```
classdef TopoDatastore < matlab.io.Datastore & ...
        matlab.io.datastore.MiniBatchable & ...
        matlab.io.datastore.Shuffleable
    %TOPODATASTORE Custom datastore to read in data from file structure for TMN
    %   To be used as a datastore to a local directory of TUEG files.
    %
    %   Authors:
    %       Michael Caiola (Michael.Caiola@fda.hhs.gov)
    %       Meijun Ye (Meijun.Ye@fda.hhs.gov)
    %
    %   Disclaimer: This software and documentation (the "Software") were
    %   developed at the Food and Drug Administration (FDA) by employees of
    %   the Federal Government in the course of their official duties.
    %   Pursuant to Title 17, Section 105 of the United States Code,
    %   this work is not subject to copyright protection and is in the
    %   public domain. Permission is hereby granted, free of charge, to any
    %   person obtaining a copy of the Software, to deal in the Software
    %   without restriction, including without limitation the rights to
    %   use, copy, modify, merge, publish, distribute, sublicense, or sell
    %   copies of the Software or derivatives, and to permit persons to
    %   whom the Software is furnished to do so. FDA assumes no
    %   responsibility whatsoever for use by other parties of the Software,
    %   its source code, documentation or compiled executables, and makes
    %   no guarantees, expressed or implied, about its quality,
    %   reliability, or any other characteristic. Further, use of this code
    %   in no way implies endorsement by the FDA or confers any advantage
    %   in regulatory decisions. Although this software can be
    %   redistributed and/or modified freely, we ask that any derivative
    %   works bear some notice that they are derived from it, and any
    %   modified versions bear some notice that they have been modified.
    %   The Software is not intended to make clinical diagnoses or to be
    %   used in any way to diagnose or treat subjects for whom the EEG is
    %   taken.

    properties
        Datastore
        Labels
        NumClasses
        SequenceDimension
        MiniBatchSize
        Augmented
        TestSet
        newHz

        fband=[1 4; 4 8; 8 12; 12 16; 16 20; 25 40];

        chLocations
    end

    properties(SetAccess = protected)
        NumObservations
    end

    properties(Access = private)
        CurrentFileIndex
        %FileSet matlab.io.datastore.DsFileSet
    end

    methods
        function ds = TopoDatastore(folder,newHz,chLocations,varargin)
            % ds = NewDatastore(folder,newHz) creates a custom datastore
            % from the data in folder with new sampling sampling rate newHz
            %
            % chLocations are needed for topoplot, use chlocs2.mat
            %
            % Optional arguments:
            %   "DataAugmentation" (default = false)
            %       Augments the data in random shuffles of 90 seconds
            %   "TestSet (default = false)
            %       Creates a testset of 90 second data, to be used when
            %       DataAugmentation is being used

            if isempty(newHz)
                ds.newHz = 250;
            else
                ds.newHz = newHz;
            end

            inputs = ds.parseInputs(varargin{:});

            if inputs.DataAugmentation
                ds.Augmented = true;
            else
                ds.Augmented = false;
            end

            if inputs.TestSet
                ds.TestSet = true;
                ds.Augmented = false;
            else
                ds.TestSet = false;
            end

            if exist("topoplot",'file')~=2
                %open eeglab
                warning("Need EEGLAB! Trying to open...")
                try
                    path = pwd;
                    eeglabpath = uigetdir([],"Locate EEGLAB Directory");
                    cd(eeglabpath)
                    eeglab
                    close
                    cd(path)
                catch
                    error("Could not find EEGLAB. Open prior to use to add to path.")
                end
            end

            % Save chLocations
            ds.chLocations = chLocations;

            % Create file datastore.
            fds = fileDatastore(folder, ...
                'ReadFcn',@(x) readSequence(x,ds.newHz,ds.Augmented,ds.TestSet,...
                ds.fband,ds.chLocations), ...
                'IncludeSubfolders',true);
            ds.Datastore = fds;

            % Read labels from folder names.
            numObservations = numel(fds.Files);
            for i = 1:numObservations
                file = fds.Files{i};
                filepath = fileparts(file);
                [~,label] = fileparts(filepath);
                labels{i,1} = label;
            end
            ds.Labels = categorical(labels);
            ds.NumClasses = numel(unique(labels));

            % Determine sequence dimension.
            X = preview(fds);
            ds.SequenceDimension = size(X,1);

            % Initialize datastore properties.
            ds.MiniBatchSize = 128;
            ds.NumObservations = numObservations;
            ds.CurrentFileIndex = 1;
        end

        function tf = hasdata(ds)
            % tf = hasdata(ds) returns true if more data is available.

            %tf = hasdata(ds.Datastore);
            tf = ds.CurrentFileIndex + ds.MiniBatchSize - 1 ...
                <= ds.NumObservations;
        end

        function [data,info] = read(ds)
            % [data,info] = read(ds) read one mini-batch of data.

            miniBatchSize = ds.MiniBatchSize;
            if ds.NumObservations < ds.MiniBatchSize
                ds.MiniBatchSize = ds.NumObservations;
            end


            i = 0;
            while i < miniBatchSize && hasdata(ds) %hasdata(ds.Datastore)
                i = i + 1;
                Predictors{i,1} = read(ds.Datastore);
                %temppred = read(ds.Datastore);
                %predictors{i,1} = spect(temppred);
                Response(i,1) = ds.Labels(ds.CurrentFileIndex);
                ds.CurrentFileIndex = ds.CurrentFileIndex + 1;
            end

            data = preprocessData(Predictors,Response);
            info.Size = size(data);
        end

        function reset(ds)
            % reset(ds) resets the datastore to the start of the data.

