MACHINE LEARNING IN CONFOCAL LASER MICROSCOPY AND

SPECTROSCOPY

By

Yinchuan Yu

A dissertation submitted in partial fulfillment of
the requirements for the degree of

DOCTOR OF PHILOSOPHY

WASHINGTON STATE UNIVERSITY
Department of Physics and Astronomy

MAY 2022

To the Faculty of Washington State University:

The members of the Committee appointed to examine the dissertation of YINCHUAN YU find it satisfactory and recommend that it be accepted.

_____

Matthew D. McCluskey, Ph.D., Chair

_____

Mark G. Kuzyk, Ph.D.

_____

Yi Gu, Ph.D.

ACKNOWLEDGMENTS

MACHINE LEARNING IN CONFOCAL LASER MICROSCOPY AND SPECTROSCOPY

Abstract

by Yinchuan Yu, Ph.D.
Washington State University
May 2022

Chair: Matthew D. McCluskey

Confocal laser scanning microscopy (CLSM) is a preferred method for obtaining optical images with submicron resolution. Replacing the pinhole and detector of a CLSM with a digital camera (CCD or CMOS) has the potential to simplify the design and reduce cost. However, the relatively slow speed of a typical camera results in long scans. To address this issue, in the present investigation a microlens array (MLA) was used to split the laser beam into 48 beamlets that are focused onto the sample. In essence, 48 pinhole-detector measurements were performed in parallel. Images obtained from the 48 laser spots were stitched together into a final image.

Photoluminescence (PL) spectroscopy is a non-destructive optical method that is widely used to characterize semiconductors. In the PL process, a substance absorbs photons and emits light with longer wavelengths. This paper discusses a method for identifying substances from their PL spectra using machine learning, a technique that is efficient in making classifications. Neural networks were constructed by taking simulated PL spectra as the input and the identity of the substance as the output. Six different semiconductors were chosen as categories: gallium oxide (Ga_2O_3), zinc oxide (ZnO), gallium nitride (GaN), cadmium sulfide (CdS), tungsten

disulfide (WS_2) and cesium lead bromide ($CsPbBr_3$). The developed algorithm has a high accuracy (>90%) for assigning a substance to one of these six categories from its PL spectrum.

With an XY stage, a CLSM can scan a large area on a sample. Adjusting the height of the objective is necessary which made the laser beam could focus on the sample surface. However, if the surface of the sample is not flat, the laser spot will go in and out of focus, causing bad scanning results. Deep learning especially convolutional neural networks is an efficient way to treat images. It shows its success in the field of object detection, image classification, face recognition, etc. The deep learning techniques were used to design a model that predicts the out-of-focus distance with the image of laser spot. The model can develop to a system that could automatically focusing the CLSM in real time.

TABLE OF CONTENTS

APPENDIX

LIST OF FIGURES

LIST OF TABLES

CHAPTER ONE: INTRODUCTION TO DEEP LEARNING

Deep learning with neural networks has become an efficient solution for problems such as computer vision, natural language processing, and speech recognition[1]. It simulates how human brain makes decisions. In this chapter, I discuss the development of deep learning and introduce the main concepts.

Artificial Neural Networks

In 1943, McCulloch and Pitts[2] introduced a neuron model in which, given a weighted input, the neuron can be activated or inactivated (Fig. 1.1).
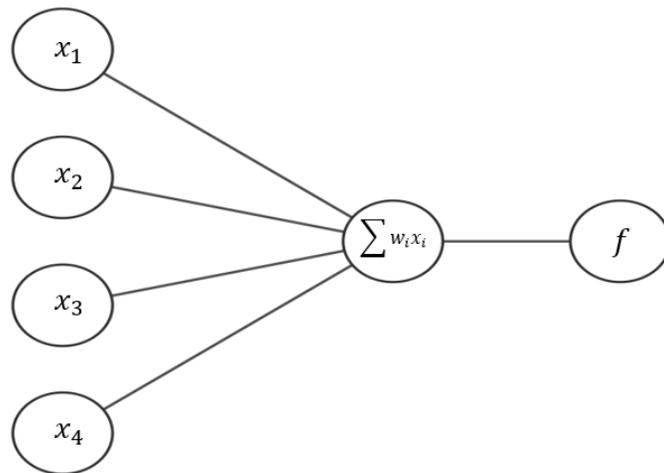
Fig. 1.1. McCulloch and Pitts model. After the dot product is taken of the input x and weight parameter vector w, the result passes through an activation function f.

The model is expressed as:

$$y = f\left(\sum w_i x_i\right) \tag{1.1}$$

The activation function f is a threshold function. If the input is greater than the threshold, the output equals 1; otherwise, it equals 0. From the threshold back to the input vector space, a boundary is formed, which is called a hyperplane. The model can only solve classification problems which are linearly separable.

To solve the nonlinear separable problem, hidden layers are introduced. Instead of directly pointing to the output, each dot product is treated as one cell of the next layer. Fig. 1.2 shows a two hidden layer neural network.
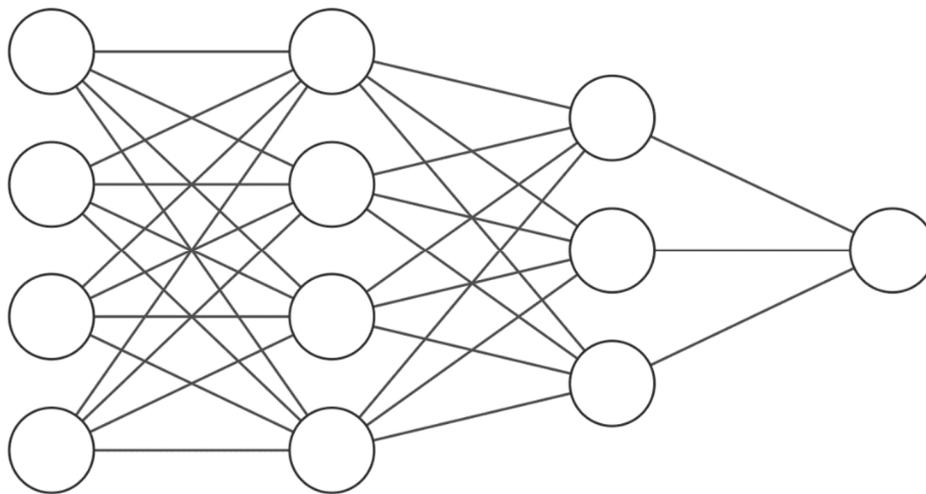
Fig. 1.2. Graphic representation of a two hidden layer neural network. The input is a 4-element vector, while the first and second hidden layers are 4-element and 3-element vectors respectively.

An element in each layer is a linear combination of all elements in the previous layer, followed by passing through a nonlinear activation function. The nonlinear activation function, discussed in the next section, is necessary to break the linearity.

In neural networks, each layer is treated as a vector. Therefore, the weighted parameters between two layers can be treated as a matrix. The propagation between two layers is shown in Eq. (1.2) and Eq. (1.3):

$$z_l = W_l \cdot a_{l-1} + b_l \qquad (1.2)$$

$$a_l = \sigma(z_l) \qquad (1.3)$$

where a_l is the vector of the lth layer, W_l is a parameter matrix with rows equal to the number of elements in layer l and columns equal to the number in layer l–1, and b_l is a bias vector with same number of elements as layer l. σ is the nonlinear activation function, which will be discussed later. Without the activation function, no matter how many hidden layers, the final cell is still a linear combination of the original input. For example, if there is no activation function, then we have $a_l = z_l$, thus we can get:

$$a_l = W_l \cdot a_{l-1} + b_l \qquad (1.4)$$

$$a_{l+1} = W_{l+1} \cdot a_l + b_{l+1} \qquad (1.5)$$

Substitute Eq. (1.4) into Eq. (1.5):

$$a_{l+1} = W_{l+1} \cdot (W_l \cdot a_{l-1} + b_l) + b_{l+1}$$

$$= (W_{l+1}W_l) \cdot a_{l-1} + (W_{l+1} \cdot b_l + b_{l+1}) \qquad (1.6)$$

From Eq. (1.6), the two layers are combined as one where the weighted matrix is $W_{l+1}W_l$ and bias vector is $W_{l+1} \cdot b_l + b_{l+1}$.

Activation Function

Traditionally, the two main nonlinear activation functions are sigmoid function [Eq. (1.7)] and hyperbolic tangent function [Eq. (1.8)].

$$f(x) = \frac{1}{1+e^{-x}} \tag{1.7}$$

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{1.8}$$
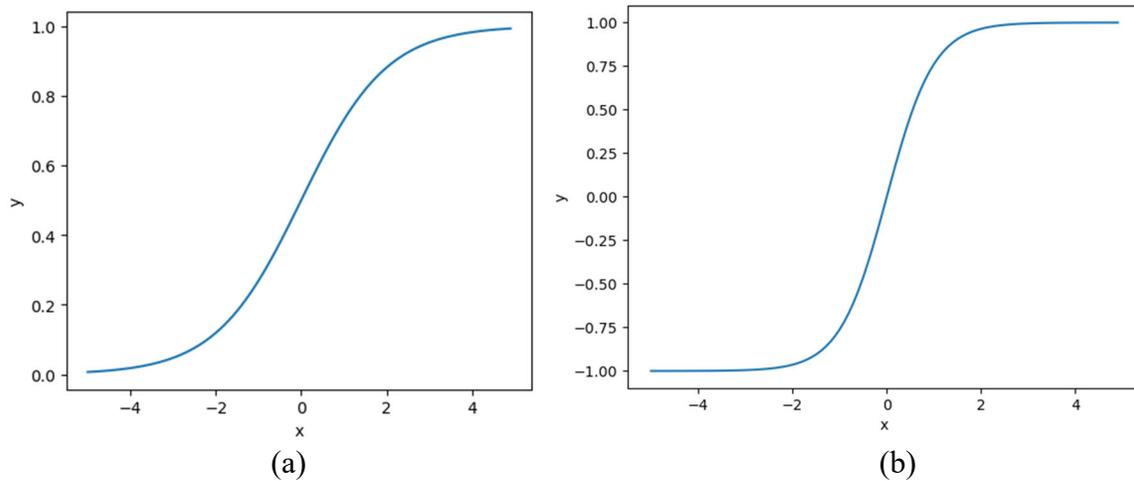
Fig. 1.3. (a) Plot of the sigmoid function. (b) Plot of the hyperbolic tangent function.

The sigmoid function has a value between 0 to 1, which is useful in representing probability. Meanwhile the hyperbolic tangent function has a value between -1 to 1. It is easier to train and

often has a better performance compared to the sigmoid function. However, a problem is that both functions have only a limited sensitive range. When the input of the function is either too small or too large, the function will have a very small gradient value. With the development of hardware, computers could train deeper and deeper networks, which caused the gradient at the earlier layers to vanish and thus failed to update the weighted parameters[3].

In order to train deep neural networks, an activation function should be nonlinear but perform similar as a linear function. The rectified linear activation unit (ReLU) [Eq. (1.9)] was introduced[4]:

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases} \tag{1.9}$$

The ReLU function is a linear function for positive inputs, with a gradient equal 1. It neglects negative inputs, breaking the linearity. Therefore, ReLU makes it possible to train very deep neural networks.

The output layer is varied depending on the task type. For a regression problem, there is usually no need for an activation function. For a classification problem, if the class number is two, the output layer is usually a single value, and the sigmoid function is used. The result is always between 0 to 1, which can represent the probability of the instance of data belonging to the first class. If there is more than one class, the output layer should have same number of elements as the number of classes, and the softmax function is used as the activation function:

$$f(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}} \tag{1.10}$$

where the K is the number of classes. Each element of the output vector represents the probability belonging to the corresponding class.

Loss Function

After the result is outputted by the neural networks, a function is needed to describe the difference between the estimated and true value of the input data. This function is called the loss function. If the task is a regression problem, which should predict a certain value or vector, then the loss function typically used is the mean absolute error (L1) [Eq. (1.11)] or mean squared error (L2) [Eq. (1.12)].

$$L1 = \frac{1}{m} \sum_{i=1}^{m} |y_i - \tilde{y}_i| \tag{1.11}$$

$$L2 = \frac{1}{m} \sum_{i=1}^{m} (y_i - \tilde{y}_i)^2 \tag{1.12}$$

where y_i represents the true value and $\tilde{y}_i$ is the output value of the networks. The value m is the total number of training examples in the batch.

