Deep learning networks for the recognition and quantitation of surface-enhanced Raman spectroscopy

Shizhuang Weng[a,*], Hecai Yuan[a], Xueyan Zhang[a], Pan Li[b], Ling Zheng[a,*], Jinling Zhao[a], Linsheng Huang[a]

[a]National Engineering Research Center for Agro-Ecological Big Data Analysis & Application, Anhui University, Hefei 230601, China.

[b]Center of medical physics and technology, Hefei institutes of physical science, CAS, Hefei 230021, China.

Supporting Information:

The similarity of the spectrum is significant for modelling analysis. Mean Euclidean distance (*M_Ed)* was used to evaluate the similarity of spectra in each type. First, the intensity of spectrum was normalize to 0-1. Then, *M_Ed* between spectra in each type was calculated for each dimension according to the following equation.

$$M\_Ed = \frac{\sum_{j=1}^{K}\sqrt{\sum_{i=1}^{N}(x_{j,i}-\overline{x_{j,i}})^2}}{N*K} \quad (1)$$

Where N was the dimension of spectra, and K was the number of spectra in each type. $x_{j,i}$ was the intensity of the *j-th* spectra at the *i-th* dimension, $\overline{x_{j,i}}$ was the mean value of $x_{j,i}$ in each type. The closer the value of *M_Ed* was to 0, the higher similarity data set was.

From Table S1 and Table S2, the *M_Ed* of spectra in recognition of drugs in urine was 0.0164 to 0.0316, and the *M_Ed* of spectra in quantitation of pirimiphos-methyl in wheat extract was 0.0255 to 0.0480, indicating the spectra were of high similarity.

Table S1. Similarity of data in recognition of drugs in urine.

Type of spectra	*M_Ed*
Raw urine	0.0222
Urine samples with10 mg/L MDMA	0.0164
Urine samples with 50 mg/L MDMA	0.0245
Urine samples with 25 mg/L MAMP and 25 mg/L MDMA	0.0316
Urine samples with10 mg/L MAMP	0.0208
Urine samples with 50 mg/L MAMP	0.0236

Table S2. Similarity of spectra in quantitation of pirimiphos-methyl in wheat extract.

Type of spectra	*M_Ed*
Wheat extract solution with 25 mg/L pirimiphos-methyl	0.0370
Wheat extract solution with 10 mg/L pirimiphos-methyl	0.0480
Wheat extract solution with 5 mg/L pirimiphos-methyl	0.0336
Wheat extract solution with 2.5 mg/L pirimiphos-methyl	0.0407
Wheat extract solution with 1 mg/L pirimiphos-methyl	0.0340
Wheat extract solution with 0.5 mg/L pirimiphos-methyl	0.0255
Wheat extract solution with 0.2 mg/L pirimiphos-methyl	0.0303

Performance of deep learning networks heavily depended on the parameter optimization. The architecture of CNNs was derived from LeNet-5 in our study. 1) For CNNs in classification, the activation and optimization functions became Relu and Rmsprop, and the activation function of the final output layer was Softmax. The number of epochs and batch_size (16, 32 and 64) was changed to make the networks train fully. 2) Then, the learning rate (0.0001, 0.001, 0.1 and 1) and the number of network nodes were tuned for the better results. 3) After the above parameters optimization, the number of network layer was increased or reduced for improving the performance of networks furtherly. 4) To prevent the overfitting, the BatchNormlization was generally added behind the network layer. The above steps were cycled in order until the performance of CNNs reaches stable. For CNNs in regression, the activation function of the last layer was omitted, but the hyperparameter optimization was same.

The architecture of FCNs was derived from CNNs. The pooling and the full connection layer were replaced by the convolution layer in FCNs. Global average pooling was used to connect features and outputs (category or concentration). The optimization of hyperparameters was basically consistent as CNNs. The architecture of PCANet was fixed, and a small number of parameters could be adjusted one by one.

Parameter setting of classification models:

The parameter settings for the FNN, CNN^a, FCN^a, $PCANe^a_{KNN}$, $PCANet^a_{RF}$, $PCANet^a_{SVM}$, CNN^b, FCN^b, $PCANet^b_{KNN}$, $PCANet^b_{RF}$, $PCANet^b_{SVM}$, KNN, RF and SVM models with raw spectra were displayed in Table S3.

Table S3. Parameter setting of different classification models.

