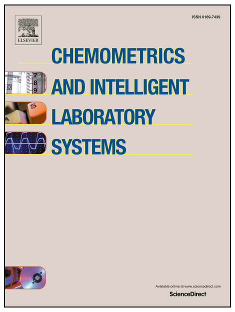


Modern practical convolutional neural networks for multivariate regression: Applications to NIR calibration

Chenhao Cui, Tom Fearn

PII: S0169-7439(18)30138-2

DOI: 10.1016/j.chemolab.2018.07.008

Reference: CHEMOM 3655

To appear in: *Chemometrics and Intelligent Laboratory Systems*

Received Date: 27 February 2018

Revised Date: 3 July 2018

Accepted Date: 15 July 2018

Modern practical convolutional neural networks for multivariate regression: applications to NIR calibration

Chenhao Cui and Tom Fearn

Department of Statistical Science,
University College London,London, WC1E 6BT, U.K.

Email:chenhao.cui.14@ucl.ac.uk; Tel:+44 747 838 3032

Abstract

In this study, we investigate the use of convolutional neural networks (CNN) for near infrared (NIR) calibration. We propose a unified CNN structure that can be used for general multivariate regression purpose. The comparison between the CNN method and the partial least squares regression (PLSR) method was done on three different NIR datasets of spectra and lab reference values. Datasets are from different sources and contain 6998, 1000 and 415 training and 618, 597 and 108 validation samples, respectively. Results indicated that compared to the PLSR models, the CNN models are more accurate and less noisy. The convolutional layer in the CNN model can automatically find the suitable spectral preprocessing filter on the dataset, which significantly saves efforts in training the model.

Keywords: Near-infrared spectroscopy; Multivariate regression; Partial least squares regression; Convolutional neural networks; Automatic spectral preprocessing

1 Introduction

Convolutional neural networks have recently become the popular solution for different machine learning tasks, including object detection [1], image classification [2], natural language processing [3], time series classification [4] and many other applications.

Implementation of neural networks (NN) as a chemometrics technique is relatively recent. Previous researches have shown that artificial neural networks (ANN) can be used as a tool for data reduction, pattern recognition and multivariate regression in spectroscopic analysis [5] [6] [7] [8] [9]. Comparison of ANN to the PLS regression has been done on various spectroscopic datasets [10]. We have also proposed some replacement methods for solving nonlinearity in multivariate regression, such as Gaussian process regression [11] and Bayesian graphical modeling [12]. Common features of these methods are high in flexibility and adaptability, but difficult to train and prone to overfitting.

Considering a lot of recent contributions to the neural networks improvements, there is very little advances on their application to the resolution of chemometrics tasks. Especially, there are few examples on implementation of CNN for spectroscopic analysis. Two very recent works reported using CNN for spectroscopic analysis. Malek, Melgani and Bazi (2017) [13] explored 1D-CNN for spectroscopic regressions, with integration of particle swarm optimization for the training purpose. Bjerrum, Glahder and Skov (2017) [14] introduced an architecture of CNN + GP (Gaussian process regression) for regression tasks, with Bayesian optimization for hyperparameter tuning. However, we found that these works missed interpretation of the CNN models, and the experiments were based on relatively small datasets (up to a few hundreds samples).

In the article, we summarize some of the recent improvements of the CNN that can be used for multivariate regression, and systematically evaluate each of the CNN components. We will first introduce a unified CNN structure for multivariate regression, and provide insights and recommendations on constructing and optimizing the CNN models. Then we will compare the results obtained from the CNN method to the traditional treatment — PLS regression in many aspects, such as the profile of the regression coefficient curves, noise level in the model, ways to tune hyperparameters, and the prediction accuracy.

2 Methods

2.1 Preprocessing and initialization

2.1.1 Preprocessing

There are many commonly used preprocessing methods for spectroscopic calibration [15], such as standard normal variate (SNV), detrending, derivative .... In the classical treatment, raw spectra (output of the instruments) will be preprocessed by these methods before feeding into a calibration framework. The purpose is to remove instrumental noise, and reduce the variance of the spectra from sources such as scatter effect to improve robustness of the calibration model. However, there is no standard rule on choosing proper preprocessing methods. The common practice is trial-and-error experimentation.

Later in this paper, we will describe how to use the convolutional layer to automate spectroscopic preprocessing. The only required preprocessing is normalization (i.e. each variable should have a mean of zero and standard deviation of 1 along the column axis). Non-zero-centered data significantly limits the allowed gradient update directions during the optimization process. For example, the worst case is that all inputs into a fully connected layer are positive (or negative). The output of a neuron in such kind of layer will be $f(\sum_i w_i x_i + b)$, where $f(\cdot)$ is the activation function and w_i are the weights. Gradient updates on all weights of such layer can only be either all positive or all negative, which is obviously inefficient.

2.1.2 Initialization

Proper initialization of the weights in NN ensures fast convergence. If the initial weights are too small, then the signal flowing through the network gradually diminishes and eventually becomes meaningless; if the weights are initialized with too large values, the variance of input data into each layer increases rapidly and soon overflows. We hope that the signal variance does not change significantly when passing through the network. One good idea is to initialize the weights according to the layers' sizes [16]. The weights are initialized by a zero-mean Gaussian distribution whose standard deviation is $\sqrt{\frac{2}{n_l}}$, where n_l is the number of input neurons. This initialization method helps to keep the variance of all the variables in NN roughly at the same scale.

2.2 Layers

Densely connected layer also termed fully connected layer (FC layer), linearly maps input vector into another one. Neurons between two adjacent layers are pairwise connected, refer to figure 1.

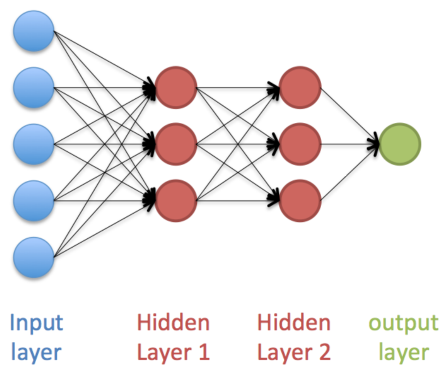


Figure 1: A neural network with two hidden layers of 3 neurons each

In our case, the initial inputs are the spectral variables, and the final outputs are the predictions of the target constituent.

Convolutional layer convolves a specific filter with the inputs. The filter is slid over the spectra spatially. We compute the stepwise dot product of the filter with a local window of the spectra, with a stride of 1. We also pad the input spectra to keep the inputs and the outputs at the same size. The convolutional layer is usually followed by fully connected layers, see figure 2 for an example.

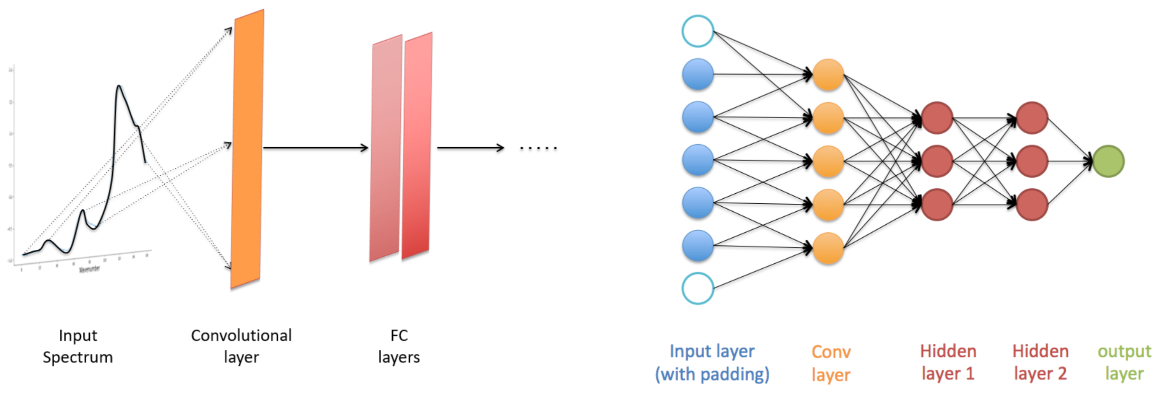


Figure 2: Architecture of a convolutional neural network

Considering that most of the preprocessing methods (e.g., detrend, Savitzky-Golay derivatives) in spectroscopic calibration are equivalent to moving weighted average of the input spectra, we believe that a properly trained convolutional layer can replace a huge range of the classical pre-processing treatments. Since the convolutional layer can be tuned by back-propagation, we do not need to manually choose any specific preprocessing method, but just let the optimization algorithm find the most efficient filter.