Cross-entropy loss is used in classification tasks. For a binary classification task, the cross-entropy loss is given by Eq. (1.13):

$$l = -[y ln(\tilde{y}) + (1 - y) \ln(1 - \tilde{y})] \tag{1.13}$$

The value of y is either be 0 or 1, and the $\tilde{y}$ is between 0 to 1. If $y = \tilde{y}$, the prediction is perfect, and the loss will be 0. If $\tilde{y}$ is different from y, the loss increases rapidly.
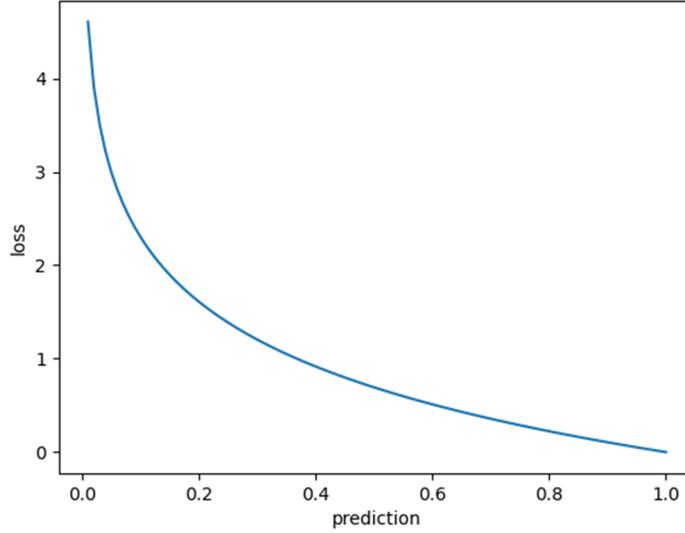
Fig. 1.4. The loss value for a true label equal 1 instance as a function of the predicted probability from 0 to 1.

If the number of classes is more than two, the loss function is in form of Eq. (1.14):

$$l = -\sum_{c=1}^{\#\ class} y^c \ln(\tilde{y}^c) \qquad (1.14)$$

where the superscript c stands for the vector's c^{th} element. The binary cross-entropy loss is equivalent to the Eq. (1.14) when class number equals 2. Taking average of the whole training batch, the two loss functions will become Eq. (1.15) and Eq. (1.16):

$$L = -\frac{1}{m}\sum_{i=1}^{m}[y_i ln(\tilde{y}_i) + (1 - y_i)\ln(1 - \tilde{y}_i)] \qquad (1.15)$$

$$L = -\frac{1}{m}\sum_{i=1}^{m}\sum_{c=1}^{\#\ class} y_i^c \ln(\tilde{y}_i^c) \qquad (1.16)$$

Backward Propagation

David Rumelhart et al.[5] in 1986 developed a backward propagation algorithm that works fast in neural networks learning. As discussed in the previous section, the input passes through all the weighted matrices, which gives an estimated value. As all the parameters in the weighted matrices are randomly initialized, there is no expectation for a good approximation in the beginning, and the loss is large. The learning process tries to minimize the loss by updating the parameters. In general, the method is to find the gradient of the loss function over the parameter space. Each parameter is updated simultaneously by subtracting a value proportional to the gradient:

$$\theta := \theta - \alpha \cdot \nabla_\theta L \qquad (1.17)$$

where α is the learning rate, which needs to be set to an appropriately small number, and θ represents parameters (weighted matrices, bias values, etc.).

I will use W_l^{ij} to denote the element in the ith column, j^{th} row of the weight matrix between the $(l\text{-}1)^{th}$ and l^{th} layer. It connects the ith element in the $(l\text{-}1)^{th}$ layer to the jth element in the lth layer; b_l^j denotes the bias value used to get the jth element in the lth layer; a_l^k denotes the kth element of the lth layer; z_l^k represent the value before applying the activation function of a_l^k. Using the network in Fig. 1.2 as an example, we assume the loss function is a cross-entropy loss. The activation function for the output layer is sigmoid and for the hidden layers is ReLU.
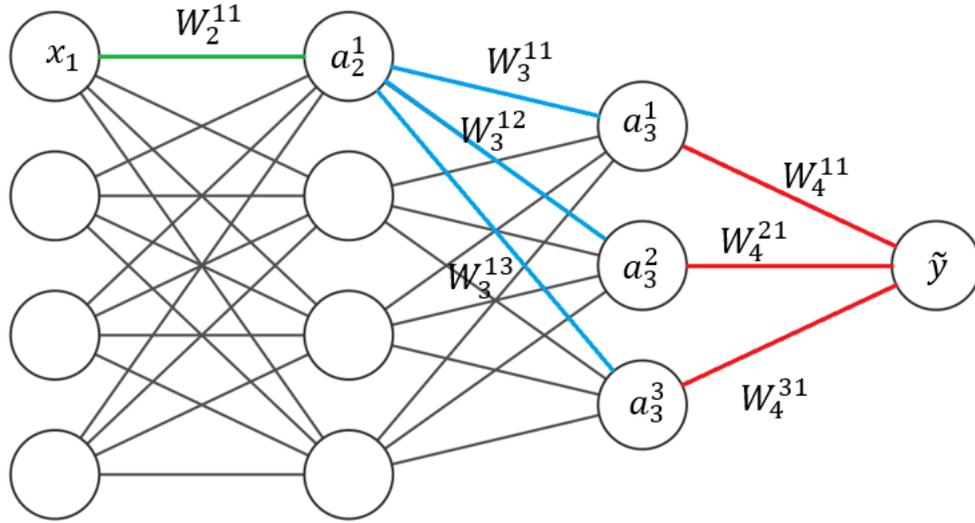
Fig. 1.5. Illustration of the steps that determine how the input's first

element affects the output result. Some of the parameters and layers are labeled.

$\tilde{y}$ is the output, and the true value is y. From the cross-entropy loss function, we calculate the partial derivative of the loss over $\tilde{y}$:

$$\frac{\partial l}{\partial \tilde{y}} = -[y \frac{\partial \ln(\tilde{y})}{\partial \tilde{y}} + (1-y) \frac{\partial \ln(1-\tilde{y})}{\partial \tilde{y}}]$$

$$= -[y \frac{1}{\tilde{y}} + (1-y) \frac{-1}{1-\tilde{y}}] \qquad (1.18)$$

Since $\tilde{y}$ is equal to the sigmoid of z_4^1, then by the chain rule:

$$\frac{\partial l}{\partial z_4} = \frac{\partial l}{\partial \tilde{y}} \frac{\partial \tilde{y}}{\partial z_4}$$

$$= -[y \frac{1}{\tilde{y}} + (1-y) \frac{-1}{1-\tilde{y}}][\tilde{y}(1-\tilde{y})]$$

$$= \tilde{y} - y \qquad (1.19)$$

As all the z_l^j are linear combinations of a_{l-1}^i ($z_l^j = \sum_i W_l^{ij} a_{l-1}^i + b_l^j$), we obtain:

$$\frac{\partial z_l^j}{\partial a_{l-1}^i} = W_l^{ij} \tag{1.20}$$

$$\frac{\partial z_l^j}{\partial W_l^{ij}} = a_{l-1}^i \tag{1.21}$$

$$\frac{\partial z_l^j}{\partial b_l^j} = 1 \tag{1.22}$$

Since the activation function used are ReLU, if the $a_l^i > 0$, $\frac{\partial a_l^i}{\partial z_l^i} = 1$, else $\frac{\partial a_l^i}{\partial z_l^i} = 0$.

Let dW_l, da_l and dz_l represent the matrix or vector, in which each element is the partial derivative of the loss over the corresponding element of W_l, a_l and z_l. Then we obtain:

$$dz_4 = \tilde{y} - y \tag{1.23}$$

$$dz_l^i = \begin{cases} da_l^i & \text{if } a_l^i > 0 \\ 0 & \text{if } a_l^i \leq 0 \end{cases} \quad l = 2,3 \tag{1.24}$$

$$db_l^i = dz_l^i \quad\quad\quad l = 2,3,4 \tag{1.25}$$

$$da_l^i = \sum_j W_{l+1}^{ij} \times dz_{l+1}^j \quad\quad l = 2,3 \tag{1.26}$$

$$dW_l^{ij} = a_{l-1}^i \times dz_l^j \quad\quad l = 2,3,4 \tag{1.27}$$

Expressing Eq. (1.25), Eq. (1.26) and Eq. (1.27) in the matrix and vector form,

$$db_l = dz_l \quad\quad\quad l = 2,3,4 \tag{1.28}$$

$$da_l = W_{l+1}^T dz_{l+1} \quad\quad l = 2,3 \tag{1.29}$$

$$dW_l = a_{l-1}dz_l{}^T \qquad\qquad l = 2, 3, 4 \qquad\qquad (1.30)$$

Backward propagating from the loss all the way to the first layer, all the gradient values can be calculated. Then, applying Eq. (1.17), the learning parameters are updated. A combination of a forward propagation and a backward propagation is called one iteration. With enough iterations, the parameters will converge to the values that fit the training data very well. The parameters can be used to make predictions on new data.

Initialization

Before starting to train a neural network model with any data, the weighted parameters are given initial values. If all the parameters are set to the same number, then every unit will have same influence on the loss, causing the parameters to have identical gradients. Therefore, parameters should be initialized with different values. Another problem is if all the weight matrices are slightly larger than the identity matrix, then the values of a hidden layer vector a_l will increase exponentially as l gets larger, which will cause the gradient to explode. At the opposite extreme, if all the matrices are slightly smaller than the identity matrix, as the network gets deeper, the gradient will vanish.

In 2010, Xavier et al.[6] introduced Xavier initialization. The weights are set following the normal distribution $G(0, \frac{1}{n_{l-1}})$, where the mean is set at 0, and the variance is one over the number of elements in the previous layer. Xavier initialization works well when the nonlinear activation functions are hyperbolic tangent and sigmoid, but not if the activation functions are ReLU. In 2015, He et al.[7] introduced He initialization, which is similar with Xavier initialization.

The difference is that the variance is $\frac{2}{n_{l-1}}$. Xavier initialization works well with ReLU activations.

Optimization Method

The way parameters update themselves with iterations affects the learning efficiency. The method shown in Eq. (1.17) is called gradient descent optimization. Gradient descent optimization has three forms. If the entire dataset is taken to calculate the loss and update the parameters, the optimization is called batch gradient descent. Batch gradient descent always converge to the global minimum or a local minimum for a non-convex surface. However, if the training dataset is large, the batch gradient will exhaust the computational resources and be very slow even when propagating a single iteration. In contrast, propagating an iteration with only one training example is called Stochastic Gradient Descent (SGD):

$$\theta := \theta - \alpha \cdot \nabla_\theta L(\theta, x_i, y_i) \tag{1.31}$$

SGD is much faster than batch gradient descent, but the learning curve is not as stable as the latter one. In between batch gradient descent and SGD there is the mini-batch gradient descent. The training set is separated into mini-batches, and each mini-batch has the same amount of training examples. The parameters will update one iteration with one mini-batch. In general, the larger the mini-batch, the more stable it is, but the learning process will be slower.

In low dimensional space, besides the global minimum, functions usually have many local minima. When it hits the local minimum, the gradient can no longer escape. However, in deep learning algorithms, there usually are thousands of parameters. Let's assume when a

12

gradient of the loss over a parameter is 0, the probability that it hit a convex point or concave

point is half-half. When the gradient of the loss over all parameters is 0, the chance that it is a

local minimum is:

$$p = \frac{1}{2^N} \qquad (1.32)$$

where N is the number of parameters. With thousands of parameters, the probability hitting a

local minimum can be neglected. Instead, part of the dimensions are at convex point and part of

the dimensions are at a concave point. This kind of position is called a "saddle point" (Fig. 1.6).
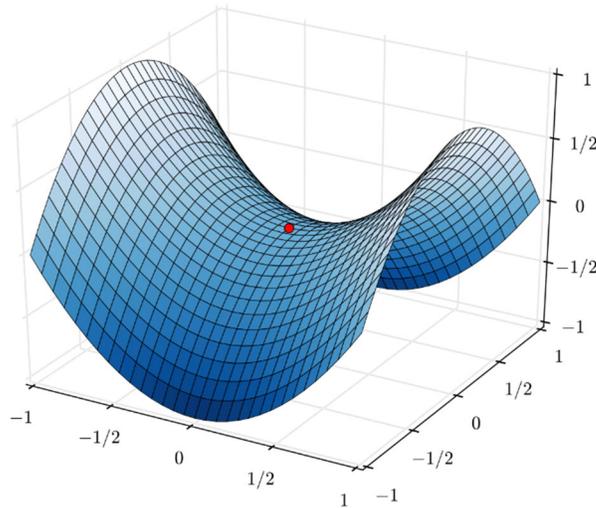
Fig. 1.6. Plot of a saddle point in a 2D function[8].

As all gradients near the saddle point are near 0. When the learning process hits the saddle point,

it will take a very long time to escape.

There are many optimization algorithms developed from gradient descent. The default

algorithm used in this work is called Adaptive Moment Estimation (Adam)[9]. Instead of only

using the gradient in the current stage to update the parameters, Adam keeps an exponentially decaying average of the gradients and the squared gradients in the past iterations (m_t and v_t). m_t and v_t are also called first moment and second moment. If we define $d\theta_t$ as $\frac{\partial L}{\partial \theta}$ at the tth iteration, m_t and v_t are given by:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1)d\theta_t \tag{1.33}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2)d\theta_t{}^2 \tag{1.34}$$

where β_1 and β_2 are constant set close to 1. By default, β_1 set to 0.9 and β_2 set to 0.999. After m_t and v_t are calculated, there is a step called bias correction:

$$\widehat{m}_t = \frac{m_t}{1-\beta_1{}^t} \tag{1.35}$$

$$\widehat{v}_t = \frac{v_t}{1-\beta_2{}^t} \tag{1.36}$$

Then the parameters are updated following the rule:

$$\theta_{t+1} = \theta_t + \alpha \frac{\widehat{m}_t}{\sqrt{\widehat{v}_t}+\varepsilon} \tag{1.37}$$

The constant ε is a small number that prevents the denominator being 0, 10^{-8} by default. Optimizers like Adam help escape from saddle points and usually coverage faster. When using mini-batches, Adam can also increase the stability of the learning process, since the previous gradients contribute to the moments.