Methods	Parameters
FNN	Dense_1:1024, Relu; Dense_2: 512, Relu; Dense_3: 32, Relu; Dense_4: 6, SoftMax; optimizer=optimizers.rmsprop(lr=0.0001), loss='categorical_crossentropy', metrics=['accuracy'], batch_size=64, epochs=90
CNN^a	Conv_1:32,1×3,Relu; Max-pooling: 1×2, stride=(1,2); BatchNormalization Conv_2:64,1×3,Relu; Max-pooling: 1×2, stride=(1,2); BatchNormalization Dense_1:32, Relu; Dense_2:6, SoftMax optimizer=optimizers.rmsprop(lr=0.0001), loss='categorical_crossentropy', metrics=['accuracy'], batch_size=64,epochs=60
FCN^a	Conv_1:32,1×3,Relu; Conv_2: 32, 1×2, stride=(1,2), Relu; BatchNormalization

	Conv_3:64,1×3,Relu; Conv_4: 64, 1×2, stride=(1,2), Relu; BatchNormalization Conv_5:32,1×1,Relu; GlobalAveragePooling2D; Dense_1:6, SoftMax optimizer=optimizers.rmsprop(lr=0.0001), loss='categorical_crossentropy', metrics=['accuracy'], batch_size=64,epochs=60
PCANet[a]*KNN*	filter_shape_l1= (1,3), step_shape_l1= (1,2), n_l1_output=3, filter_shape_l2= (1,4), step_shape_l2= (1,4), n_l2_output=3, filter_shape_pooling=1, step_shape_pooling=1 n_neighbors=1
PCANet[a]*RF*	filter_shape_l1=(1,3), step_shape_l1=(1,2), n_l1_output=3, filter_shape_l2=(1,4), step_shape_l2=(1,4), n_l2_output=3, filter_shape_pooling=1, step_shape_pooling=1 'max_depth': None,'max_features': 'auto', 'n_estimators': 10,
PCANet[a]*SVM*	filter_shape_l1=(1,3), step_shape_l1=(1,2), n_l1_output=3, filter_shape_l2=(1,4), step_shape_l2=(1,4), n_l2_output=3, filter_shape_pooling=1, step_shape_pooling=1, C=10
CNN[b]	Conv_1:32,3×3,Relu; Max-pooling: 2×2, stride=(2,2); BatchNormalization Conv_2:64,3×3,Relu; Max-pooling: 2×2, stride=(2,2); BatchNormalization Dense_1:32, Relu; Dense_2:6, SoftMax optimizer=optimizers.rmsprop(lr=0.0001), loss='categorical_crossentropy', metrics=['accuracy'], batch_size=64,epochs=60
FCN[b]	Conv_1:32,3×3,Relu; Conv_2:32,2×2, stride=(2,2); BatchNormalization Conv_3:64,3×3,Relu; Conv_4:32,2×2, stride=(2,2); BatchNormalization Conv_5:32,1×1,Relu; GlobalAveragePooling2D; Dense_6: SoftMax optimizer=optimizers.rmsprop(lr=0.0001), loss='categorical_crossentropy', metrics=['accuracy'], batch_size=64,epochs=60
PCANet[b]*KNN*	filter_shape_l1=2, step_shape_l1=1, n_l1_output=4, filter_shape_l2=2, step_shape_l2=1, n_l2_output=4, filter_shape_pooling=1, step_shape_pooling=1 n_neighbors=1
PCANet[b]*RF*	filter_shape_l1=2, step_shape_l1=1, n_l1_output=4, filter_shape_l2=2, step_shape_l2=1, n_l2_output=4, filter_shape_pooling=1, step_shape_pooling=1 'max_depth': None,'max_features': 'auto','n_estimators': 10,
PCANet[b]*SVM*	filter_shape_l1=2, step_shape_l1=1, n_l1_output=4, filter_shape_l2=2, step_shape_l2=1, n_l2_output=4, filter_shape_pooling=1, step_shape_pooling=1 C=10
KNN	n_neighbors=1, weights=uniform
RF	'max_depth': None,'max_features': 'auto', 'n_estimators': 10,
SVM	'C': 1, kernel': 'linear',

[a] Input of the networks is one dimensional vector. [b] Input of the networks is two dimensional matrix.

Parameter setting of regression models:

The parameter settings for the FNN, CNN[a], FCN[a], PCANet[a]$_{PLSR}$, PCANet[a]$_{RF}$, PCANet[a]$_{SVM}$, CNN[b], FCN[b], PCANet[b]$_{PLSR}$, PCANet[b]$_{RF}$, PCANet[b]$_{SVM}$, LR, RF and PLSR models with raw spectra were displayed in Table S4.

Table S4. Parameter setting of different regression models.