Since the input vectors are one dimensional spectra, so the filter is also a one dimensional vector. The bandwidth of the filter is relevant to the resolution of the input spectra. It is possible to implement multiple paralleled convolutional channels to increase the flexibility of the model, but accordingly we would need an adequate number of training samples to tune multiple convolutional filters. In this article we only consider single channel (i.e., there is only one convolutional layer) for simplicity.

2.3 Activation functions

Outputs of each hidden layer, including the convolutional layer and the fully connected layer, are transformed by the activation function to allow for nonlinearity. In the following discussion, we denote the input to the activation functions as x and the output as y.

Sigmoid function computes sigmoid of x element-wise. Specifically:

$$y = \frac{1}{1 + \exp(-x)} \tag{1}$$

The sigmoid function has been historically popular, due to nice biological interpretations. However, saturated neurons can cause gradient vanishing: gradient flow during back propagation is almost 0 when x is not near 0. In addition, the sigmoid function is not zero-centered, which means updates on the parameters can be very inefficient.

Tanh function computes hyperbolic tangent of x element-wise. Specifically:

$$y = \tanh(x) \tag{2}$$

It achieves zero-centering by squashing the output into $[-1, 1]$. However, the gradient vanishing issue remains [17].

Rectified linear units(ReLU) takes element-wise rectified linear of the input, where all negative values of the input are set to zero. More specifically:

$$y = \begin{cases} x, & x \geq 0 \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

ReLU does not saturate. In addition, it converges faster than sigmoid and tanh [18]. However, it is still non zero-centered. The main annoyance is that the output of negative input is always 0, which may cause a specific issue called "dead ReLU". When inputs into some of the ReLU neurons are always negative, such kind of ReLU neurons will never contribute to the weight updates. This happens when a bad initialization or learning rate is assigned to the model.

Exponential linear units(ELU) is very similar to ReLU, while the negative part is exponential instead of zero. More specifically:

$$y = \begin{cases} x, & x \geq 0 \\ \exp(x) - 1, & \text{otherwise} \end{cases} \tag{4}$$

It has all the benefits of ReLU, but the neurons cannot become "dead". It is also closer to zero-centered, which makes weight updates more efficient. In addition, there is a negative saturation regime which makes it robust against some noises [19]. One minor weak point is that calculating $\exp(x)$ is slow, especially when the network is relatively large.

2.4 Regularization

The model has many more parameters than observations, so some form of regularization is necessary to prevent overfitting.

L2 regularization is probably the most common form of regularization. It has been applied to many different machine learning schemes. By penalizing the sum of squared amplitude of the weights in the model via $\frac{1}{2}\lambda w^2$ (where λ is the regularization parameter, and w is the weight), L2 regularization encourages the model to use all the neurons. The gradient of L2 regularization is λw, which means every weight is steadily shrunk towards zero whatever its current sign.

Dropout is a simple way to prevent neural networks from overfitting [20]. The idea can be illustrated by figure 3. During training, we randomly activate a proportion of neurons in FC layers, and only update the parameters of the selected neurons. During testing no dropout is applied. Dropout is simple but efficient. It is widely used in modern practical neural networks. Some recent studies [20] [21] also illustrates its principle and relation with other regularization techniques.

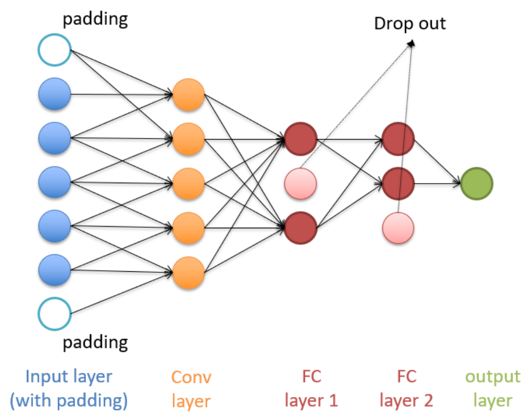


Figure 3: Illustration of dropout: two random neurons in the fully connected layers are dropped out during training

2.5 Loss functions

The loss function adopted in this research is the mean squared error (MSE) plus L2 penalty on all weights, including weights in the convolutional layer and the fully connected layers:

$$Loss = MSE + \frac{1}{2}\lambda \sum w_i^2 \tag{5}$$

2.6 Optimization

Back-propagation is the basic optimization method used in training neural networks [22]. It calculates the error contribution of each neuron by evaluating some training samples (normally a batch of data drew from the whole training set). It is normally used by the gradient descent optimization algorithms (including the Adam optimizer [23], which is used in this study) to tune the parameters in the model. The gradient of the loss function is calculated and then distributed backwards through the whole network.

Batch optimization is a necessary tool to speed up the optimization process for large datasets. When the training set is relatively small, we can simply use all the samples to tune the model simultaneously. However, when the training set is too large, it is very slow to evaluate the contribution from all the samples in each update. This is critical for neural networks because the parameters are updated during repeated forward/backward passes. Normally we only evaluate a small proportion of the training samples, which is called a mini-batch, for each update. The term 'epoch ', instead of iteration (or loops), is used to indicate the training process. One epoch means one forward/backward pass of all the training samples (i.e. $1\ epoch = \frac{train\ set\ size}{batch\ size}\ iterations$).

Adam optimization is a recently proposed optimizer for neural networks [23], and is often considered the best optimizer currently available. It has many beautiful features, such as rescaling invariance, suitability for non-stationary loss function, and automatic learning rate annealing. When using the Adam optimizer, we need to predetermine 4 hyperparameters: β_1 (decay parameter for the gradient, normally set to 0.9); β_2 (decay parameter for the squared gradient, normally set to 0.999); α (learning rate) and ϵ (a very small number to prevent division by 0, for example 10^{-6}). Adam is often recommended as the default optimizer. There is also a nice research benchmarking other popular stochastic optimizers [24] as optional choices.

2.7 Regression coefficients of a neural network

When using nonlinear activation functions or nonlinear layers (e.g., the convolutional layer is a nonlinear transformation), the whole neural network will be nonlinear. In this case we cannot use a single linear regression equation to represent the model. The analytical regression formula is often not mathematically available or cannot be obtained. We think it is the major resistance to popularize neural networks for NIR calibration, because usually the users wish to 'see' the model. In this study, we propose a method to visualise the regression coefficients of neural networks numerically, which helps the users to understand and interpret the obtained models.

The method can be used for any predictor, linear or nonlinear. We treat the predictor as a black box that maps the input spectrum to a single prediction value. We denote the input spectrum as $\mathbf{x} = \{x_1, x_2, \ldots, x_n\}$, and the predictor as a unknown function $f(\cdot)$. For a linear predictor the function is:

$$f(\mathbf{x}) = \sum_{i=1}^{n} w_i x_i + b \tag{6}$$

For a nonlinear predictor, such as neural networks, the formula cannot be obtained. However, for a known input spectrum, we can use the numerical method to calculate the regression coefficients on that specific input:

$$w_i = \frac{f(x_1, \ldots, x_i + \epsilon, \ldots, x_n) - f(\mathbf{x})}{\epsilon} \tag{7}$$

where ϵ is a very small number such as 10^{-6}. Obviously the coefficients for a nonlinear predictor depend on the inputs. In this study, when presenting the regression coefficients of a CNN model, we randomly draw a few spectra from the dataset and plot corresponding regression coefficients. By looking at the profile and variability of the regression coefficients, we can tell robustness and nonlinearity of the model. We can use the Fourier transform on the regression coefficients to quantitatively evaluate the robustness of a model. More specifically, high frequency components in Fourier domain are usually random noises in the model. In this research, we consider any component with a frequency higher than 0.3 periods per sampling space as noise factor. Proportion of noise in magnitude (i.e., sum of noises' magnitude/sum of all components' magnitude) was denoted as noise level of a regression model.