Cross Validation

Cross validation is a method in which the researcher can observe the learning process and help it make decisions on choosing parameters[10]. There is a small dataset called the cross-validation set, which differs from the training dataset. During the iterations, the loss of the cross-validation set is also calculated. When the loss of cross-validation set starts to increase, or no longer decreases, the training should be stopped. The parameters resulting in the minimum loss of the cross-validation set should be accepted for the current model.
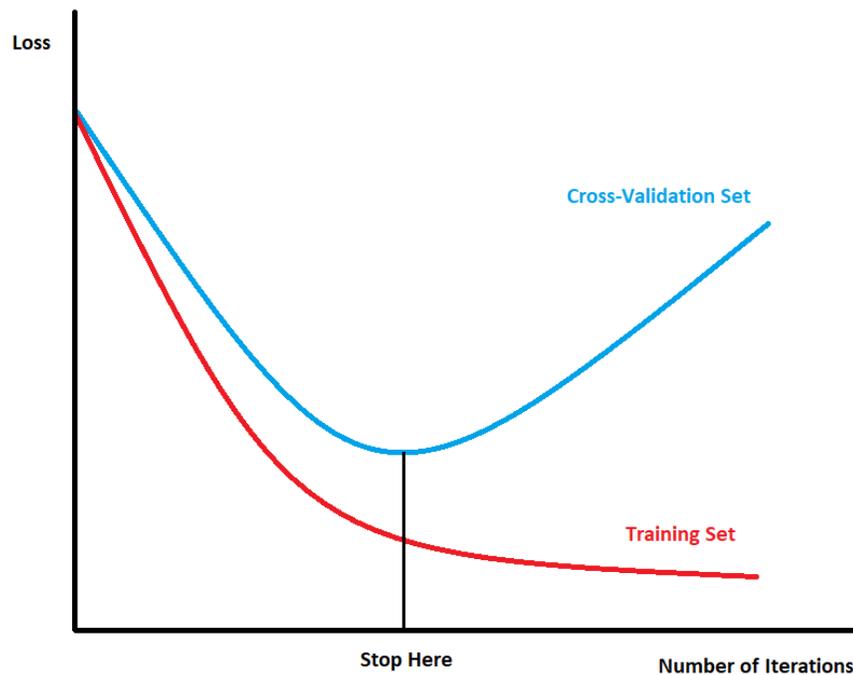
Fig. 1.7. A plot showing how the cross-validation loss may start to increase as the training loss keeps decreasing. An early stop is suggested by the cross-validation.

There are many untrainable parameters in the model such as number of layers, number of elements in each layer, and learning rate. These kinds of parameters are called hyper parameters.

15

The cross-validation set also helps to determine the most suitable combinations of hyper parameters. The group of hyper parameters that yields the least cross-validation loss is chosen.

Regularization

When the number of parameters is large, the function has enough degrees of freedom that the fitting line can correctly separate all the training examples. The performance on the training data might be almost perfect. However, when the function turns to additional unseen data, the prediction may be far from the truth. This phenomenon is called over-fitting, and the fitting function is said to have a high variance (Fig. 1.8c). At the opposite extreme, if the function cannot describe the training data very well, it is under-fitting or has a high bias (Fig. 1.8a).
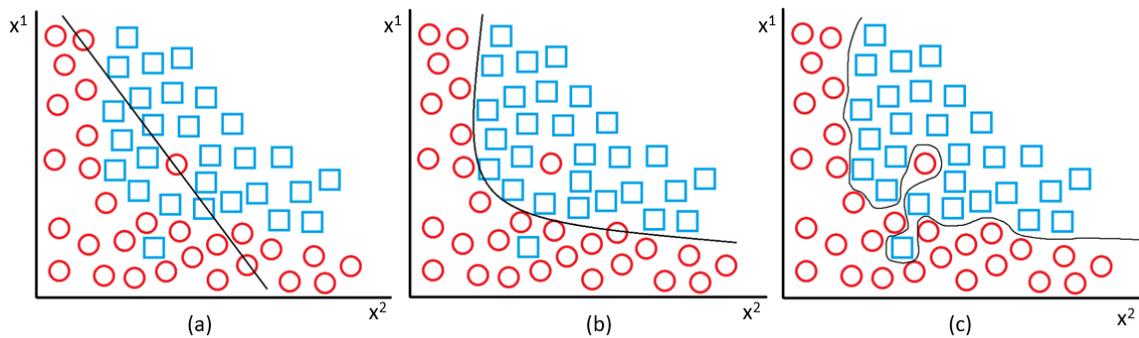
Fig. 1.8. Schematic plots showing different situations for a binary classification task. (a) Under fitting. (b) Appropriate fitting. (c) Over fitting. The red circles and blue squares are training examples in categories. The solid line describes the function that trying to separate the training examples into two classes.

Regularization is a technique that can reduce the error caused by over-fitting. There are three commonly used regularization methods: L1 regularization, L2 regularization and dropout regularization[11]. L1 regularization adds a term proportional to the sum of absolute value of all the weight parameters to the loss function [Eq. (1.38)]. L2 regularization adds a term proportional to the sum of squared weight parameters to the loss function [Eq. (1.39)].

$$L_{reg\_L1} = L_o + \lambda \sum_i |w_i| \tag{1.38}$$

$$L_{reg\_L2} = L_o + \lambda \sum_i w_i^2 \tag{1.39}$$

L_o is the original loss described in the previous section, and λ is a hyper parameter called the regularization constant. λ is a small number greater than 0.

Drop-out regularization is a different method. In each iteration during the training, some randomly picked elements in each layer and their relevant connections are removed. Then the network after the drop-out is used for a forward propagation to get the loss and a backward propagation to update the parameters. After one iteration, the neural networks are restored, and a new random drop-out takes place and trains another iteration. This process repeats until the training process is finished.
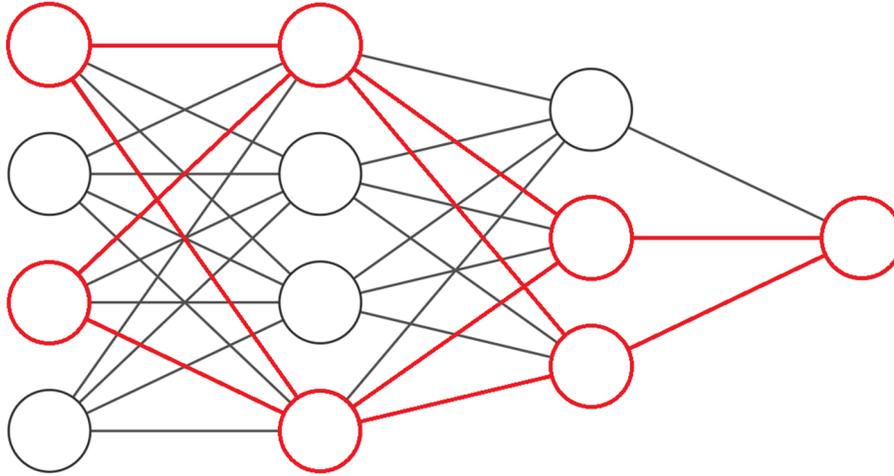
Fig. 1.9. Schematic of drop-out regularization. The red elements are the neurons left. The drop-out ratio in the first and second layers is 1/2, and in the third layer is 1/3.

By temporarily dropping neurons, it tends to average the effect of different combinations of thinned networks on the prediction. Therefore, using the full network after trained with drop-out to make predictions have less over-fitting. In addition, the drop-out regularization reduces the number of parameters used in each iteration. This reduces the computation resources needed, increasing the efficiency of training the network[12].

Convolutional Neural Networks

As discussed in the previous section, deep learning has been shown to be successful in classification and regression tasks. In general, tasks related to images are an important part of this field. The classical way to process images is to expand all pixel values and put them into a column vector. Then the column vector is used as an input that passes through the normal neural

networks to get a prediction. Since 2012, convolution neural networks have quickly shown an overwhelming advantage in the field related to images. A convolutional neural network usually contains convolutional layers, pooling layers, and fully connected layers, which will be discussed in the following sections.

Convolutional Layer

A convolutional layer is made from several convolutional filters that scan the whole image. The output of a convolutional layer is called the feature map. Each filter in the same layer should have the same size: $F \times F \times C$. F is the filter width, which is usually a small odd number such as 3 or 5, and C is the channel number of the image. For example, a gray scale image has only one channel, while an RGB color image has 3 channels. At each position, the sum of the product of each element in the filter and the corresponding pixel value in the image is calculated, then activated by an activation function (usually ReLU) [Eq. (1.40)].

$$v = ReLU\left(\sum_i w_i p_i + b\right) \tag{1.40}$$

When one filter finishes scanning the image, all the output values will form one channel of a new feature map, shown as Fig. 1.10.
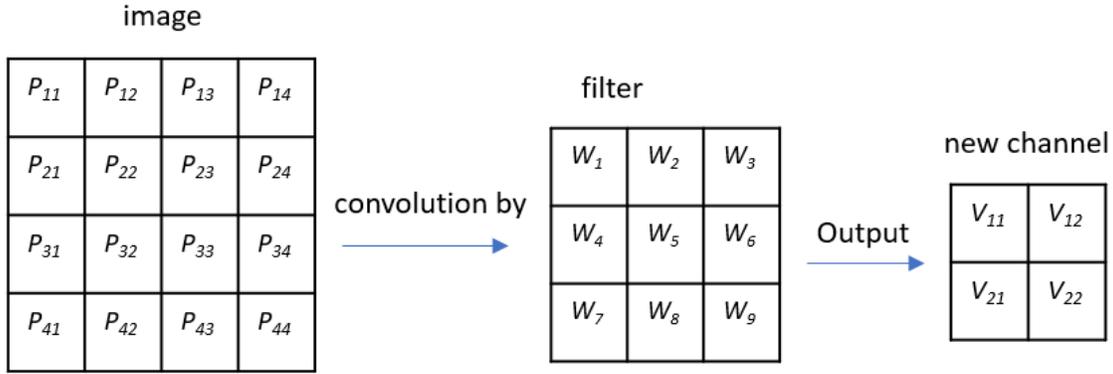
Fig. 1.10. Diagram showing convolution of a $4 \times 4 \times 1$ image with a

$3 \times 3 \times 1$ filter.

In Fig. 1.10 the output channel is calculated by:

$$v_{11} = \text{ReLU}(w_1 p_{11} + w_2 p_{12} + w_3 p_{13} + w_4 p_{21} + w_5 p_{22} + w_6 p_{23} + w_7 p_{31} + w_8 p_{32} + w_9 p_{33} + b)$$

$$v_{12} = \text{ReLU}(w_1 p_{12} + w_2 p_{13} + w_3 p_{14} + w_4 p_{22} + w_5 p_{23} + w_6 p_{24} + w_7 p_{32} + w_8 p_{33} + w_9 p_{34} + b)$$

$$v_{21} = \text{ReLU}(w_1 p_{21} + w_2 p_{22} + w_3 p_{23} + w_4 p_{31} + w_5 p_{32} + w_6 p_{33} + w_7 p_{41} + w_8 p_{42} + w_9 p_{43} + b)$$

$$v_{22} = \text{ReLU}(w_1 p_{22} + w_2 p_{23} + w_3 p_{24} + w_4 p_{32} + w_5 p_{33} + w_6 p_{34} + w_7 p_{42} + w_8 p_{43} + w_9 p_{44} + b)$$

Each additional filter will get a new channel in the output map, increasing the thickness. Fig. 1.11 shows a 3D view of a convolutional layer.
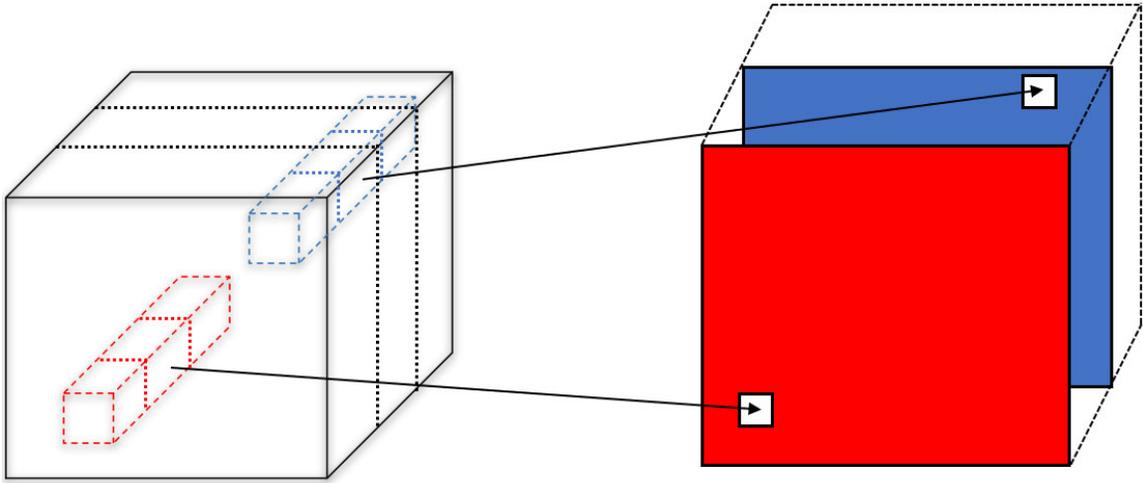
Fig. 1.11 A convolutional layer with two filters. The red filter and blue filter both scan the image with 3 channels on the left side, and each of them will form a channel. The two channels stack together to form the new feature map.