Methods	Parameters
FNN	Dense_1:16, Relu; Dense_2: 8, Relu; Dense_3: 1; optimizer=optimizers.rmsprop(lr=0.0001), loss='mse', batch_size=4, epochs=200
CNN[a]	Conv_1:16,1×3,Relu; Max-pooling: 1×2, stride=2; BatchNormalization; Conv_2:32,1×3,Relu; Max-pooling: 1×2, stride=2; BatchNormalization; Dense_1:32, Relu; Dense_2:16, Relu; Dense_3: 1 optimizer=optimizers.rmsprop(lr=0.0001), loss='mse', batch_size=4, epochs=120
FCN[a]	Conv_1:16, 1×3, Relu; Conv_2:16, 1×2, stride= (1,2), Relu; BatchNormalization; Conv_3:32, 1×3, Relu; Conv_4:32, 1×2, stride= (1,2), Relu; BatchNormalization; Conv_5:32, 1×1, Relu; Conv_6:16, 1×1, Relu; GlobalAveragePooling2D; Dense_1: 1, optimizer=optimizers.rmsprop(lr=0.0008), loss='mse', batch_size=4, epochs=120
PCANet[a]$_{PLSR}$	filter_shape_l1=(1,4), step_shape_l1=(1,1), n_l1_output=3, filter_shape_l2=(1,4), step_shape_l2=(1,1), n_l2_output=3, filter_shape_pooling=(1,3), step_shape_pooling=(1,1) n_components=30
PCANet[a]$_{RF}$	filter_shape_l1=(1,4), step_shape_l1=(1,1), n_l1_output=3, filter_shape_l2=(1,4), step_shape_l2=(1,1), n_l2_output=3, filter_shape_pooling=(1,3), step_shape_pooling=(1,1) 'max_depth': None, 'max_features': 'auto', 'n_estimators': 10,
PCANet[a]$_{SVM}$	filter_shape_l1=(1,4), step_shape_l1=(1,2), n_l1_output=3, filter_shape_l2=(1,4), step_shape_l2=(1,4), n_l2_output=3, filter_shape_pooling=(1,3), step_shape_pooling=(1,1) C=10
CNN[b]	Conv_1:16,3×3,Relu; Max-pooling: 2×2, stride=2; BatchNormalization; Conv_2:32,3×3,Relu; Max-pooling: 2×2, stride=2; BatchNormalization; Dense_1:32, Relu; Dense_2:16, Relu; Dense_3: 1 optimizer=optimizers.rmsprop(lr=0.0001), loss='mse', batch_size=4, epochs=120
FCN[b]	Conv_1:16, 3×3, Relu; Conv_2:16, 2×2, stride= (2,2), Relu; BatchNormalization; Conv_3:32, 3×3, Relu; Conv_4:32, 2×2, stride= (2,2), Relu; BatchNormalization; Conv_5:32, 1×1, Relu; Conv_6:16, 1×1, Relu; GlobalAveragePooling2D; Dense_1: 1; optimizer=optimizers.rmsprop(lr=0.0001), loss='mse', batch_size=4,

	epochs=120
PCANet$^{b}_{PLSR}$	filter_shape_l1=2, step_shape_l1=1, n_l1_output=3, filter_shape_l2=2, step_shape_l2=1, n_l2_output=3, filter_shape_pooling=2, step_shape_pooling=2 n_components=30
PCANet$^{b}_{RF}$	filter_shape_l1=2, step_shape_l1=1, n_l1_output=3, filter_shape_l2=2, step_shape_l2=1, n_l2_output=3, filter_shape_pooling=2, step_shape_pooling=2 'max_depth': None, 'max_features': 'auto', 'n_estimators': 10,
PCANet$^{b}_{SVM}$	filter_shape_l1=2, step_shape_l1=1, n_l1_output=3, filter_shape_l2=2, step_shape_l2=1, n_l2_output=3, filter_shape_pooling=2, step_shape_pooling=2 C=10
LR	'copy_X': True, 'fit_intercept': True, 'n_jobs': None, 'normalize': False
RF	'max_depth': None, 'max_features': 'auto', 'n_estimators': 10,
PLSR	'n_components': 100, 'tol': 1e-06

[a] Input of the networks is one dimensional vector. [b] Input of the networks is two dimensional matrix.

KNN: n_neighbors—the value of k in KNN.
weights—dentified the weights of the nearest neighbor samples for each sample.
'uniform' -- all the nearest neighbor samples have the same weight.

RF:
n_estimators—the number of decision trees in a random forest.
max_features—maximum feature number of random forest partition.
max_depth—the maximum depth of the decision tree.

PLSR:
n_components—the number of components to keep.
tol—tolerance used in the iterative algorithm default 1e-06.

LR:
fit_intercept—calculate the intercept for this model.

SVM:
C—Penalty parameter C of the error term.
kernel—Specifies the kernel type to be used in the algorithm.

PCANet:
parameters for the 1st layer: filter_shape_l1, step_shape_l1, n_l1_output
parameters for the 2nd layer: filter_shape_l2, step_shape_l2, n_l2_output
parameters for the pooling layer: filter_shape_pooling, step_shape_pooling