2.7.1 Model comparison

In this study, the PLSR and the CNN models were always trained on the same training sets, then evaluated on the common test sets. In PLSR, the NIPALS decomposition was employed [25]. Preprocessing methods and number of PLS factors were chosen by trial-and-error experiments via cross-validation on training sets. To compare the performance of PLSR and CNN on the test samples, the SEPs (the standard deviations of the prediction errors) for the two methods were compared using a test for equality of correlated variances due to Pitman [27] and described by Fearn [26]. The test is reported as a confidence interval for the ratio of the SEPs. If the calibrations compared showed any significant bias on the test set, the SEP is a suitable statistic to compare.

3 Software

All the simulations in this research were done in Python. CNN models were built on Tensorflow (CPU only, version 1.4.0). PLSR was done with scikit-learn (version 0.18.2).

4 Datasets

Dataset1 is on wheat flour. The target constituent is the protein content. Origins of the samples span the globe. The samples also consist of multiple varieties. The dataset contains 1000 training samples and 597 separate test samples. NIR spectral range is from $850nm$ to $1650nm$, with a spectral resolution of $5nm$.

Dataset2 is also on wheat flour. The target constituent is the ash concentration. The samples have different origins and varieties. There are 6987 training samples and 618 independent test samples. The spectral range covers $850nm$ to $1650nm$, with a resolution of $5nm$.

Dataset 3 is a public dataset [28]. The spectra and the corresponding protein content were collected from different locations in Denmark. The dataset contains 415 training samples and 108 test samples.

Dataset 1 and dataset 2 are original to this paper. The two datasets were provided by Bühler AG. Wheat flour was thoroughly homogenized before measured. Multiple measurements were repeated on the same sample, under different conditions. There are no obvious outliers in the two datasets. The training samples and the test samples were collected and measured independently, under the same spectroscopic and lab systems.

5 Results and discussions

5.1 Experiment 1: Self-preprocessing, learning rate, regularization and dropout

This part of study is based on dataset 1. The CNN is constructed as shown in table 1:

CNN structure for dataset 1			
Layer name	Input dimension	No. of parameters	Output dimension
Input	161		161
Conv layer	161	$5 \times 1 \times 1$	161
FC 1	161	$161 \times 36 + 36$	36
FC 2	36	$36 \times 18 + 18$	18
FC 3	18	$18 \times 12 + 12$	12
Output	12	$12 \times 1 + 1$	1

Table 1: CNN structure for dataset 2

Initial inputs to the CNN model are the normalized (in column axis) spectra. Notice here we do not use any other preprocessing methods, while in PLS regression spectra were preprocessed by first derivative followed by SNV. The preprocessing methods for the PLS regression model were selected by cross-validation. There are some predetermined hyperparameters in the CNN model: learning rate = 0.01, L2 regularization parameter $\lambda = 0.005$, batch size =256, no dropout. Later we will discuss impact of these hyperparameters in more detail.

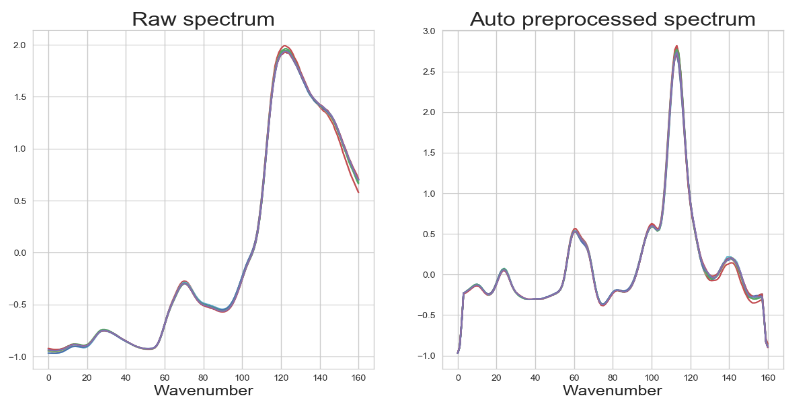
The CNN model was compared to the PLSR model (PLS factors: 8, preprocessing: 2 side points, 2^{nd} order polynomial fitted 1^{st} derivative followed by SNV). The PLSR was tuned by cross-validation. The results are showed in table 2. Since the 95% confidence interval on the ratio of SEP does not include 1, we can conclude that the prediction precision of the CNN model is significantly better then the PLSR model on that test set.

Model	RMSEC	RMSEP	SEP
PLSR	0.329	0.291	0.290
CNN	0.254	0.230	0.229
CI			(1.229,1.310)

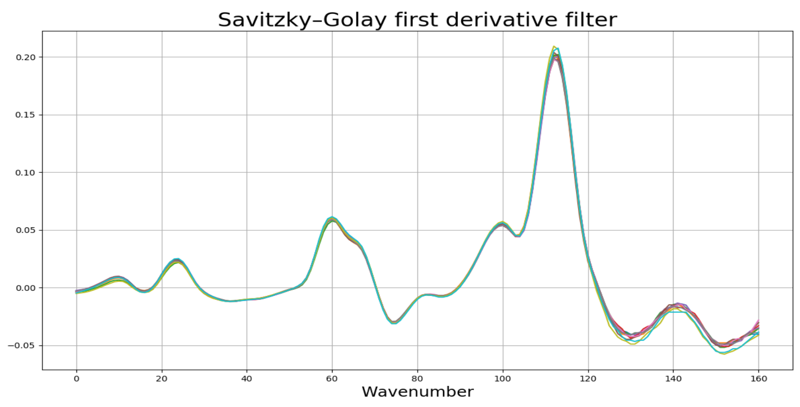
Table 2: Comparison of the results of the two models on dataset 1, CI stands for 95% confidence interval of the ratio on SEP (PLSR/CNN). Since the interval does not include 1, the CNN model is significantly better than the PLSR model on the test set.

5.1.1 Automatic preprocessing

As introduced in section 2.2, the convolutional layer can transform the spectra to fit in the following regression scheme. Since the convolutional layer was trained automatically by the optimizer, we visualize the output of the convolutional layer to see the actual transformation. The result is shown in figure 4.



(a) Spectroscopic transformation by the convolutional layer



(b) Savitzky–Golay first derivative filter:bandwidth=5, polynomial fitting order=2

Figure 4: (a) 20 random spectra in dataset 1 and the corresponding outputs of the convolutional layer. (b) 2 side points, second polynomial fitting Savitzky-Golay first derivative on the same 20 spectra.

Based on the plots, we can see that the transformation of the convolutional layer is similar to a Savitzky-Golay first derivative filter. In fact, they are not exactly the same. Outputs of the convolutional layer correspond to a skewed and rescaled standard first derivative. The convolutional layer can continually tune the variables in the filter, until it finds the best form of preprocessing. This means the spectroscopic preprocessing done by the convolutional layer is more flexible. More importantly, the traditional way of choosing the preprocessing methods is via trial-and-error experiment, which is quite labour and time consuming. The proposed CNN method saves a lot of effort when building new calibrations.

5.1.2 regression coefficients

We use the method introduced in section 2.7 to plot the regression coefficients of the CNN model. The results are shown in figure 5. We randomly drew 100 spectra from the training set, and calculated the corresponding regression coefficient curves. As we can see in figure 5, the two models have similar profiles. The CNN model recognizes a few more signals in the spectra, but overall it is very smooth and robust.