            reset(ds.Datastore);
            ds.CurrentFileIndex = 1;
        end

        function dsNew = subset(ds,in)
            dsNew = copy(ds);
            dsNew.Datastore=subset(ds.Datastore,in);
            dsNew.NumObservations = numel(dsNew.Datastore.Files);
            for i = 1:dsNew.NumObservations
                file = dsNew.Datastore.Files{i};
                filepath = fileparts(file);
                [~,label] = fileparts(filepath);
                labels{i,1} = label;
            end
            dsNew.Labels = categorical(labels);
            dsNew.NumClasses = numel(unique(labels));

        end

        function varargout = grabEachLabel(ds,N,varargin)
            % pulls a certain number of each individual label
            opt_random=0;
            numvarargin=length(varargin);
            k=1;
            while k<=numvarargin
                switch varargin{k}
                    case "random"
                        opt_random=1;
                end
                k=k+1;
            end
            for i=1:length(N)
                in{i}=[];
            end
            labels=unique(ds.Labels);
            for i=1:ds.NumClasses
                labs=ds.Labels==labels(i);
                numLabs=sum(labs);
                x=find(labs);
                if opt_random
                    x=x(randperm(length(x)));
                end
                if sum(N)>numLabs
                    n=floor(numLabs.*N/sum(N));
                    warning("Not enough " + string(labels(i))+" Labels")
                else
                    n=N;
                end
                in{1}=[in{1}; x(1:n(1))];
                for j=2:length(n)
                    in{j}=[in{j};x(n(j-1)+1:n(j-1)+n(j))];
                end
            end
            for i=1:length(in)
                varargout{i} = subset(ds,in{i});
            end
        end

        function dsNew = shuffle(ds)
            % dsNew = shuffle(ds) shuffles the files and the corresponding
            % labels in the datastore.

            % Create copy of datastore.
            dsNew = copy(ds);
            dsNew.Datastore = copy(ds.Datastore);
            fds = dsNew.Datastore;

            % Shuffle files and corresponding labels.
            numObservations = dsNew.NumObservations;
            idx = randperm(numObservations);
            fds.Files = fds.Files(idx);
            dsNew.Labels = dsNew.Labels(idx);
        end


    end

    methods (Hidden = true)
        function frac = progress(ds)
            % frac = progress(ds) returns the percentage of observations
            % read in the datastore.
            frac = (ds.CurrentFileIndex - 1) / ds.NumObservations;
        end
    end

    methods (Access = 'private')
        function inputStruct = parseInputs(ds,varargin)
            p = inputParser();

            p.addParameter('DataAugmentation',false,@augmentationValidator);
            p.addParameter('TestSet',false,@augmentationValidator);
            p.parse(varargin{:});
            inputStruct = p.Results;
        end
    end
end

function data = preprocessData(Predictors,Response)
% data = preprocessData(predictors,responses) preprocesses
% the data in predictors and responses and returns the table
% data

miniBatchSize = size(Predictors,1);

% Pad data to length of longest sequence.
sequenceLengths = cellfun(@(X) size(X,2),Predictors);
maxSequenceLength = max(sequenceLengths);
for i = 1:miniBatchSize
    X = Predictors{i};

    % Pad sequence with zeros.
    if size(X,2) < maxSequenceLength
        X(:,maxSequenceLength) = 0;
    end

    Predictors{i} = X;
end

% Return data as a table.
data = table(Predictors,Response);
end

function data = readSequence(filename,newHz,aug,ts,fband,chlocs)
            % data = readSequence(filename) reads the sequence y from the MAT file
            % filename
            oldHz = 250;
            S = load(filename);
            if newHz == oldHz
                data = S.y;
            else
                data = single(resample(double(S.y),newHz,oldHz,'Dimension',2)); %↙
resampled
```

```
            end

            downsample = 90; %sec
            if aug
                r = randi(downsample*newHz-2)+1;
                data = data(:,r:r+(downsample*newHz-1));
            end
            if ts
                data = data(:,1:1+(downsample*newHz-1));
            end

            %Topo
            data = data';
            tot=bandpower(data,newHz,[1 newHz/2]);
            n = size(fband,1);
            abs_psd = zeros(size(data,2),n);
            for i=1:n
                abs_psd(:,i)=bandpower(data,newHz,fband(i,:));
            end
            rel_psd = abs_psd./tot';
            gscale = 67*2; %interpoltation value
            y = zeros(gscale,gscale,n);
            for i = 1:n
                [~,y(:,:,i)]= topoplot(rel_psd(:,i),chlocs,'noplot','on','gridscale',↙
gscale);
            end
            data = y;
            data = fillmissing(data,'constant',0);
end

function TF = augmentationValidator(valIn)

% if ischar(valIn) || isstring(valIn)
%     TF = string('none').contains(lower(valIn)); %#ok<STRQUOT>
% elseif isa(valIn,'imageDataAugmenter') && isscalar(valIn)
%     TF = true;
if islogical(valIn)
    TF = true;
else
    TF = false;
end

end
```