Supplementary Guide

This supplementary guide serves 2 purposes: 1, to help navigate the supplementary file configuration and 2, to illustrate the methodology and results referenced in the manuscript.

Important Notes:

* NN Workspace Folder: These workspaces contain the accuracy rates from the 10% holdout, the neural network before and after transfer learning, as well as the external dataset batch predictions. (Note: M1\_M6 is SGDM (.0009/.001/.0011), M13\_M18 is ADAM (.0009/.001/.0011), and the others are organized in a similar manner excluding GN (V1).
* Pictorial Data: Divided in 2, from different authors, but are organized according to the methodology.
* SoftmaxScores: RN (V1) is labeled similarly to the convention in “NN Workspace Folder. V2’s softmax scores, and RN (V1) labeled scores, can be found workspace in conjunction with use of “TestingDSClassification.m”.
* Spectral Data of Unknown: This folder possesses the original raw dataset after exclusion. However, the data has separated x and y values from the Witec Confocal Raman and the notation is in the German (comma decimal) format. Therefore, a transformation was done on dataset to order, combine, and convert the files accordingly. For your file order (by date), refer to this helpful link so that MATLAB does not call your data alphanumerically: “User “Rik”: <https://www.mathworks.com/matlabcentral/answers/402162-read-files-in-a-specific-order>”. The final file is a master csv file of the processed version of the unknown MPs from OpenSpecy’s online signal processing application.
* Statistics: Statistics for Mann-Whitney and Spearmen are placed here. Please refer to the “ReadMe”.
* Unknown MP Images: Image set of the entire 422 particles studied. Unfortunately, 2 were lost.
* PredCalc (Ver1, Ver2, and Rand): Spreadsheet used to calculate accuracy rates of each of the 48 models plus the random set against the testing datasets.
* OS Matches: For the unknown MPs after exclusion.
* Other: Refer to the ReadMe’s in a specific file folder.

Controls for Prediction Calculations (subtraction):

RW 1 – CTRL 4

TW 1 – CTRL 2

WW 1 – CTRL 3

DW 1 – CTRL 1

RW2 – CTRL 5

TW 2 – CTRL 4

WW 2 – CTRL 6

DW 2 – CTRL 5

RW 3 – CTRL 6

TW 3 – CTRL 8

WW 3 – CTRL 7

DW 3 – CTRL 7

Procedure:

Samples in the triplicate space are subtracted for samples of the same polymer type in the corresponding control. Because the space is triplicate, there are 24 particles possible vs. 8. This means that if 3 particle of cellulosic origin is found in TW 3, for example, and 1 is found in CTRL 8 then CA concentration is 0%.

A close-up of several bottles

Description automatically generated

**Figure S1.** Dual separatory funnel setup. The current step in the image is the final alcohol wash used as an MPL concentrator. The top layer was decanted into vials for later centrifugation.

A metal plate with green marks on it

Description automatically generated

**Figure S2.** Example of aluminum-coated slides of dropcasted environmental samples. Markings denote areas (in this example, 33 to 40 and 49 to 53 with (54-55-56) used as a control for atmospheric deposition of microfibers.

A collage of different cells

Description automatically generated

**Figure S3.** Collection of controls used as a visual exclusion criterion atmospherically deposited-MPs on the slide dropcast (10X: Top 9 Samples. 50X: Remainder). (Left to Right) **First Row**: CA, LDPE, HDPE, and PP. **Second Row**: PS, PET, PMMA, PA. **Third Row**: PVC, Oil, Marker, and Alcohol. **Fourth Row**: Dust\_AirborneParticle1 – 4. **Fifth Row**: Dust\_AirborneParticle5 – 8.

A screenshot of a graph

Description automatically generated

**Figure S4.** OS matches for Unknown MP: 95 (**A**), 119 (**B**), 88 (**C**), 101 (**D**), 62 (**E**), and 63 (**F**) with matches corresponding to fluorapatite, cotton, cotton, PE resin, cotton and cotton, respectively. Details on matches in SI’s file folder.

Table S1. Relative concentration of spiked samples.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Polymer | CA | HDPE | LDPE | PP | PS | PET | PMMA | PA | PVC |
| CNN | 12.5% | 12.5% | 12.5% | 0% | 0% | 12.5% | 12.5% | 0% | 0% |
| OS | 25% | 12.5% | 25% | 75% | 0% | 37.5% | 0% | 62.5% | 0% |

Code:

Training Dataset Generation

Note: NOTE: Line mismatches occur due to margins of the document. Refer to the semi-colon convention of suppressing output of code in command window as a guide. The last section of the methodology will describe the formatting of the training data.

**PictorialDataGenerator\_NN.m**

filenum = input('How many synthetic spectra is needed?:');

swc = input('Does the spectra need rescaling (1: Yes, 2: No)?');

B = readmatrix("PS\_RCowgerMASTER.csv");

B = B(1:numel(B(:,1)),1:222); %Excising metadata (ADJUST AS NECESSARY)

B = transpose(B); %row to column-wise organization

if swc == 1

for i = 1:(numel(B(1,:))-1)

B(:,1+i) = rescale(B(:,1+i));

end

end

Lines 1-2: The first input asks the users how many synthetic spectra the script needs to output. This helps set the limit of the interpolation further down the script.