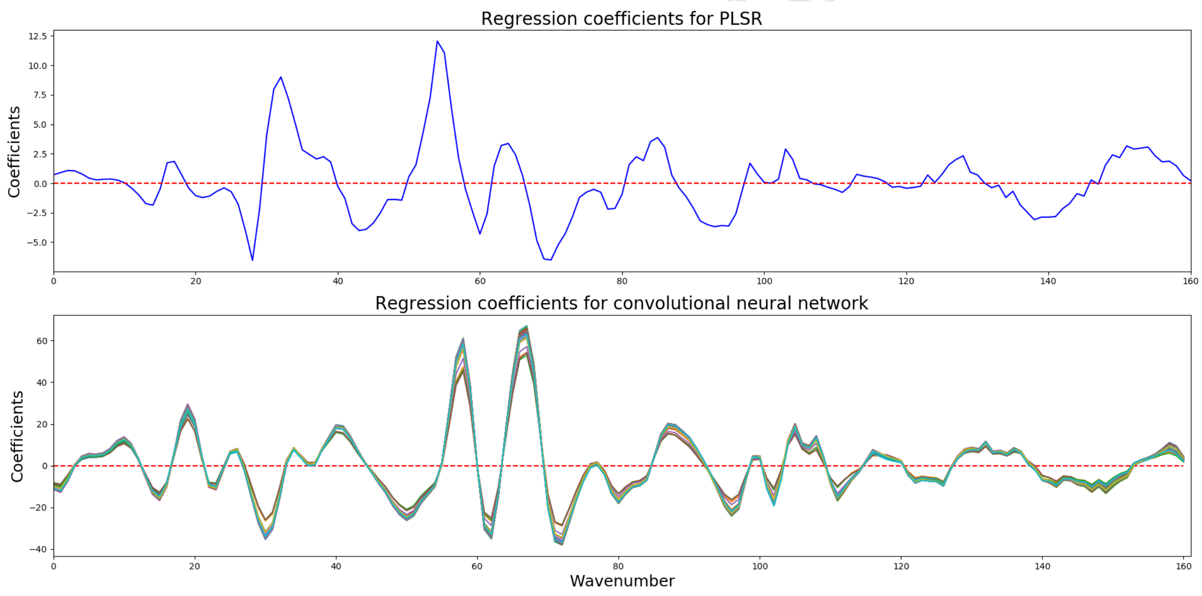


Figure 5: Comparison of regression coefficients trained by PLSR and CNN

When training a CNN model, we need to determine a few hyperparameters: λ (regularization parameter), learning rate, activation function and dropout ratio. In what follows, we explore the impacts of these hyperparameters on the CNN model, and hence provide insights how to determine them. It is worth noting the best practice to tune the hyperparameters is by coarse-fine grid searching on a separate validation set. However for NIR calibration, a separate validation set is not always available, so cross-validation can be used instead. We propose some common choices for these hyperparameters. They can be used as default values when lacking resources to tune.

5.1.3 Learning rate

During the training process, we randomly draw a batch of samples from the training set, and then update all the parameters based on the gradients. Learning rate determines the step of each update. If the learning rate is too large, the update will become too aggressive, which might lead to a bad local optimum. If the learning rate is too small, the loss function converges very slowly.

Krizhevsky proposed a way to determine the learning rate [29]:

$$learning\ rate = 0.01 \times batch\ size/256 \tag{8}$$

For example, if we fix the batch size at 256, then the learning rate should be 0.01. We found the heuristic proposed is valid on our tested datasets. Refer to figure 6 for the results. We plot three training loss curves through the optimization process, with different learning rates. When we set the learning rate to 0.01 (red line), the loss decays consistently and efficiently. We think it is a good learning rate. When we set the learning rate as 0.05 (blue line), which is too high, the objective function finally converges to a bad local optimum. When learning rate is too small (green line, learning rate =0.005), we found it slow to converge (but will eventually converge to a good local optimum). In summary we recommend using 0.01 as default learning rate when the batch size is 256.

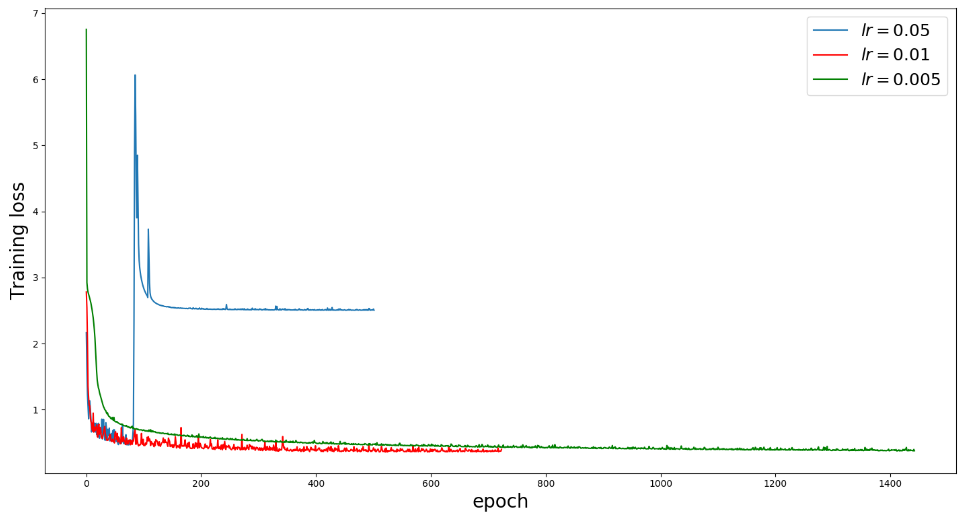


Figure 6: Optimization process with different learning rate

5.1.4 Activation function

Referring to table 1, we inserted an activation function after each hidden layer. The full structure is: Input - Conv layer - activation - FC1 - activation - FC2 - activation - FC3 - activation - output. We tested 4 different activation functions: Sigmoid, ReLU, ELU, ELU + ReLU. In ELU+ReLU we used ELU after the convolutional layer but ReLU after the FC layers. The final RMSEP on the test set are showed in figure 7. We use PLSR as a benchmarking method as well.

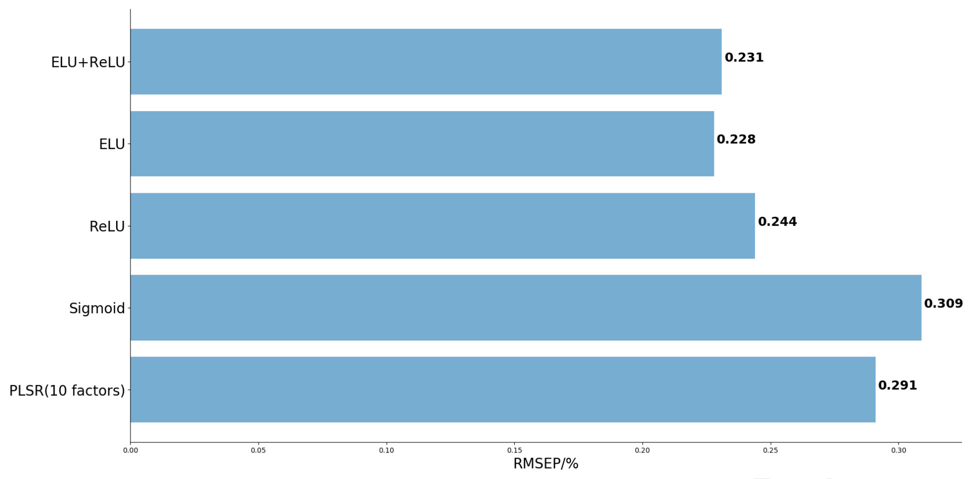


Figure 7: Performance of different activation functions

1. We found that the sigmoid function is not suitable in our CNN model. This is due to the fundamental limitations of the sigmoid function (gradient vanishing, non-zero-centered). The result is worse than the PLSR method.

2. ELU is slightly better than ReLU. The ELU+ReLU configuration is very close to using ELU alone, we consider they have equivalent performance.

3. The biggest advantage of ReLU over ELU is lower computational cost and fast convergence. In ReLU all the negative outputs are set to zeros while positive outputs are unchanged. In contrast ELU involves calculating exponentials, which is one of the most expensive operations. Refer to figure 8, the ReLU CNN converges in around 600 epochs while ELU takes roughly 1000. A good trade-off is using ELU after the convolutional layer, to keep the desired nonlinearity, then ReLU after the FC layers to improve efficiency in training. This configuration benefits from both training speed and prediction accuracy. It is worth noting that we found the training time with different activation functions vary from 70 seconds to 130 seconds (on our computational system, a 2.50GHz CPU), which might be trivial in many practical cases.