Besides the filter width F and number of filters N, there are two other hyper parameters in a convolutional layer: the stride size and padding size. The stride is the step size when a filter scans the image. It defines how many pixels shift in the input image as the filter moves, to get a pair of neighboring cells in the output channel. In Fig. 1.10, the stride equals 1.

In the output channel, the height and width are always few pixels smaller than the input image. The number could add up as more convolutional layers succeed each other. Padding is a solution by adding extra pixels (usually with values equal to 0) around the boundary of the input image. The padding size is always an even number, since left and right, top and bottom occur in pairs (Fig. 1.12).

0	0	0	0	0	0
0	P_{11}	P_{12}	P_{13}	P_{14}	0
0	P_{21}	P_{22}	P_{23}	P_{24}	0
0	P_{31}	P_{32}	P_{33}	P_{34}	0
0	P_{41}	P_{42}	P_{43}	P_{44}	0
0	0	0	0	0	0
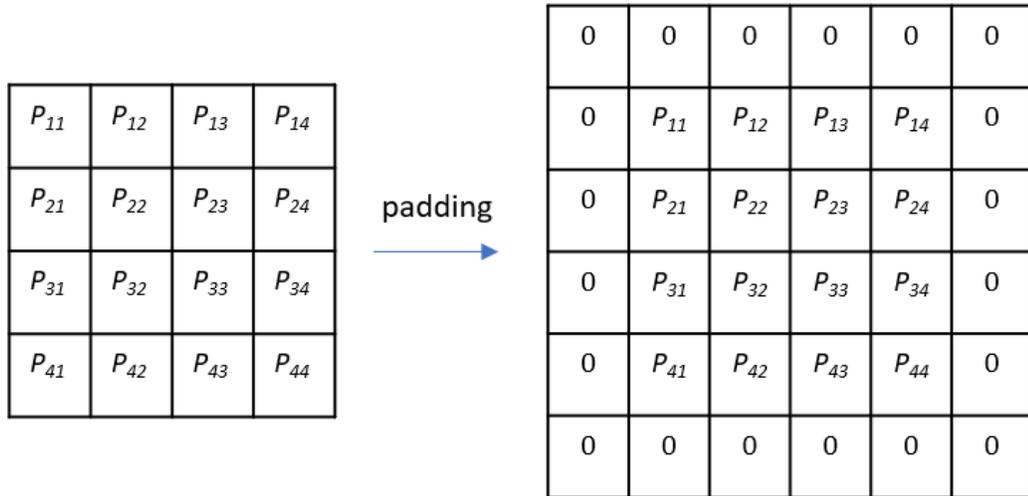
Fig. 1.12. A $4 \times 4 \times 1$ image padded with 0. The padding size equals two.

From all the information above we calculate that for an input image size $H_I \times W_I \times C$ convoluted by $N\ F \times F \times C$ filters, the output feature map size is:

$$(\frac{H_I - F + padding}{stride} + 1) \times (\frac{W_I - F + padding}{stride} + 1) \times N$$

The new feature map could be treated as an image which could be put into another convolutional layer. Beside the convolutional layers, the convolution neural networks usually also contain pooling layers and fully connected layers, discussed next.

Pooling Layer

A pooling layer (or subsampling layer) is a layer usually put after a convolutional layer, that shrinks the height and width of an image or feature map by a factor of 2. The most common pooling layers are average pooling and max pooling. For every channel of the image or feature

map, the pooling layer separates them into non-overlapping 2 by 2 pixel grids. Average pooling

calculates the average value of the 4 values in the grid, while max pooling chooses the maximum

value in the grid. Fig. 1.13 shows average pooling and max pooling of a $4 \times 4 \times 1$ feature map.
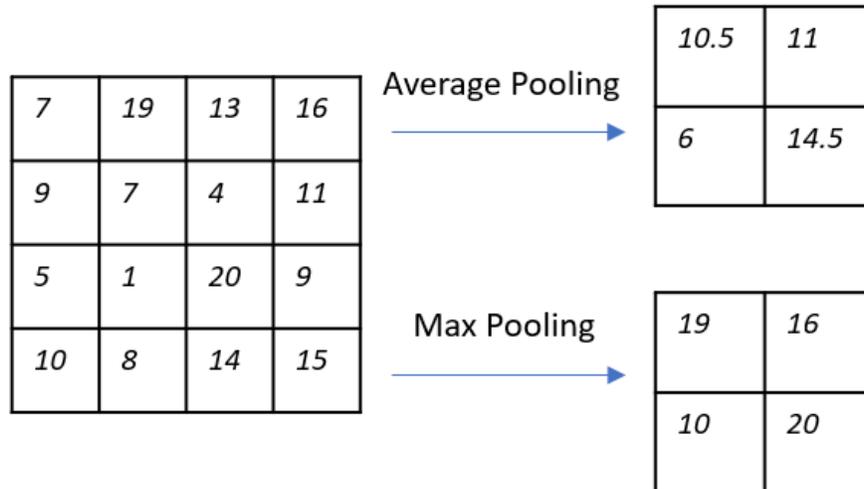
Fig. 1.13. The results of a $4 \times 4 \times 1$ image passing through an average

pooling layer and a max pooling layer.

The pooling layer reduces the number of parameters to learn by factor of 4. It essentially plays a

role in summarizing the feature maps. Besides saving the computational resources, the pooling

layer also helps to reduce the chance of over-fitting. In some situations, the pooling cell can be

larger than 2×2. The stride and padding techniques can also apply to pooling layers.

Fully Connected Layer

Fully connected layers are usually put at the last step to get the prediction of the whole

neural network. After passing through the convolutional layers and pooling layers, the input

image will become a feature map, which is a 3-dimensional matrix. If there are no convolutional or pooling layers anymore, the feature map will be expanded into a column vector. The column vector is treated as the input of the fully connected layer. Fig. 1.14 shows a simple example of how the fully connected layers work.
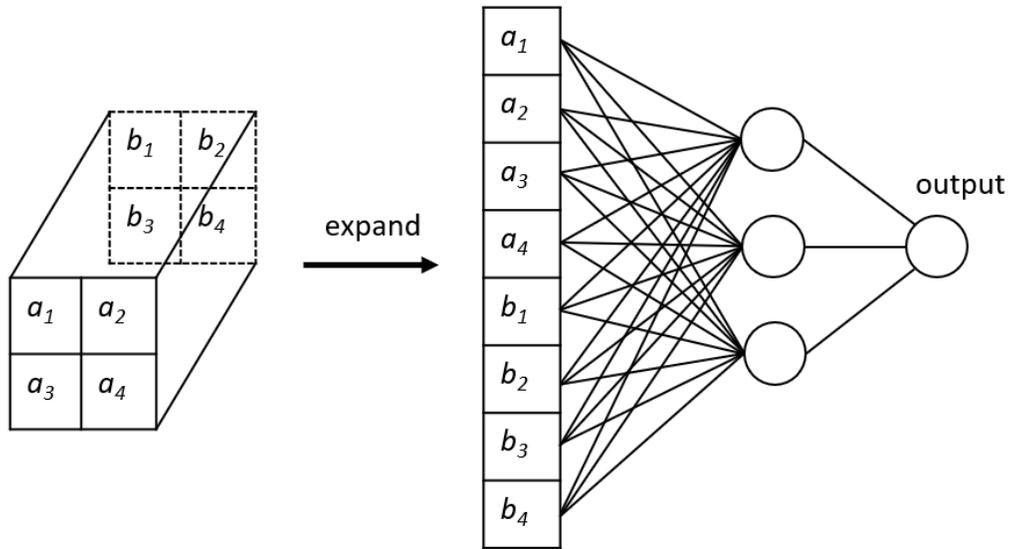
Fig. 1.14. A diagram shows a $2 \times 2 \times 2$ feature map expanded into an 8-element column vector, which then feeds into a fully connected layer and output the prediction. The fully connected layer has 3 elements.

Fully connected layers work exactly the same as traditional neural networks mentioned in the previous section. Thus, the loss function will be determined based on the task type, such as mean squared error for regression tasks, cross-entropy loss for binary classification tasks, etc.

Convolutional Neural Network Models

Starting in 2010, there is an annual international competition called the ImageNet Large Scale Visual Classification Challenge (ILSVRC)[13]. ILSVRC uses the subset of ImageNet[14], which contains 1.2 million training images, 50,000 validation images and 150,000 testing images. The images are separated into 1000 categories. The two main metrics used in the competition are the top-5 error and top-1 error. The top-5 error is the fraction of predictions that failed to include the true label of the image among the top 5 guesses. The top-1 error is the fraction of the predictions that do not give the true label as the highest guess. The ILSVRC has become the standard benchmark of convolutional neural networks in image classification. The main milestones in the development of convolutional neural networks are presented here.

AlexNet[15] was the winner of the LSVRC-2012. It achieved 17.0% top-5 error and 37.5% top-1 error. By contrast, the best top-5 grade of networks without convolutional neural networks was 26.1%. AlexNet used 2 GPUs to train the images with two groups of convolutional layers going parallel. The 1 GPU version of AlexNet is called CaffeNet. Fig. 1.15 shows the architecture of AlexNet and CaffeNet.
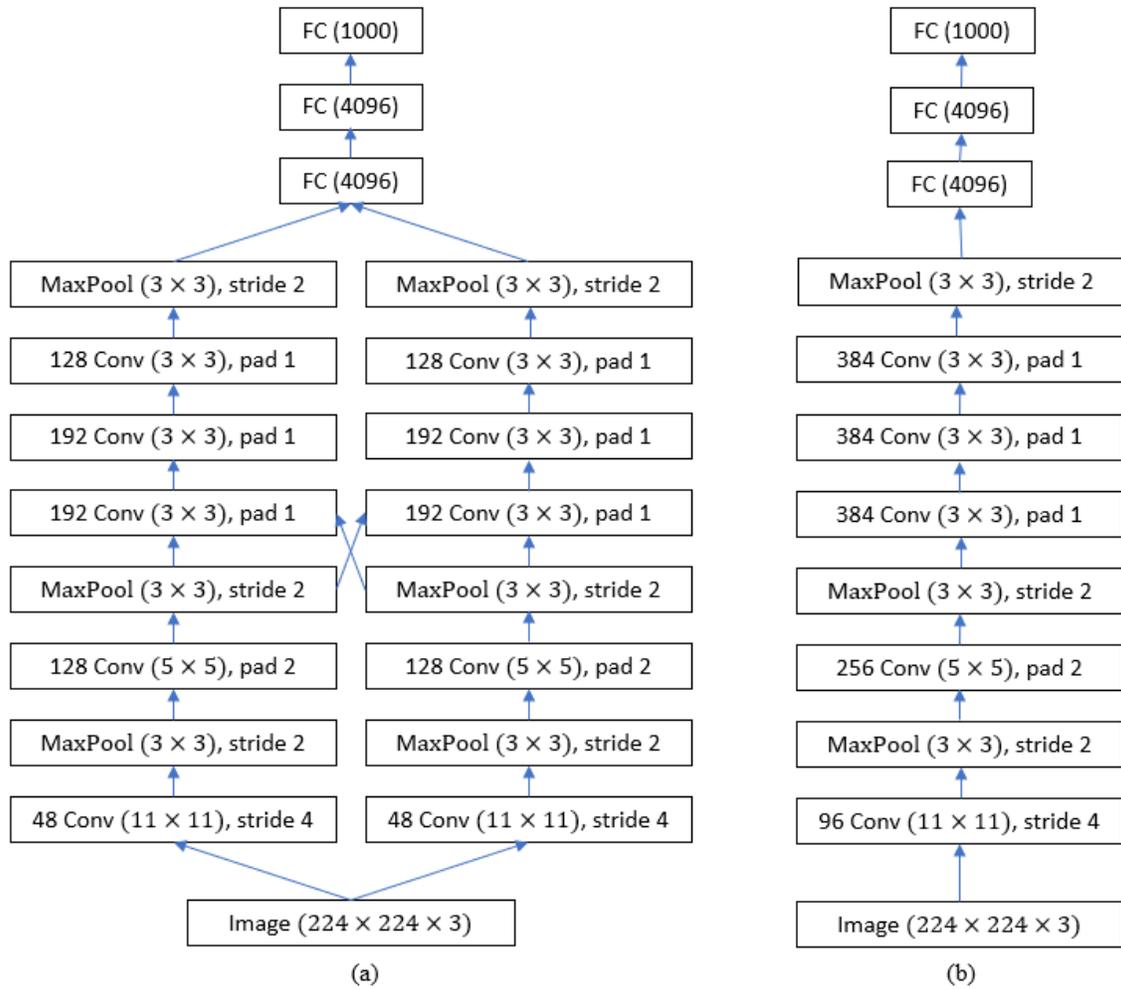
Fig. 1.15. The architecture of (a) AlexNet, (b) CaffeNet.