Lines 4-6: The matrix of a user-specified set of polymer classes is derived from file formats such as .csv/.xls/.xlsx or a comma-delimited text file. The data in the spectra is organized, row-wise, meaning each entry has its own row with the first row being the x-array and the 2nd through nth row being the y-array. In the following line, the matrix is truncated due to presence of “string” inputs in the matrix or “NaNs” from the “readmatrix” function. The range of the dataset is automatically counted by “numel” for the row boundary and manually counted in excel for the column-boundary. Lastly, the matrix is transposed for each tracking of changes to the spectra.

l\_files = numel(B(1,:)) - 1;

C = zeros(length(B(:,1)),l\_files);

D = zeros(length(B(:,1)),filenum);

r = rand(1,l\_files);

r = r/sum(r);

l = 2;

Line 12: Counts the number of entries in the uploaded file of any size.

Line 14-18: Pre-allocates *C* used for holding the value of the scaled y-values for a given entry. *D* is also pre-allocated for space for the summation of the y-values calculated in *C* up to the user-specified limit of spectra. The variable, *r*,is assigned for an array of length, *filenum*, of random numbers generated by MATLAB. This value is then reassigned by the normalization of its sum giving an array of random numbers whose sum is 1. This satisfies the normalization of the relative absorbance values of the spectra. The variable *l* is used as an independent incrementing variable due to *B* being offset by the x-array by 1 row.

for n = 1:filenum

for i = 1:l\_files

C(:,i) = B(:,l)\*r(i);

l = l + 1;

end

D(:,n) = sum(C,2); %New spectra

r = rand(1,l\_files);

r = r/sum(r);

l = 2;

disp(n)

end

D = [B(:,1) D];

Line 20: Outer loop responsible for the sum of a set of partial spectra generated by the scaling of a random number.

Lines 21-23: The inner loop which, for length of the number of files in the uploaded dataset, will scale the spectra of the *lth* entry by the *ith* random weight into the *ith* column in *C*. The loop increments by one to overwrite the rest of the matrix.

Lines 25-29: The column-wise summation of *C* is performed resulting in the interpolated synthetic datum/spectrum (indicated by “2” in the second variable of the function). Next, *r* and *l* are re-initialized for the next calculation. *C* is not reinitialized as it will simply be overwritten for the next synthetic spectrum’s calculation. The iteration number of the outer loop is displayed indicating progress in terms of the file number from the user-specified entry.

for i = 1:filenum

plot(D(:,1),D(:,i+1))

filenum = ['PS',num2str(i+300),'.png'];

xlim([224 1992])

ylim([0 1])

yticks([0 0.2 0.4 0.6 0.8 1])

yticklabels({0 0.2 0.4 0.6 0.8 1})

ax = gca;

exportgraphics(ax,filenum)

end

Line 33: Outer loop responsible for the export of the user-specified number of synthetic spectra generated in the last part of the code.

Lines 34-41: Plots the x- and y-arrays for the *ith* synthetic spectrum. Next, the *filenum* is redefined to label the .png files accordingly (e.g. “PS301.png”-“PS400.png” generated in this example). Next, we standardized the x- and y-limits so that they match across the whole training and external testing datasets. Next, we standardize y-ticks due to variable y-ticks generated during looping. We wish for the CNNs only to consider the spectra for significant features. Lastly, the graphic is exported with the correct file name and axes object. Although not used, the axes object can reformat the axes to the user’s discretion.

The script for the external testing dataset is remarkably similar in that the only difference is that the interpolation is not performed and for one of our datasets, rescaling and reordering of the elements was needed for the external datasets (6 additional lines). See the SI section for more details.

**NNModifcationCode\_Accuracy\_ConfusionMatrix.m**

imds = imageDatastore("Version 2 - 9000 TrainingDS\FTIR\", IncludeSubfolders=true, LabelSource="foldernames");

classNames = categories(imds.Labels); %gathers class names for verification

numClasses = numel(classNames); %Verifies correct number of classes

Lines 2-4: Function creates an image datastore of the folder housing the generated synthetic spectra of 8/9 MPL polymers from 5 different sources. In addition, the parameter to include subfolders aims in the ease of organizing the data. Next, the names of the classes are dot indexed and assigned as categorical variables aiding correct attribution of softmax output to the labels defined by the subfolders of the training dataset folder. Lastly, the number of classes is counted to prepare the transfer learning for the correct number of class outputs.