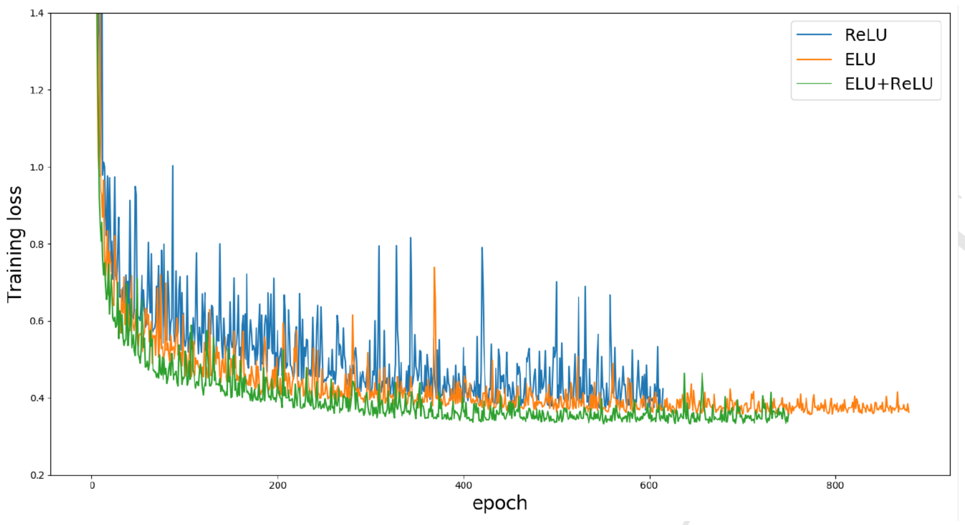


Figure 8: Convergence speed of 3 different activation functions

5.1.5 Regularization parameter

L2 regularization is the most efficient way to prevent CNN models from overfitting. We performed an experiment to learn the impact of the regularization parameter (λ) on the final CNN model. We trained the CNN models with different amounts of L2 regularization. Here we also plot the regression coefficients correspondingly, refer to figure 9.

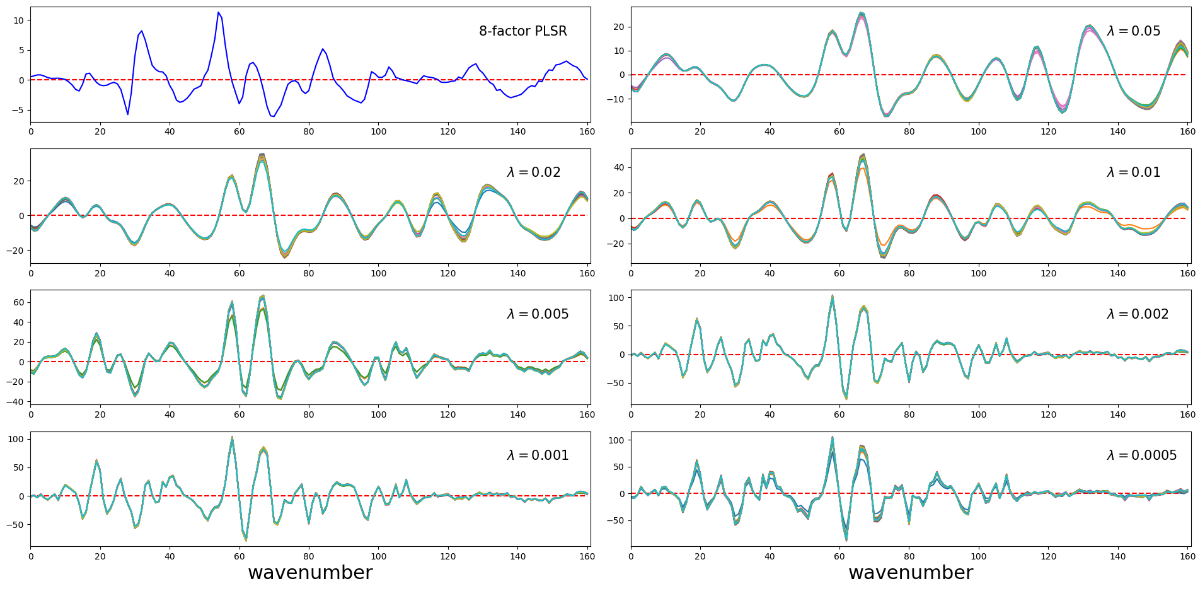


Figure 9: Regression coefficients trained by different regularization parameter. Plot on the top left (blue curve) is a 8-factor PLS regression model.

We found that λ controls the noise level of the regression coefficients. In PLSR, we use the number of factors to tune the complexity (hence the noise level) of a prediction model. λ has a similar effect but does not change the overall profile of the regression coefficients. When we apply L2 regularization to a CNN model, we always use the same principal signal and change tolerance on presence of small signals, which can be noise.

When choosing λ, we should refer to the cross-validation accuracy, i.e. the root mean squared error of cross-validation (RMSECV). However, for NIR calibration, we do not always choose the model with smallest RMSECV, which can be overfitting. Instead we often prefer the most robust model with an acceptable performance. In figure 10 we plot the Fourier transform on regression coefficients of different models. For the CNN models, magnitudes were averaged out over 100 spectra in the training set. We can clearly see how the noise level grows when the PLSR and the CNN models become more complex. In table 3 we show the RMSECV, RMSEP, SEP and noise level when changing number of PLS factors and λ.

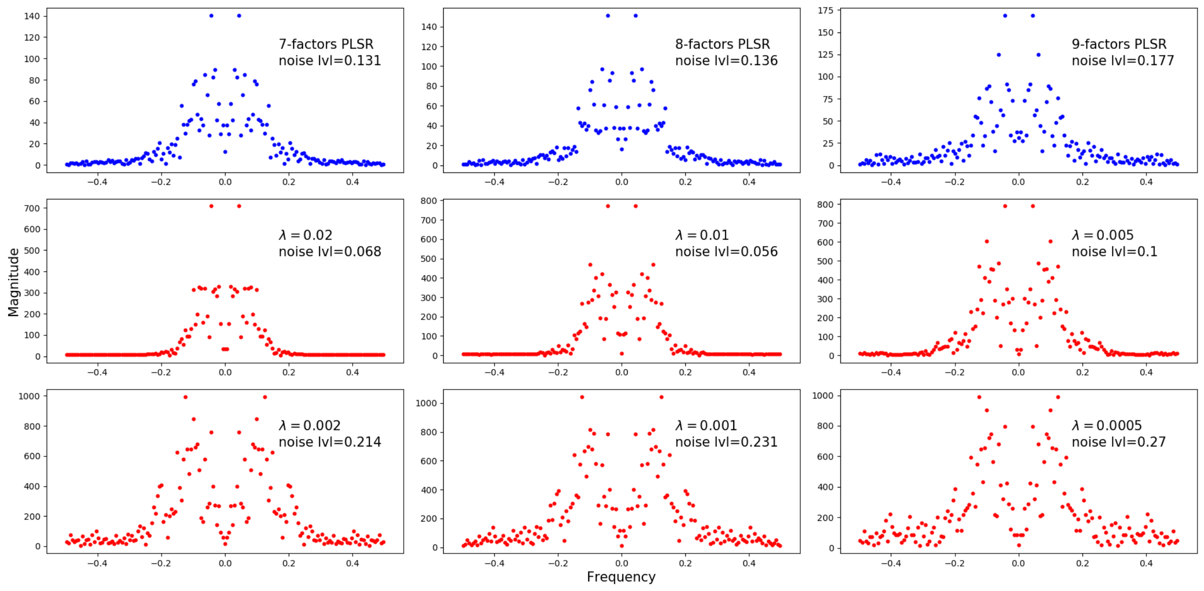


Figure 10: Fourier transform on regression coefficients of different PLSR (blue) and CNN models(red). Noise level was calculated by the ratio of magnitude of high frequency (>0.3) components.

λ/PLS factors	RMSECV	RMSEP	SEP	Noise level
7-factor PLSR	0.354	0.321	0.320	0.131
8-factor PLSR	0.329	0.291	0.290	0.136
9-factor PLSR	0.310	0.286	0.286	0.177
0.05	0.419	0.365	0.364	0.045
0.02	0.371	0.325	0.325	0.068
0.01	0.315	0.285	0.284	0.056
0.005	0.254	0.230	0.229	0.100
0.002	0.233	0.223	0.222	0.214
0.001	0.214	0.222	0.222	0.231
0.0005	0.205	0.215	0.215	0.270

Table 3: λ selection in cross validation

Some discussions on the results:

1. We choose the model with $\lambda = 0.005$. Further decreasing regularization on the network does not significantly improve RMSECV, but doubles the noise level.

2. Compared to the 8-factor PLSR model, we can see that the $\lambda = 0.005$ CNN model has a slightly lower noise level, but a significantly better performance. It is also worth noting the $\lambda = 0.01$ CNN model has similar RMSEP and SEP with the 8-factor PLSR model, but a much smaller noise level on the regression coefficients. From figure 9 we can see the CNN model with $\lambda = 0.01$ is very smooth and clean.

