017461
Validation frequency: 50
Validation patience: Inf
Objective metric: Loss
Output network: Best validation
Hardware resource: Single CPU

Same training options, but we can get different performance behaviors.

```
%% test the trained network
scoresTest = minibatchpredict(net_raw,xTest);
YTest = onehotdecode(scoresTest,classNames,2);
confusionchart(label_Test,YTest);
```



```
accuracy = mean(YTest == label_Test)
```

```
accuracy = 0.8343
```