[imdsTrain,imdsValidation,imdsTest] = splitEachLabel(imds,0.6,0.3,"randomized"); %splits data into given ratio for training, validation, and testing.

net = imagePretrainedNetwork("resnet18",NumClasses=numClasses); %First argument can be any pretrained network specified in the email notes!!!

inputSize = net.Layers(1).InputSize;

Lines 6-9: Function splits the dataset into 3 separate training, validating, and testing datastores based on the sub-folder (classes). In our report, we randomized our results which is summed to be negligible since the classes are interpolated across the diversity of its original dataset. The CNN was loaded for transfer learning “googlenet/resnet18” keeping the correct number of classes in mind. Lastly, the size of the first layer is retrieved and used as a guide for the augmentation of the training dataset.

augimdsTrain = augmentedImageDatastore(inputSize(1:2),imdsTrain);

augimdsValidation = augmentedImageDatastore(inputSize(1:2),imdsValidation);

augimdsTest = augmentedImageDatastore(inputSize(1:2),imdsTest);

Lines 11-13: The 3 datastores are augmented by the size of the *inputSize* dictated by the neural network of choice. This keeps the resolution consistent with the CNNs specifications.

options = trainingOptions("sgdm", ...

MaxEpochs=10, ...

InitialLearnRate=0.0009, ...

ValidationData=augimdsValidation, ...

ValidationFrequency=5, ...

Plots="training-progress", ...

Metrics="accuracy", ...

Verbose=false);

Lines 15-22: Specifies the options for the neural network to be trained with. The first parameter specifies the learning algorithm, “sgdm”, “adam”, or “rmsprop”. The last is typically used for regression analysis. The first two is used for convergence whereby sgdm [49], stochastic gradient descent method, converges efficiently as it ignores redundant samples and prevents accidental convergence at a local minimum (due to stochasticity). Adaptive moment estimation, *adam* [50], is an alternative optimization method which is efficient, requires little memory, and suited for large data/parameters. Adaptive learning rates are used and step sizes are annealed for faster convergence. *MaxEpochs* refer to the number passthroughs of the training dataset. 10 was selected as to optimize training time (not too large for computational cost but not too small to avoid underfitting). *InitialLearningRate* refers to the frequency in which the network adjusts its weights (in some cases, people institute adaptive learning rates for faster convergence). Next, the validation data is set for adjustments during training. *ValidationFrequency* refers to the number of iterations taken during training until a subset of the validation dataset is tested for fine-tuning the model. *Plots*, *Metrics*, *Verbose* specify details outputted as the training proceeds. NOTE: Version 2 (V2) of our dataset uses an additional metric for “options” known as *MiniBatchSize* enabling the model to converge due to the memory limit of MATLAB online.

**Parameters Used in Study:** Learning Rate: .0009, .001, and .0011; SGDM and ADAM; and (V2 only) MiniBatchSize = 32 (default: 128)

net\_2 = trainnet(augimdsTrain,net,"crossentropy",options);

YTest = minibatchpredict(net\_2,augimdsTest);

YTestLabels = scores2label(YTest,classNames);

TTest = imdsTest.Labels;

acc = mean(TTest==YTest);

disp(acc)

plotconfusion(TTest,YTestLabels)

Lines 24-30: Transfer learning takes place here with the training portion of the datastore, the neural network, the cross-entropy loss function is specified as an output indicating the progress of classification between different classes, and the options specified in lines 15-22.

imds\_ext = imageDatastore("PrimpkeFTIR\_Pic\",IncludeSubfolders=true,LabelSource="foldernames");

auds\_ext = augmentedImageDatastore(inputSize(1:2),imds\_ext);

YTest\_ext = minibatchpredict(net\_2,auds\_ext);

disp(YTest\_ext)

Lines 33-36: Creates the image datastore of the external test dataset. Similar to lines 2, 11-13, the datastore is organized by subfolder and augmented to the correct input size of the user-selected algorithm. Lastly, the softmax scores are saved and displayed for later evaluation from the newly modified neural network’s prediction.

**TestingDSClassification.m**

load("V2 Data\M24\_V2.mat")

Maximums = max(YTest\_ext,[],2);

LPred = length(YTest\_ext(:,1));

Pred = cell(LPred,1);

Lines 5-8: Loaded data from the workspace saved from the previous script includes the softmax output from the batch prediction. Maximums are sought out for which help index the rows (entry) for the correct MPL polymer class label. Next, the prediction matrix is pre-allocated by the size of the predictions from the previous script as a 1D cell array.

for i = 1:length(YTest\_ext(:,1))

[row(i) col(i)] = find(YTest\_ext(i,:)==Maximums(i));

if col(i) == 1

Pred(i) = {"CA"};

elseif col(i) == 2

Pred(i) = {"HDPE"};

elseif col(i) == 3

Pred(i) = {"LDPE"};

elseif col(i) == 4

Pred(i) = {"PA"};

elseif col(i) == 5

Pred(i) = {"PET"};

elseif col(i) == 6

Pred(i) = {"PMMA"};

elseif col(i) == 7

Pred(i) = {"PP"};

elseif col(i) == 8

Pred(i) = {"PS"};

elseif col(i) == 9

Pred(i) = {"PVC"};

end

end

Lines 9-10: For loop generated based on the length of predictions. The next line of code indexes the maximum found earlier at specific positions (column-based) for the *ith* entry of the matrix of softmax scores.

Lines 11-30: Applies a cell-formatted label to the column-vector corresponding to the class labels (subfolders from previous script). Check “classNames” or dot index the datastore for “labels” in order to appropriately order the classes of interest.