3. The amount of regularization added to a CNN model is dependent on the spectral quality, the target property and the structure of the neural network. When we use the same spectroscopic system and CNN architecture, the regularization parameter should roughly scale with the square of the target property concentration. This may help us assign initial search range for λ.

5.1.6 Dropout ratio

Dropout is another way to add regularization to a CNN model. Normally dropout is only employed after FC layers. In this test we add an additional dropout layer after FC1, FC2 and FC3. By tuning the dropout rate we found that the regression coefficients become less noisy, at the price of losing fitting ability. Refer to figure 11 and table 4.

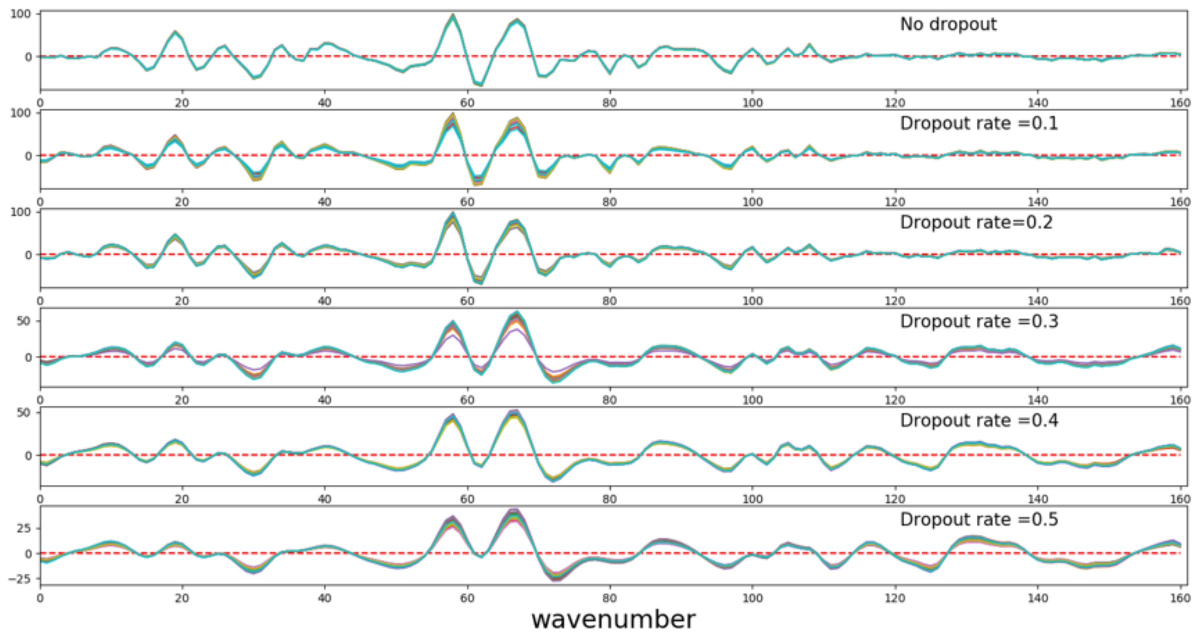


Figure 11: Regression coefficients trained with different dropout rate

Dropout rate	RMSECV	RMSEP
0	0.254	0.230
0.1	0.277	0.231
0.2	0.291	0.248
0.3	0.336	0.315
0.4	0.359	0.289
0.5	0.383	0.349

Table 4: Effect of dropout in a CNN calibration

From the results we can observe that the impact of dropout is very gentle, especially when dropout rate is small. By increasing the dropout rate the cross-validation error continuously goes up, due to reduced usable features in the network, but the out-of-sample prediction error can be almost consistent.

Based on this dataset we find that the benefits of dropout are not very obvious. In general we can just set the dropout rate to 0. When a robust and smooth calibration model is desired, we recommend using dropout as an additional regularization tool, in addition to L2 regularization.

5.2 Experiment 2: Application on large datasets and scalability

Here we show another example of using the CNN method for NIR calibration. The experiment was done on dataset 2. The CNN model was trained on a very large training set (6987 samples), then evaluated on a separate test set. The samples in the test set were measured under the same spectroscopic system. However, training samples and test samples have different varieties, origins, physical conditions, etc. We see from both the PLSR and the CNN calibration models, the training and the cross-validation errors are much smaller than the test error, which indicates the structures of the two datasets are different. However, we observe that a properly regularized CNN model is more robust and more accurate on the test set.

For the PLSR calibration, we applied the second order polynomial detrend + SNV to the raw spectra. Cross validation on the training set suggests using 10 PLS factors. RMSEP and SEP on the separate test set are 0.094% and 0.088% correspondingly.

We use the same CNN structure as introduced in experiment 1, refer to table 1 for details. Raw spectra were normalized in column axis before feeding to the CNN model. ELU activation function was employed. During the training process, we fixed the batch size at 256, and learning rate at 0.01 (corresponding to equation 8). Similar to model selection in PLSR, we use cross-validation to choose regularization parameter. No dropout is applied. A summary of cross validation is shown in table 5.

λ	RMSECV	RMSEP	SEP	Noise level
... ...				
6×10^{-4}	0.056	0.093	0.081	0.028
4×10^{-4}	0.053	0.089	0.077	0.040
2×10^{-4}	0.051	0.086	0.076	0.015
1×10^{-4}	0.048	0.085	0.075	0.036
8×10^{-5}	0.045	0.091	0.077	0.081
6×10^{-5}	0.058	0.083	0.074	0.122
... ...				

Table 5: CNN model selection on dataset 2: fine tuning on λ.

In the following discussion, we set the regularization parameter at 1×10^{-4}, since it has the best RMSECV/noise level trade-off. First we carried a paired hypothesis test to compare the prediction accuracy between the PLSR model (with 10 factors) and the CNN model. Results shows that 95% confidence interval of the true ratio of the SEP is: $PLSR/CNN = (1.198, 1.325)$. Since this interval does not include 1, we can conclude that the prediction error of the CNN model is significantly smaller than the PLSR model. Refer to table 6 for a brief summary:

Model	RMSECV	RMSEP	SEP	Noise level
PLSR	0.052	0.094	0.088	0.165
CNN	0.048	0.085	0.075	0.036
CI			(1.178, 1.314)	

Table 6: Comparison of the two calibration methods on dataset 2. PLSR: 10 factors, 2^{nd} order polynomial detrend +SNV; CNN: $\lambda = 10^{-4}$, learning rate=0.01, no dropout. CI stands for the 95% confidence interval for the ratio of SEP.

In figure 12, we compare the prediction — reference plots and the regression coefficients of the two calibration methods. We can clearly observe that while the CNN model is more precise than the PLSR model, the regression coefficients of the CNN is still smoother than that of the PLSR. Specifically, the noise level of the CNN model (0.036) is much smaller than that of the PLSR model (0.165). This indicates that the enhanced performance does not come from overfitting on the dataset we used.

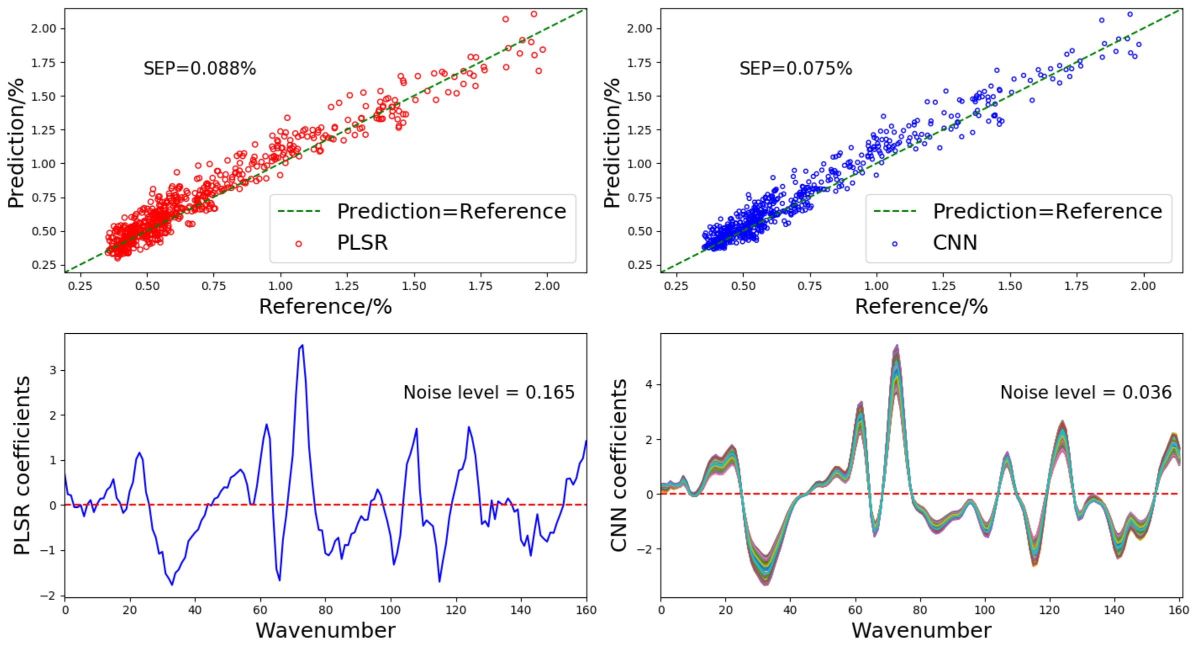


Figure 12: Comparison of the regression coefficients between the PLSR model and the CNN model. PLSR : 10 factors, detrend2 + SNV applied to the raw spectra. CNN model: $\lambda = 0.0001$, configuration refer to table 1