VGG net won the LSVRC-2014[16]. It achieved 6.8% top-5 error and 23.7% top-1 error. The VGG net only used the smallest filter (3×3), since the group believed that the larger filters are equivalent to stacking 3×3 filters which contains less parameters. Fig. 1.16 shows the architecture of the VGG net.

| Image (224 × 224 × 3) | 64 Conv (3 × 3) | 64 Conv (3 × 3) | MaxPool | 128 Conv (3 × 3) | 128 Conv (3 × 3) | MaxPool | 256 Conv (3 × 3) | 256 Conv (3 × 3) | 256 Conv (3 × 3) | 256 Conv (3 × 3) | MaxPool | 512 Conv (3 × 3) | 512 Conv (3 × 3) | 512 Conv (3 × 3) | 512 Conv (3 × 3) | MaxPool | 512 Conv (3 × 3) | 512 Conv (3 × 3) | 512 Conv (3 × 3) | 512 Conv (3 × 3) | MaxPool | FC (4096) | FC (4096) | FC (1000) |
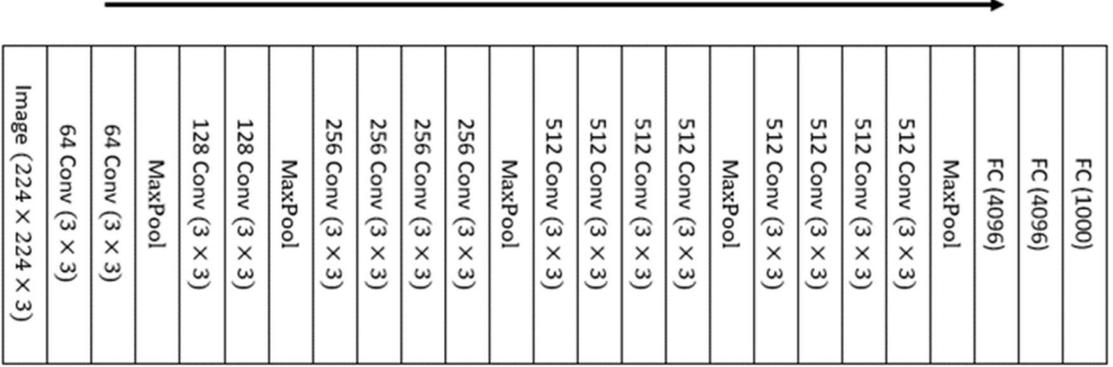
Fig.1.16. The architecture of the VGG net.

CHAPTER TWO: SEMICONDUCTORS AND PHOTOLUMINESCENCE

Band Structure and Semiconductors

Nuclei in a crystal form a periodic potential. To simplify the structure, Kronig and Penny introduced a model[17] which treated an electron in a linear array of positive nuclei as periodic square wells (Fig. 2.1).
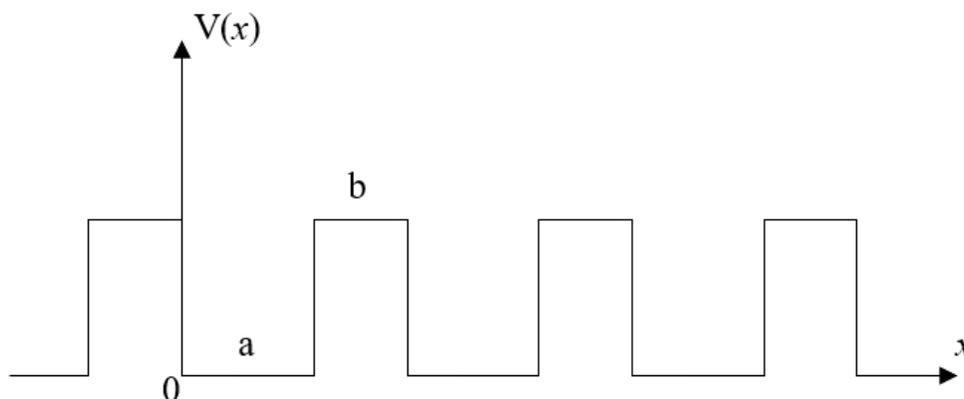
Fig. 2.1. The periodic square well potential function. $V = 0$ when

$n(a + b) < x < n(a + b) + a$, and $V = V_0$ when $n(a + b) - b < x < n(a + b)$,

where n is an integer.

Applying the Schrödinger equation,

$$\frac{d^2\psi}{dx^2} + \alpha^2\psi = 0 \qquad \text{when } n(a + b) < x < n(a + b) + a \qquad (2.1)$$

$$\frac{d^2\psi}{dx^2} + \beta^2\psi = 0 \qquad \text{when } n(a + b) - b < x < n(a + b) \qquad (2.2)$$

where $\alpha^2 = \frac{2mE}{\hbar^2}$ and $\beta^2 = \frac{2m}{\hbar^2}(V_0 - E)$. If $V_0 > E$, α and β are real.

From these equations, Kronig and Penny obtained the relationship:

$$\beta^2 - \frac{\alpha^2}{2\alpha\beta} - \sinh\beta b \sin\alpha a + \cosh\beta b \cos\alpha a = \cos k(a+b) \qquad (2.3)$$

To simplify the relation, they assumed the potential to be delta functions, $V = \delta(x - na)$. Thus V_0 goes to infinity and b goes to 0, but the product $V_0 b$ remains a finite value. Then Eq. (2.3) becomes:

$$P\frac{\sin\alpha a}{\alpha a} + \cos\alpha a = \cos ka \qquad (2.4)$$

where $P = \frac{\beta^2 ab}{2} = \frac{mV_0 ab}{\hbar^2}$. Since $\alpha = \sqrt{\frac{2mE}{\hbar^2}}$, Eq. (2.4) represents the relationship between energy and k-vector (crystal momentum). The E vs. k function is a discontinuous function (Fig. 2.2).
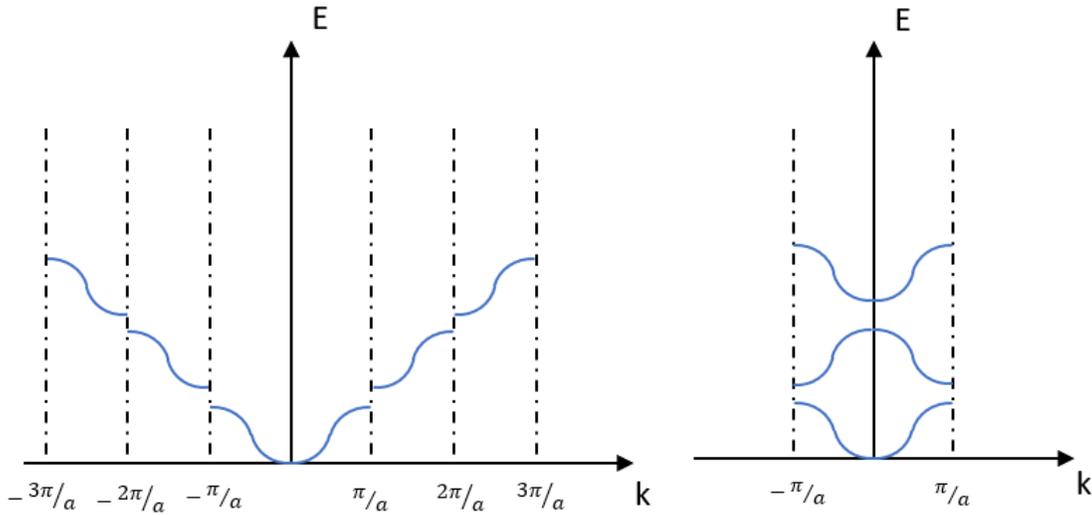
Fig. 2.2. The left side is the plot of energy in momentum space, and the right side is the plot in the reduced vector zone.

The intervals between the energy states are called forbidden bands. From the Pauli exclusion principle, each energy state can only hold a limited number of electrons. At zero temperature, electrons occupy the lower energy bands. The highest energy band which has been fully occupied is called the valence band, and the next higher band is the conduction band. The energy difference between the maximum point of the valence band (VBM) and the minimum point of the conduction band (CBM) is called the band gap. If the VBM and CBM occur at the same k, the band gap is called a direct gap; otherwise, an indirect gap. The transition of an electron from VBM to CBM in a direct gap semiconductor needs a photon that has an energy greater than or equal to the band gap. However, for an indirect gap, to conserve the momentum, the transition requires a phonon to participate. Usually, semiconductors have band gaps between 0 to 4 eV, but it is not a strict rule[18].
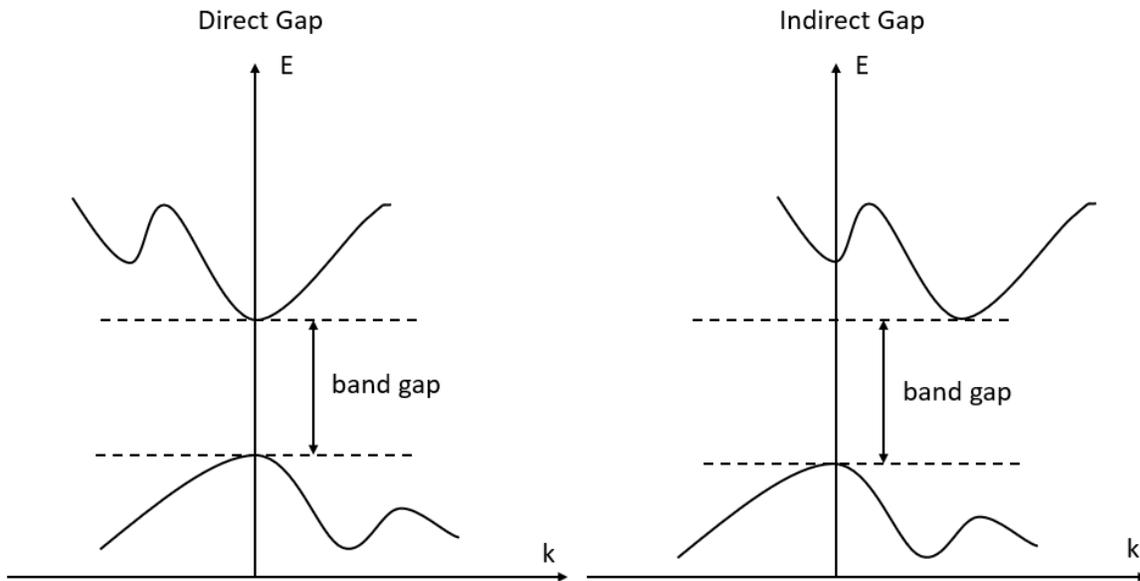
Fig. 2.3. Plots of E versus k function. The left side is a direct gap, and the right side is an indirect gap.

Vibrational Properties and Phonons

In solid, the atoms will oscillate about their equilibrium positions. These vibrational excitations are called lattice waves and quantized into phonons. A simple model is a linear chain with two different atoms. Let's assume there are two different atoms with mass M_1 and M_2 ($M_1 > M_2$) in a chain one by one. The forces between them are spring forces with spring constant C. The displacements of the sth pair of atoms from their equilibria are u_s and v_s (Fig. 2.4).
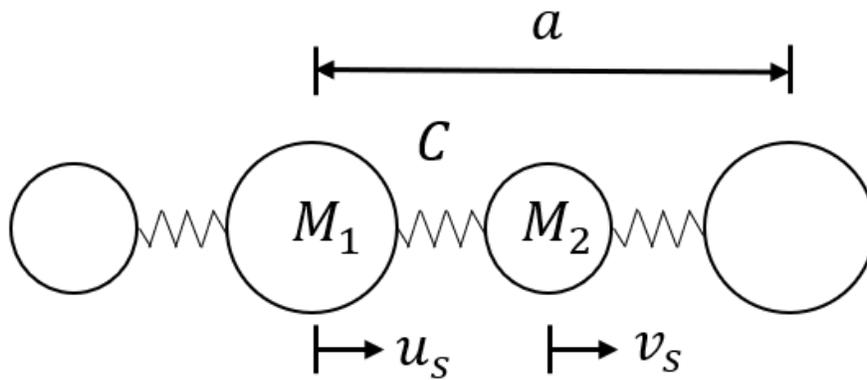
Fig. 2.4. Illustration of a two-atom chain model.