We also investigated the automatic preprocessing done by the CNN model, refer to figure 12. Different from experiment 1, the automatic preprocessing on dataset 2 is not based on the first derivative. It appears that the outputs of the convolutional layer[1] are skewed and rescaled raw spectra.

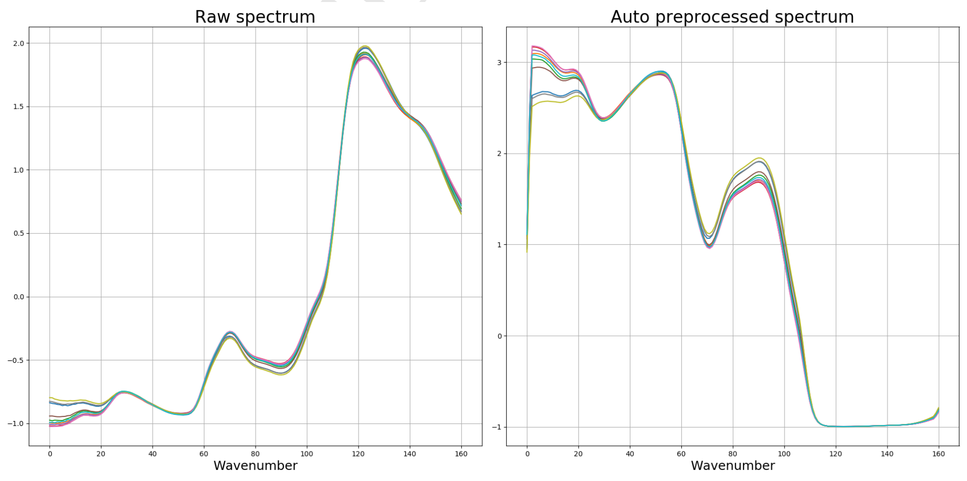


Figure 13: Automatic preprocessing by the convolutional layer. We randomly draw 10 spectra from the training set (on the left) and plot the outputs of the convolutional layer (on the right)

.

[1]when we say "the outputs of the convolutional layer", we mean the outputs of the activation function attached to the convolutional layer.

5.3 Experiment 3: Application on small datasets

Our proposed method can be applied to small datasets. However, the risk is that when the number of training samples is not adequate, it is very likely we are not able to get the best model. Here we present an example on dataset 3.

We use the same CNN architecture as introduced in experiment 1 and 2. For dataset 3, we have no information about the instrument and the commodity features. In practice, some neural network structures should be adjusted accordingly (for example, filter width in the convolutional layer). For simplicity, here we just use the same CNN structure as before.

In the PLSR model, we use the second order polynomial detrend + SNV as preprocessing methods. Cross-validation suggests using 8 PLS factors. The final prediction accuracy on the test set is: RMSEP= 0.425%; SEP=0.411%.

In the CNN model, we recommend that the users adopt second-order optimizers for small datasets. For example, in this experiment we use L-BFGS optimizer, which works very well in full batch, deterministic mode (i.e., we do not draw a batch of samples from the training set in each update, but use the whole dataset every time). The benefit of such kind of optimizers is a much faster convergence. We set the regularization at 0.003 by cross-validation. The final prediction accuracy on the test set is: RMSEP=0.420; SEP=0.414. a summary of the results is showed in table 7.

Model	RMSECV	RMSEP	SEP	Noise level
PLSR	0.439	0.425	0.411	0.278
CNN	0.425	0.420	0.414	0.080
CI			(0.981,1.103)	

Table 7: Comparison of the two calibration methods on dataset 3.

The regression coefficients are shown in figure 14. It is obvious that while the prediction performances of the CNN model and the PLSR model are quite close, the noise level in the CNN model is much smaller than in the PLSR model. In addition, from the figure we can observe that the nonlinearity is more apparent than in dataset 1 or dataset 2, namely the variance of the regression coefficient curves from the CNN model is bigger. This is due to the diversity of whole grains (surface condition, hardness, color, etc.). For flour, the samples were thoroughly homogenized before being measured, so the nonlinearity only comes from the NIR - target constituent correlation. Obviously for whole grains physical conditions of the grains also contribute to a huge part of nonlinearity in the calibration model.

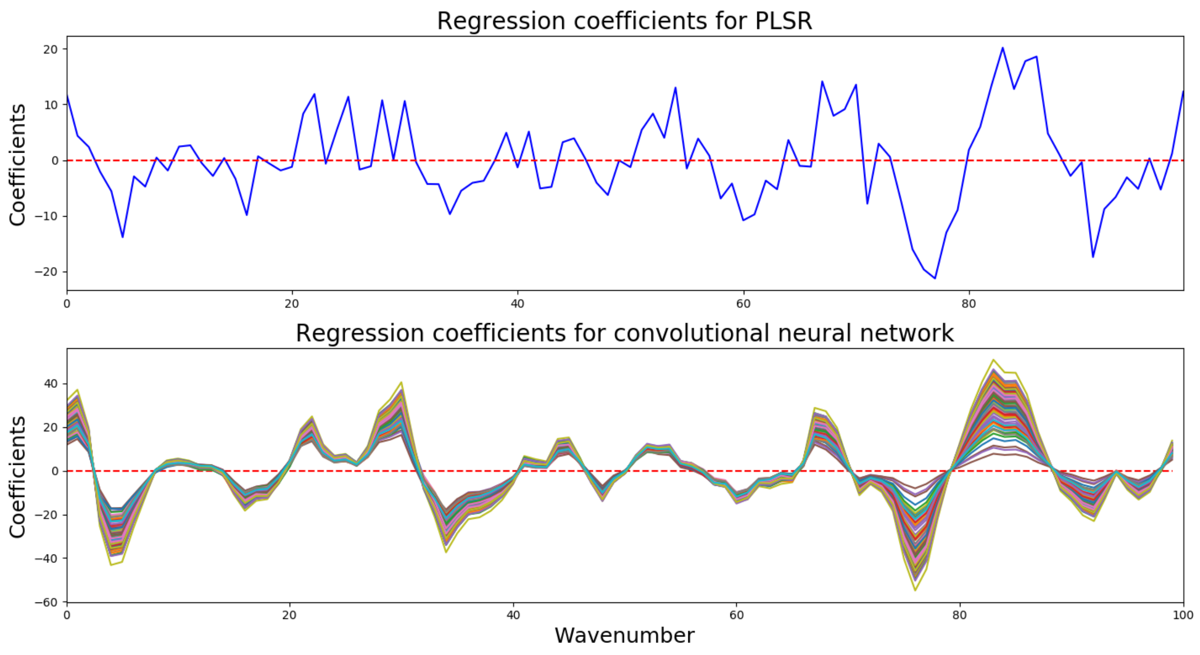


Figure 14: Comparison of the regression coefficients of the two calibration methods on dataset 3

Another issue we found is that ReLU is not suitable for the small dataset. When using RelU for activation, we can easily get a bad local optimum. Refer to figure 15. The calibration model is truncated for part of samples in the test set. We think it is related to the sparse feature of the ReLU function. When the inputs to the ReLU function are negative, the outputs are always zero, which means such neurons are "dead". If all the neurons in any of the layers are all "dead" then the whole network is not active anymore. For small dataset there is a certain risk that the weights are not properly tuned, especially for some extreme inputs.

As introduced in previous discussion, the major advantage of ReLU over ELU is low in computational cost. For a small dataset, when we use the second order full batch optimizer, convergence speed is not a issue anymore so we can always use ELU to avoid such bad local optima.

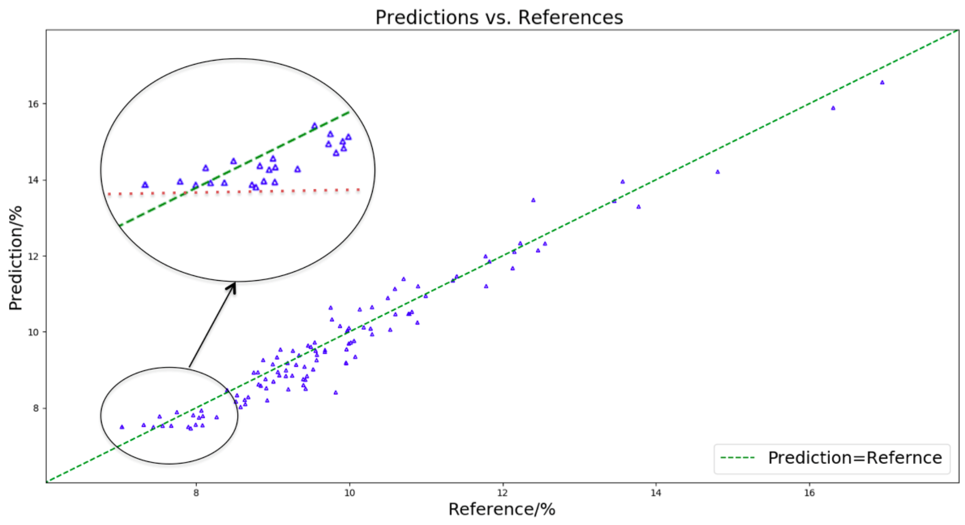


Figure 15: When using ReLU as the activation function, there is a "dead" region for samples low in constituent concentration. Prediction values are truncated.

6 Conclusions

We have discussed how to use convolutional neural networks for NIR calibration. We have done systematic tests on three different NIR datasets, to show the advantages of the CNN method over the PLSR method. A summary of our key results:

1. The convolutional layer can play a role of spectral preprocessor. This kind of layer can be automatically trained, which saves effort in selecting preprocessing methods.

2. By visualizing the regression coefficients and calculating the high frequency components in the Fourier domain, we compared the introduced CNN method with the PLSR method. We have shown that the CNN models are not only more accurate than the PLSR models (smaller RMSEP and SEP), but also less noisy and more robust

3. We have systematically compared different choices of hyperparameters in the CNN models, including activation function, learning rate, regularization parameter, and dropout rate. All the hyperparameters in the model can be selected by cross validation. We also recommended some default initial configuration for common applications.

4. We have tested the proposed method on 3 different NIR datasets, which have different sizes. We have illustrated the CNN methods can be used for general NIR calibrations, even when the dataset size is not that large.

Acknowledgements

We wish to express our gratitude to Bühler AG for supplying high-quality large datasets for the study. We also want to thank Juste Hahne and Martin Heine for demonstration and organization of the datasets.

References

[1] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," *International Journal of Computer Vision*, vol. 88, no. 2, pp. 303–338, 2010.

[2] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, *et al.*, "Imagenet large scale visual recognition challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015.

[3] R. Collobert and J. Weston, "A unified architecture for natural language processing: Deep neural networks with multitask learning," in *Proceedings of the 25th international conference on Machine learning*, pp. 160–167, ACM, 2008.

[4] Y. Zheng, Q. Liu, E. Chen, Y. Ge, and J. L. Zhao, "Time series classification using multi-channels deep convolutional neural networks," in *International Conference on Web-Age Information Management*, pp. 298–310, Springer, 2014.

[5] J. Zupan and J. Gasteiger, "Neural networks: A new method for solving chemical problems or just a passing phase?," *Analytica Chimica Acta*, vol. 248, no. 1, pp. 1–30, 1991.

[6] P. A. Jansson, "Neural networks: An overview," *Analytical chemistry*, vol. 63, no. 6, pp. 357A–362A, 1991.

[7] J. A. Burns and G. M. Whitesides, "Feed-forward neural networks in chemistry: mathematical systems for classification and pattern recognition," *Chemical Reviews*, vol. 93, no. 8, pp. 2583–2601, 1993.

[8] D. A. Cirovic, "Feed-forward artificial neural networks: applications to spectroscopy," *TrAC Trends in Analytical Chemistry*, vol. 16, no. 3, pp. 148–155, 1997.

[9] F. Marini, R. Bucci, A. Magrì, and A. Magrì, "Artificial neural networks in chemometrics: History, examples and perspectives," *Microchemical journal*, vol. 88, no. 2, pp. 178–185, 2008.

[10] V. O. Santos, F. C. Oliveira, D. G. Lima, A. C. Petry, E. Garcia, P. A. Suarez, and J. C. Rubim, "A comparative study of diesel analysis by ftir, ftnir and ft-raman spectroscopy using pls and artificial neural network analysis," *Analytica Chimica Acta*, vol. 547, no. 2, pp. 188–196, 2005.

[11] C. Cui and T. Fearn, "Comparison of partial least squares regression, least squares support vector machines, and gaussian process regression for a near infrared calibration," *Journal of Near Infrared Spectroscopy*, vol. 25, no. 1, pp. 5–14, 2017.

[12] C. Cui and T. Fearn, "Hierarchical mixture of linear regressions for multivariate spectroscopic calibration: An application for nir calibration," *Chemometrics and Intelligent Laboratory Systems*, 2018.

[13] S. Malek, F. Melgani, and Y. Bazi, "One-dimensional convolutional neural networks for spectroscopic signal regression," *Journal of Chemometrics*, 2017.

[14] E. J. Bjerrum, M. Glahder, and T. Skov, "Data augmentation of spectral data for convolutional neural network (cnn) based deep chemometrics," *arXiv preprint arXiv:1710.01927*, 2017.

[15] T. Næs, T. Isaksson, T. Fearn, and T. Davies, *A user friendly guide to multivariate calibration and classification.* NIR publications, 2002.

[16] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

[17] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, "Efficient backprop," in *Neural networks: Tricks of the trade*, pp. 9–48, Springer, 2012.

[18] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[19] D. A. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (elus)," *arXiv preprint arXiv:1511.07289*, 2015.

[20] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting.," *Journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[21] S. Wager, S. Wang, and P. S. Liang, "Dropout training as adaptive regularization," in *Advances in neural information processing systems*, pp. 351–359, 2013.

[22] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[23] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[24] T. Schaul, I. Antonoglou, and D. Silver, "Unit tests for stochastic optimization," *arXiv preprint arXiv:1312.6055*, 2013.

[25] P. Geladi and B. R. Kowalski, "Partial least-squares regression: a tutorial," *Analytica chimica acta*, vol. 185, pp. 1–17, 1986.

[26] E. Pitman, "A note on normal correlation," *Biometrika*, vol. 31, no. 1/2, pp. 9–12, 1939.

[27] T. Fearn, "Comparing standard deviations," *NIR news*, vol. 7, no. 5, pp. 5–6, 1996.

[28] D. K. Pedersen, H. Martens, J. P. Nielsen, and S. B. Engelsen, "Near-infrared absorption and scattering separated by extended inverted signal correction (eisc): analysis of near-infrared transmittance spectra of single wheat seeds," *Applied spectroscopy*, vol. 56, no. 9, pp. 1206–1214, 2002.

[29] A. Krizhevsky, "One weird trick for parallelizing convolutional neural networks," *arXiv preprint arXiv:1404.5997*, 2014.

- We have discussed how to use the convolutional neural networks for spectroscopic calibration. The comparison between the CNN method and the PLSR method was done on 3 different NIR datasets.
- The convolutional layer in the CNN model can perform automatic spectral preprocessing.
- We have visualized the models and evaluated the noise level of different methods. We have shown that the CNN method is more precise, less noisy and more robust than the PLSR method.
- We have systematically compared different choices of hyperparameters in the CNN models, including activation function, learning rate, regularization parameter, and dropout rate