The waves are expressed in terms of complex exponentials:

$$u_s = u e^{i(Ksa - \omega t)} \qquad (2.5)$$

$$v_s = v e^{i(Ksa - \omega t)} \qquad (2.6)$$

where u and v are amplitudes, and K is the phonon momentum. With $K > 0$, the wave travels to the right, and with $K < 0$, the wave travels to the left. From the displacement and spring constant, we get the force applied on the two atoms:

$$F_{M_1} = C(v_s - u_s) + C(v_{s-1} - u_s) = C(v_s + v_{s-1} - 2u_s) \tag{2.7}$$

$$F_{M_2} = C(u_{s+1} - v_s) + C(u_{s-1} - v_s) = C(u_{s+1} + u_s - 2v_s) \tag{2.8}$$

Here $u_{s+1} = ue^{i(K(s+1)a-\omega t)} = e^{iKa}ue^{i(Ksa-\omega t)}$, and $v_{s-1} = ve^{i(K(s-1)a-\omega t)} =$

$e^{-iKa}ve^{i(Ksa-\omega t)}$. From Newton's second law and Eq. (2.5) and Eq. (2.6), we get:

$$F_{M_1} = M_1 \frac{d^2 u_s}{dt^2} = -M_1 \omega^2 u e^{i(Ksa-\omega t)} \tag{2.9}$$

$$F_{M_2} = M_2 \frac{d^2 v_s}{dt^2} = -M_2 \omega^2 v e^{i(Ksa-\omega t)} \tag{2.10}$$

Combining the equations above yields:

$$-M_1 \omega^2 u = C[v(1 + e^{-iKa}) - 2u] \tag{2.11}$$

$$-M_2 \omega^2 v = C[u(1 + e^{iKa}) - 2v] \tag{2.12}$$

Then the solution for ω is:

$$\omega^2 = C(\frac{1}{M_1} + \frac{1}{M_2}) \pm C\sqrt{(\frac{1}{M_1} + \frac{1}{M_2})^2 - \frac{2}{M_1 M_2}(1 - \cos Ka)} \tag{2.13}$$

Eq. (2.13) shows the vibration has two modes. In the low frequency mode, two atoms move with the same phase, and the branch is called the acoustical branch. In the high frequency mode, atoms move with the opposite phase, and the branch is called the optical branch. At the Brillouin zone center ($K = 0$), the acoustical branch frequency ω equals 0, and the optical branch frequency equals $\sqrt{2C(\frac{1}{M_1} + \frac{1}{M_2})}$. At the Brillouin zone edge ($K = \pm\frac{\pi}{a}$), the acoustical branch

frequency equals $\sqrt{\frac{2C}{M_1}}$, and the optical branch frequency equals $\sqrt{\frac{2C}{M_2}}$. The dispersion relation is shown in Fig. 2.5.
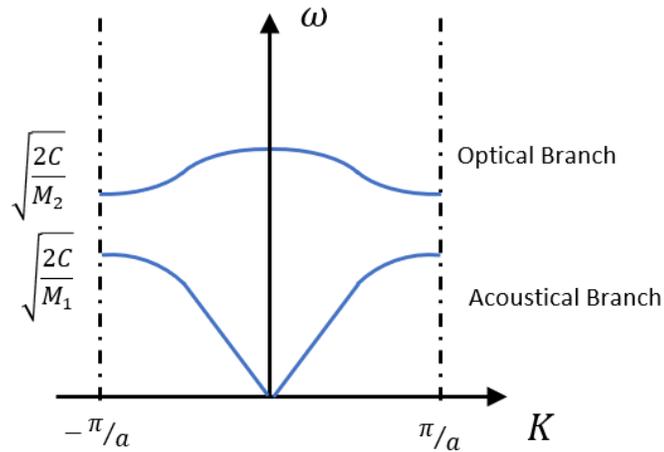
Fig. 2.5. Dispersion relation of the two-atom linear chain.

Between the branches, there is a region of frequencies that the wave cannot propagate, which is called forbidden gap. Besides longitudinal waves, the atoms can also move perpendicular to the wave propagation direction resulting a transverse wave. As each longitudinal mode has two transverse modes, there are four vibrational modes: one longitudinal acoustical (LA) branch, two transverse acoustical (TA) branches, one longitudinal optical (LO) branch, and two transverse optical (TO) branches[19].

Electrons and Holes

33

At zero temperature in an intrinsic semiconductor, the valence band is full, and the conduction band is empty. Therefore, the semiconductor is insulating. When the temperature increases, the electrons are excited from the valence band to the conduction band. The previously occupied positions in the valence band are left with electron "holes". When applying an electric field, a nearby electron could move into a hole, which can be seen as a hole with positive charge that is moving. An electron in the conduction band and a hole in the valence band have effective masses[20]:

$$m_e = \hbar^2 \left(\frac{\partial^2 E}{\partial k^2}\right)^{-1} \tag{2.14}$$

$$m_h = \hbar^2 \left(\frac{\partial^2 E_h}{\partial k^2}\right)^{-1} \tag{2.15}$$

where E_h is the hole energy and is equal to $-E$. Both m_e and m_h depend on the curvature of the band. Near the CBM and VBM, the band structure can be approximated as a parabolic function, expressed as:

$$E = \frac{\hbar^2 k^2}{2m_{e(h)}} \tag{2.16}$$

The product of the electron density n and hole density p in a semiconductor is:

$$np = N_c N_v e^{\frac{-E_g}{k_B T}} \tag{2.17}$$

where E_g is the band gap energy and k_B is the Boltzmann constant. N_c and N_v are the density of states in the conduction band and valence band, which are given by:

$$N_c = \frac{1}{\sqrt{2}} \left(\frac{m_e k_B T}{\pi \hbar^2}\right)^{3/2} \tag{2.18}$$

$$N_v = \frac{1}{\sqrt{2}} \left(\frac{m_h k_B T}{\pi \hbar^2} \right)^{3/2} \tag{2.19}$$

From Eq. (2.17), the free carrier density is related to the band gap.

The Fermi energy E_f represents the energy where the probability of an electron occupying a state is 50%. In an intrinsic semiconductor, the Fermi energy is near the middle of the band gap. The Fermi-Dirac distribution gives the probability that an energy level E is occupied[18]:

$$f(E) = \frac{1}{e^{(E-E_f)/k_B T} + 1} \tag{2.20}$$

Within a range of $k_B T$ around E_f, the electron probability distribution drops from nearly 1 to almost 0 (Fig. 2.6).
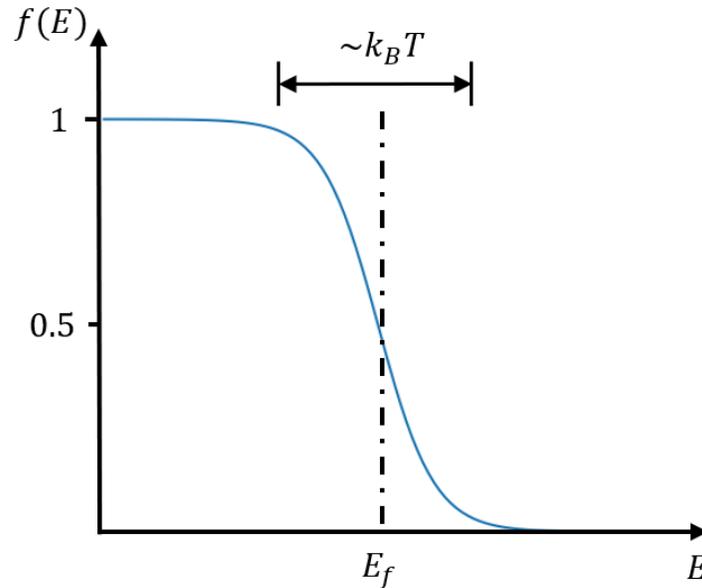
Fig. 2.6. Plot of Fermi-Dirac distribution.

At room temperature (300 K), $k_B T$ is about 0.026 eV. It is very unlikely that an electron will be thermally excited to the conduction band in a wide band gap intrinsic semiconductor.

An electron and a hole can attract each other by the coulomb force, forming an electron-hole pair. This electron-hole pair quasi-particle is called an exciton. Frenkel in 1931 first gave a model to describe the exciton[21]. In Frenkel excitons, the electron-hole pair is tightly bound together, and the particle size is small. It happens in materials with a small dielectric constant. In semiconductors, the dielectric constant is large, and the coulomb interaction between the electron-hole pair is weaker. Wannier in 1937 developed exciton theory and introduced the Wannier-Mott model[22]. In this model, the electron-hole pair is loosely coupled, which describes the excitons in semiconductors more appropriately. The exciton can absorb energy and dissociate into a free electron and a hole. Therefore, the binding energy of an exciton is smaller than the band gap. There are other many-particle states in semiconductors. Biexcitons are formed by van der Waals binding of two excitons. More excitons can also bind into other excitonic molecules.

Point Defects

Point defects change the properties of a semiconductor. Point defects are classified into different groups such as vacancy, interstitial and substitutional. Point defects can lead to extra electron energy states inside the band gap. If the defect can easily contribute free electrons, it is called a donor. At the opposite extreme, if the defect easily binds electrons, thus leaving free holes, it is called an acceptor. The extra energy state created by donors (acceptors) are called donor (acceptor) levels. If the donor (acceptor) level is close to conduction (valence) band edge, the donor (acceptor) is called shallow.

In silicon, the fifth group elements (P, As, Sb, Bi) have one more valence electron needed to form a Lewis octet. They become shallow donors when substituting a host atom. The third group elements (B, Al, Ga, In, Tl) need one more electron to form a Lewis octet[23]. They become shallow acceptors when substituting a host atom. The electron or hole orbit the defect like a hydrogen atom. The binding energy is weak, such that thermal energy causes the bound electron or hole to become free electron or hole. Fig. 2.7 shows examples of a shallow donor and acceptor.
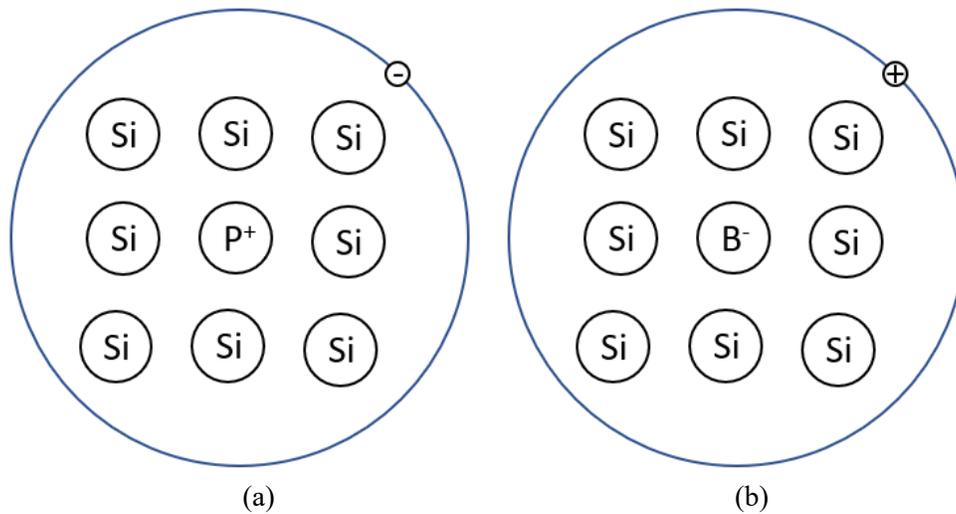
Fig. 2.7 (a) Phosphorus substitutes a silicon atom donating an electron, and the electron is weakly bound to P^+, forming a shallow donor. (b) Boron substitutes a silicon atom and grabs an electron, and the hole is weakly bound to B^-, forming a shallow acceptor.

Fig. 2.7 gives examples of a single donor and acceptor. They both have a single level in the band gap. Some defects have more levels. For example, selenium in germanium is a double donor, and copper in germanium is a triple acceptor.

If the energy states are far from the band edges, they are deep levels. The electrons or holes are strongly bound at the impurity cores, and their wave functions are localized. For example, nitrogen substitutes oxygen in zinc oxide. The acceptor level is 1.5 eV higher than the valence band. An oxygen vacancy in zinc oxide is a deep double donor since it can bind two electrons tightly in the impurity core. A zinc vacancy is a deep double acceptor[18].

The type of major charge carriers puts semiconductors into two classes. If the main charge carriers are electrons, it is called *n*-type. If holes, it is called *p*-type. In semiconductors with mixed donors and acceptors, the minority could compensate the majority. Majority donors make a semiconductor *n*-type, while majority acceptors make it *p*-type.

Transitions and Photoluminescence

When semiconductors absorb energy, electrons may transition higher energy levels, generating electron-hole pairs. Inversely, electron-hole pairs could recombine and emit energy, which is called recombination. If the emitted energy is in form of photons, the process is called radiative recombination. Furthermore, if the energy that generated the electron-hole pairs is in form of photons, the radiative recombination is called photoluminescence (PL). PL is very interesting since it is easily observable and may contain energy level and defect information[24]. Many transitions can cause PL, five of which are shown in Fig. 2.8.
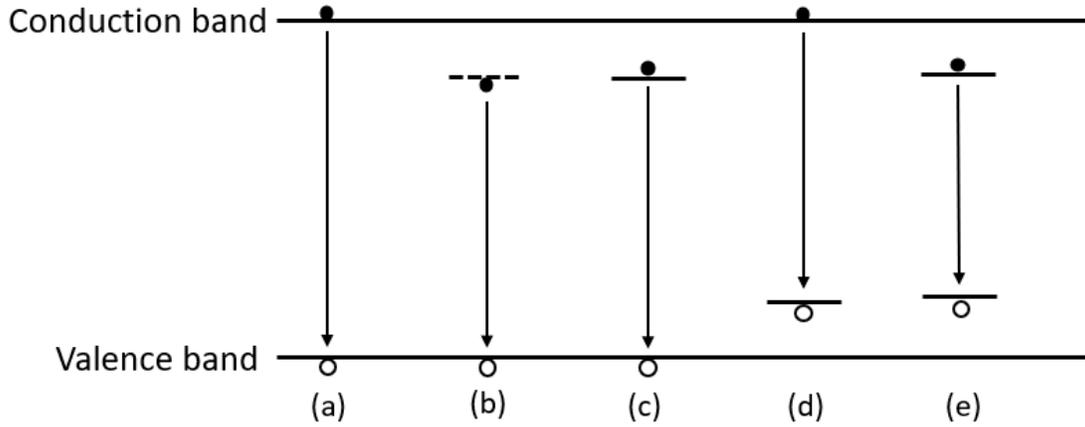
Fig. 2.8. Illustration of five observed transitions in PL. (a) Band-to-band, (b) Free exciton, (c) Donor-to-valence, (d) Free-to-acceptor, (e) Donor-acceptor pair.

The transition in Fig. 2.8(a) is the band-to-band transition. It refers to free electrons in conduction band dropping into the holes in valence band and emitting photons. When the light source has energy greater than or equal to the band gap, electrons in valence band are excited into the conduction band. The density of electrons and holes will be greater than the equilibrium. The recombination rate is proportional to the ratio of the product of electron and hole density to the equilibrium[25] [Eq. (2.21)].

$$R \propto \frac{np}{n_0 p_0} = \frac{(n_0 + \Delta n)(p_0 + \Delta p)}{n_0 p_0} = 1 + \frac{\Delta n}{n_0} + \frac{\Delta p}{p_0} + \frac{\Delta n \Delta p}{n_0 p_0} \tag{2.21}$$

As the non-equilibrium carrier densities are small, the term of $\Delta n \Delta p$ can be neglected. The recombination rate coming from the non-equilibrium carrier densities is:

$$\Delta R \propto \frac{\Delta n}{n_0} + \frac{\Delta p}{p_0} \tag{2.22}$$

From Eq. (2.22), the recombination is dominated by the minority charge carrier type. In an *n*-type semiconductor where $n_0 \gg p_0$, the recombination is controlled by non-equilibrium holes. At the opposite side, a *p*-type semiconductor's band-to-band recombination is controlled by the non-equilibrium electrons. In the indirect gap semiconductors band-to-band transitions must simultaneously emit phonons in order to conserve momentum.

The transition in Fig. 2.8(b) is a free exciton recombination. Free excitons have the ability to move though the crystal. The energy of a free exciton includes the coulomb interaction energy and its kinetic energy. When a radiative recombination of an exciton happens, the exciton annihilates, and the emitted photon has the same momentum as the exciton[25].

Fig. 2.8 (c) and (d) are both referred as free-to-bound transitions. Fig. 2.8(c) illustrates recombination of an electron localized at a donor with a hole in valence band, and Fig. 2.8(d) shows the recombination of a free electron and a hole localized at an acceptor.

The transition in Fig. 2.8(e) is a donor-acceptor pair (DAP) transition. DAPs behave as stationary molecules in the semiconductor. The Coulomb interaction lowers their binding energies. The recombination energy of a DAP is given by[25]:

$$\hbar\omega = E_g - (E_A + E_D) + \frac{e^2}{\epsilon R} \tag{2.23}$$

where E_A and E_D are the acceptor and donor binding energies, and R is the distance between the donor and acceptor. If the distance is very large, the last term of Eq. (2.23) can be neglected. If the donor and acceptor are very close to each other, the $\frac{e^2}{\epsilon R}$ gets larger and the recombination energy may exceed the band gap energy.

PL Spectroscopy

In the previous section, we mentioned that radiative recombination in a semiconductor could yield photoluminescence. Because there are many different radiative recombination processes happening simultaneously, the emitted photons have different energies. PL spectroscopy is a technique that excites the sample with a fixed wavelength light source and collects a spectrum of emitted photons. The resulting PL spectrum is usually represented in two forms. The first one plots the luminescence intensity versus the photon's wavelength, while the second one plots the intensity versus the photon energy.

The relationship between the photon energy E and the photon's wavelength λ is:

$$E = \frac{hc}{\lambda} \tag{2.24}$$

where h is Planck's constant and c is the light speed. The photon energy in eV and wavelength in nm can be converted by:

$$E(\text{eV}) = \frac{1239.8}{\lambda(\text{nm})} \tag{2.25}$$

From Eq. (2.25), the two types of PL spectra can be converted to each other, but it will result some resolution unbalance. For example, suppose a PL spectrum is collected as luminescence versus wavelength with 1 nm intervals, and is converted to luminescence versus photon energy. If energy is linear on the x-axis, the higher energy part tends to have fewer data points.

An example of a PL spectrum of gallium nitride (GaN) sample is shown in Fig. 2.9.
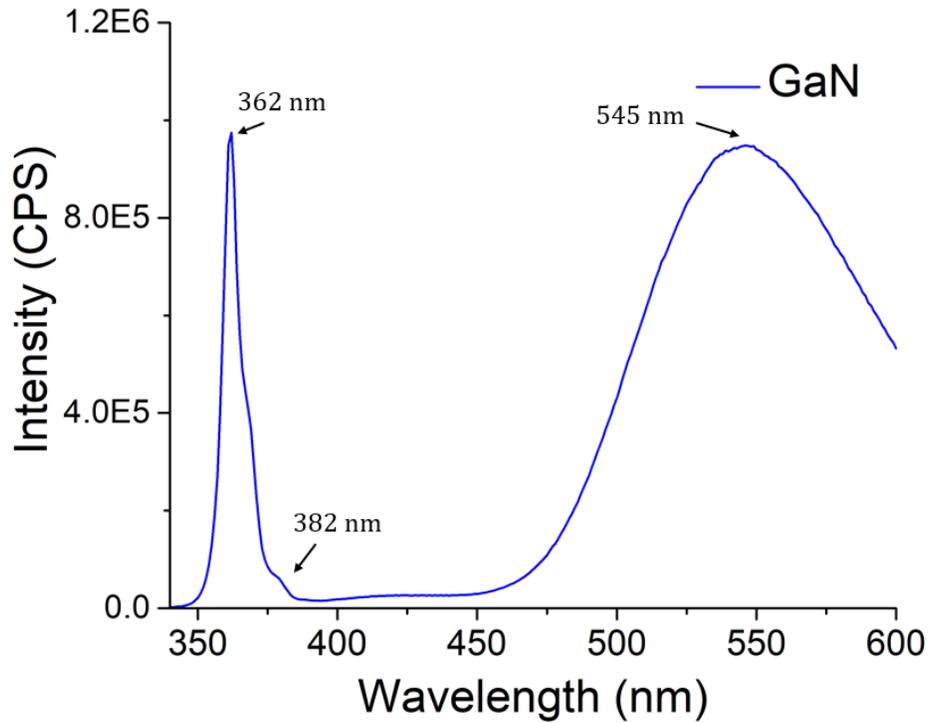
Fig. 2.9. A PL spectrum of gallium nitride plotted as intensity as a function of the emitted photon's wavelength. The excitation light source has a wavelength of 320 nm.

In Fig. 2.9 the peak near 362 nm is due to the band-to-band transition, the small peak near 382 nm is due to free electrons recombining with holes in acceptors, and the peak near 545 nm is due to carbon related defects.

While PL spectroscopy measures the emission spectrum with a fixed light source, photoluminescence excitation (PLE) spectroscopy measures the intensity of emitted light at a fixed wavelength with variable exciting light. Therefore, PLE needs a light source that can sweep

across a broad wavelength range. Fig. 2.10 shows a PLE spectrum of a gallium oxide (Ga_2O_3) sample.
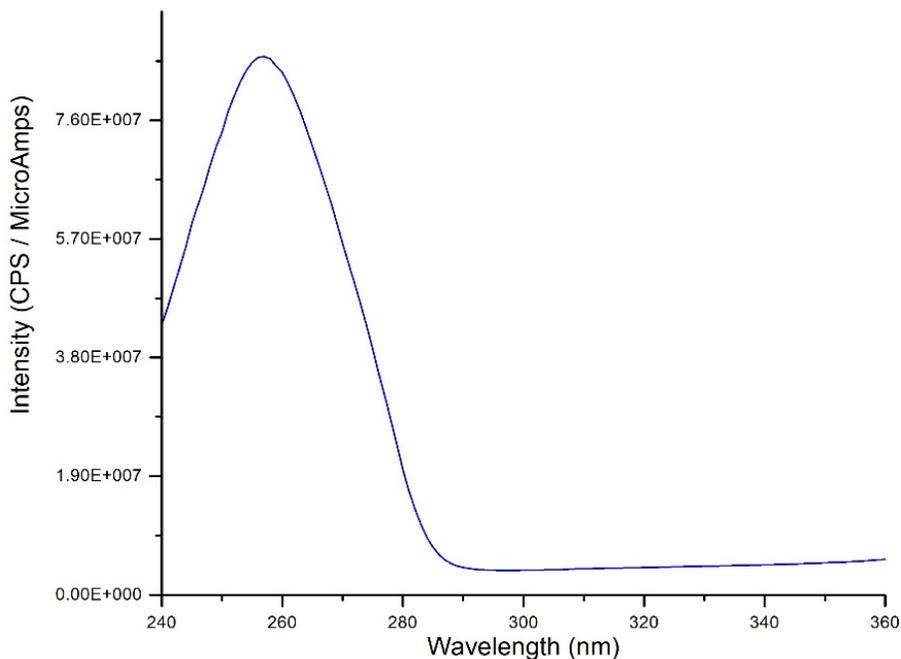
Fig. 2.10. PLE spectrum of Ga_2O_3 sample. The measured PL light is fixed at 378 nm.

Fig. 2.10 shows that PL is maximized for an excitation wavelength of 260 nm, which is about the band gap of Ga_2O_3.

PL Instrument

The spectra in Figs. 2.9 and 2.10 were collected on a Horiba Fluorolog spectrometer. The Fluorolog uses a 450 W xenon lamp and monochromator as the excitation source, and a R928P

PMT as the detector. A scanning monochromator and PMT collect the intensity of emitted light at different wavelengths one at a time. A schematic is shown in Fig. 2.11.
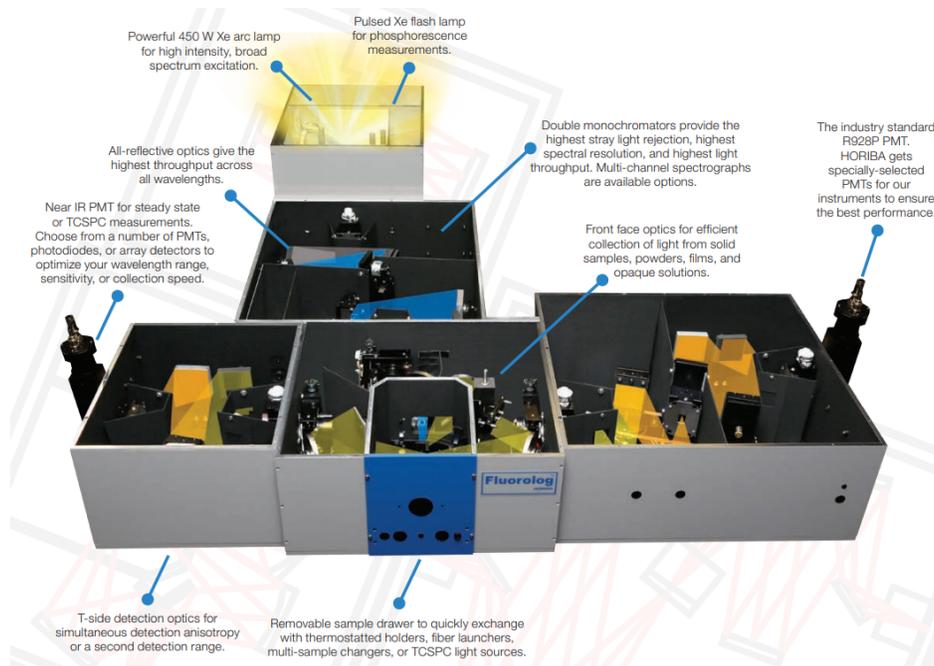
Fig. 2.11. Schematic of Horiba Fluorolog spectrometer. The image is from the Fluorolog Brochure[26].

Another kind of PL system uses a laser as excitation source and array device such as charge-coupled device (CCD) as detector. With a diffraction grating, the emitted light spreads into beams with different wavelengths that excite the CCD's different pixels. The advantage of this kind of device is speed. Instead of scanning the light energy range, it can acquire the whole PL spectrum instantly. However, it has less sensitivity and more noise than the PMT detector.

CHAPTER THREE: OPTICS AND CONFOCAL LASER MICROSCOPY

Lenses and Linear Optics

Light propagates following a straight line in a homogeneous medium. When light travels from one medium to a different medium, its trajectory will be bent. This phenomenon is called refraction. Refraction follows Snell's law[27]:

$$\frac{\sin\theta_1}{\sin\theta_2} = \frac{n_2}{n_1} \tag{3.1}$$

where θ_1 is the angle of incidence and θ_2 is the angle of refraction; n_1 and n_2 are the refractive index of two media, equal to the ratio of speed of light in vacuum over the speed of light in each medium. The process is shown in Fig. 3.1.
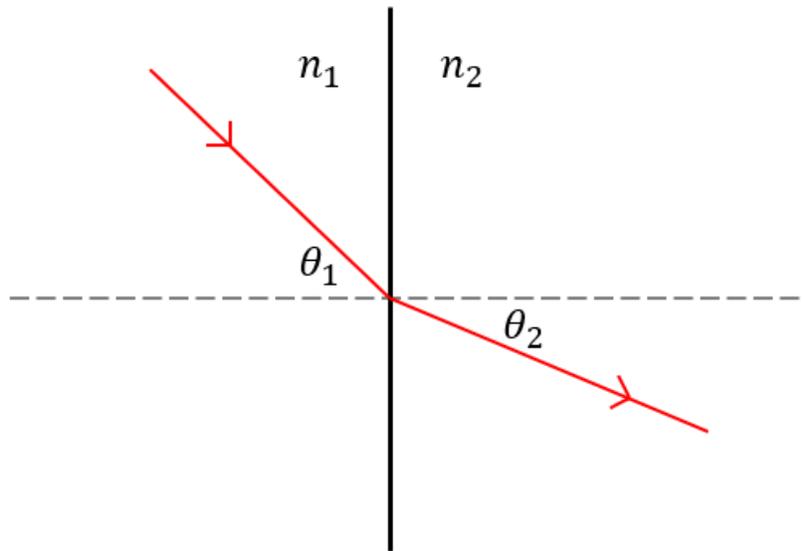
Fig. 3.1. Illustration of light refraction between two media with refractive indices n_1 and n_2.

A lens is a transmissive optical device that uses refraction to change the direction of light. The material's refractive index and the lens' shape will determine how the light rays converge or diverge after transiting the lens. A simple lens, which contains only a single piece of transparent material, can be classified into two types: convex and concave. A convex lens will focus an incident collimated light beam while a concave lens will disperse the incident collimated light beam. Fig. 3.2 shows a schematic of the convex lens and concave lens.
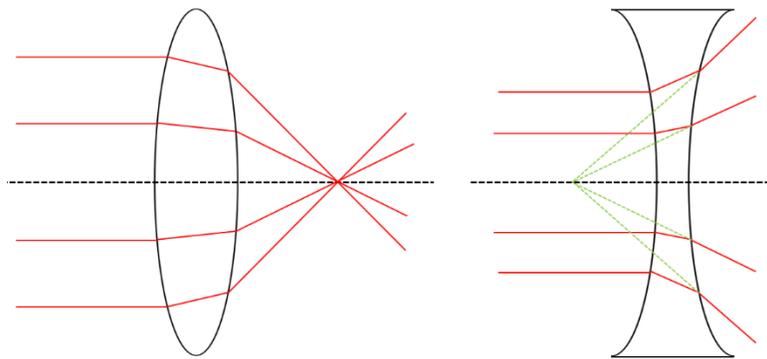
Fig. 3.2. Schematic of convex and concave lenses. The convex lens focuses the collimated light beam into a point. The concave lens will disperse the light beam; extrapolating the refracted light rays will converge to a point (green dashed lines).

To simplify the ray diagram, a lens can be treated as a plane with zero thickness. We use notations shown in Fig. 3.3 to represent convex and concave lenses.
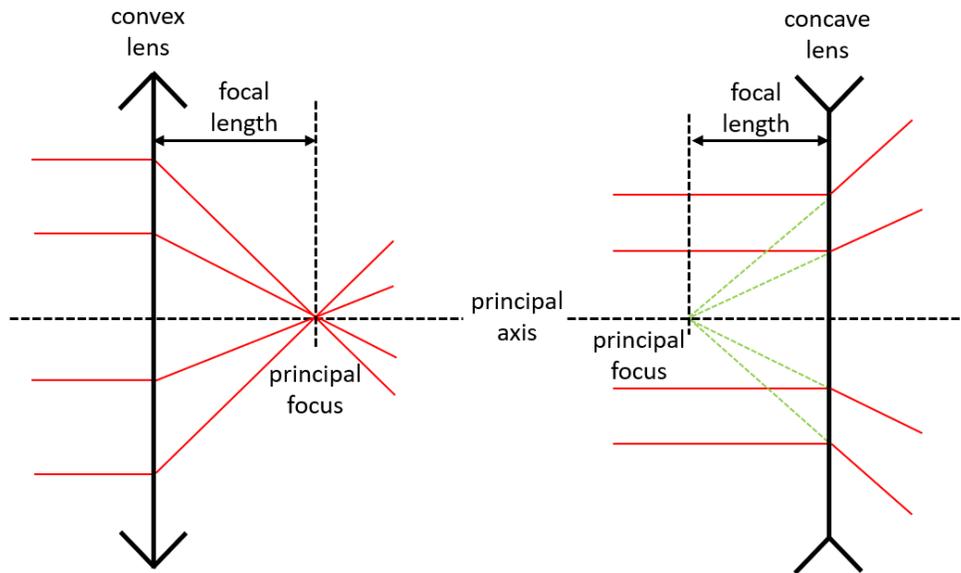
Fig. 3.3. Ray diagrams of a collimated beam passing through a convex and concave lens.

The line perpendicular to the lens and across the center of the lens is called the principal axis. When a collimated light beam is incident along the principal axis, the refracted beam or its opposite extension will focus on a point on the principal axis, and this point is called principal focus. The plane perpendicular to the principal axis and that contains the principal focus is called the focal plane, and the distance between the focal plane to the lens is called focal length. All the collimated incident light rays will have a focal point on the focal plane. An example of three collimated light beams through a convex lens is shown in Fig. 3.4.
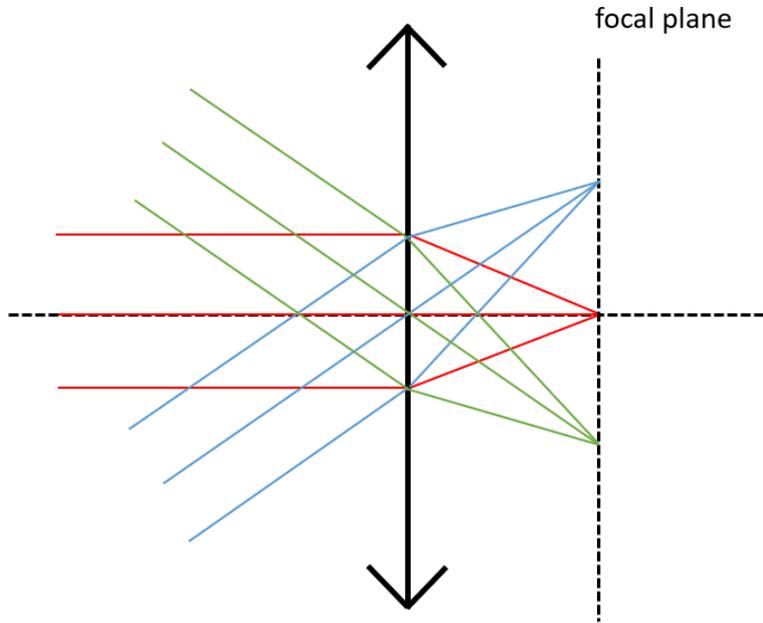
Fig. 3.4. Three collimated light beams incident from different directions

passing through a convex lens. All of them focus on the focal plane.

When light emitted by object passes through a lens, an image can be formed. If the image can be projected onto a screen, it is a real image; otherwise, it is a virtual image. The displacement from the lens to the object is called the object distance s, and the displacement from the lens to the image is called image distance s'. The relationship between s and s' is:

$$\frac{1}{s} + \frac{1}{s'} = \frac{1}{f} \qquad (3.2)$$

where f is the focal length and is positive for a convex lens and negative for a concave lens. Fig. 3.5 shows how the images form from an object through the lenses.
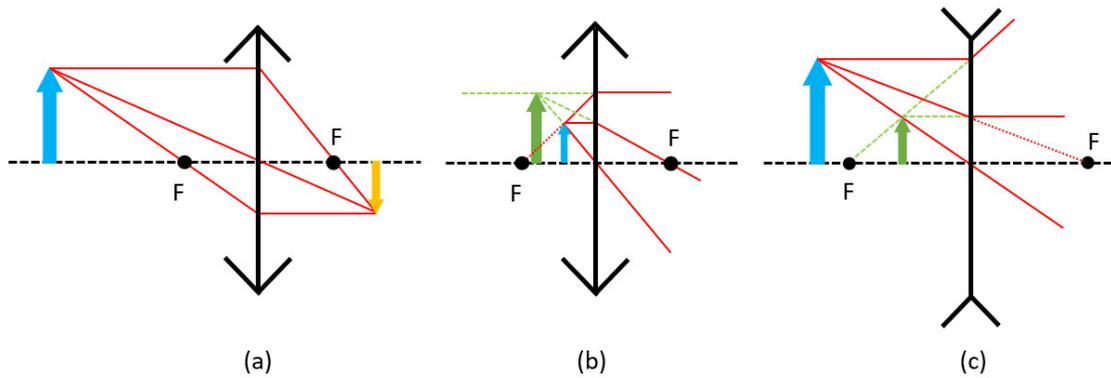
Fig. 3.5. (a) Real image formed by a convex lens. (b) Virtual image formed by a convex lens. (c) Virtual image formed by a concave lens.

The image formed by one lens can be treated as a new object and forms a new image by another lens, which is in sequence and shares the common principal axis with the first lens. Therefore, a couple of simple lenses in sequence along the principal axis can form a compound lens. A common usage of compound lens is the beam expander[28]. The simplest types of beam expander are Keplerian and Galilean, both of which contain two simple lenses. The Keplerian beam expander contains two convex lenses with different focal lengths, and these lenses are placed with a distance equal to sum of the focal lengths. The Galilean beam expander contains a concave lens with a smaller focal length and a convex lens with a larger focal length. A schematic of the two beam expanders is shown in Fig. 3.6.
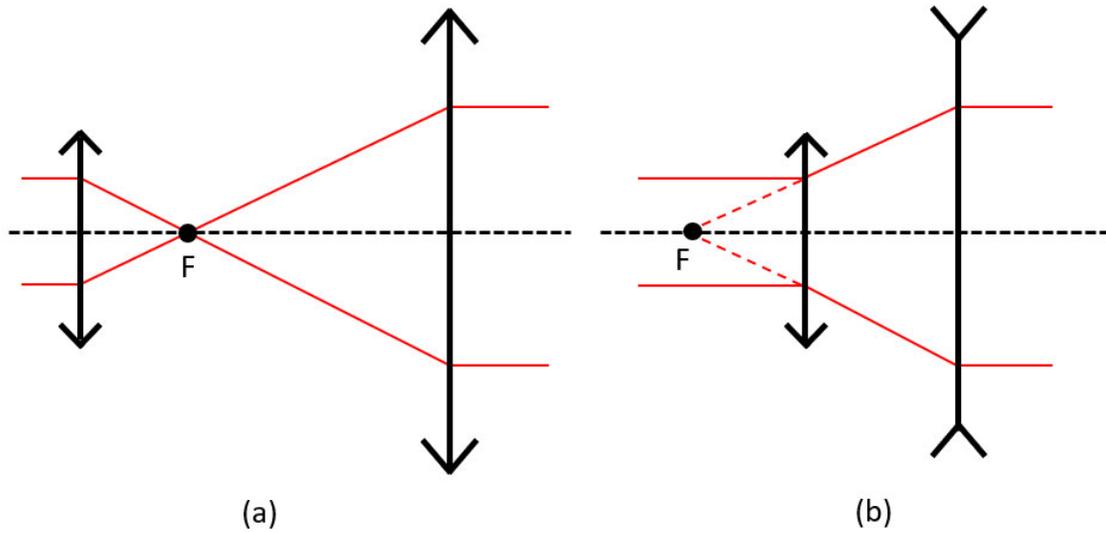
Fig. 3.6. Schematic of (a) Keplerian and (b) Galilean beam expander.

Gaussian Beam

A gaussian beam is a beam whose intensity in the transverse plane follows a gaussian distribution[29]. Usually, a laser beam is assumed to be gaussian. The beam intensity as a function of the radius from the beam center is:

$$I(r) = I_0 e^{-\frac{2r^2}{w(z)^2}} \tag{3.3}$$

where I_0 is the intensity at the beam center. From Eq. (3.3), $w(z)$ is the radius where the beam intensity is $1/e^2$ of I_0. Therefore $w(z)$ is a quantity that represents the size of the beam. The z-axis is along the direction of the beam propagation. An illustration of a gaussian beam in transverse plane is shown in Fig. 3.7.
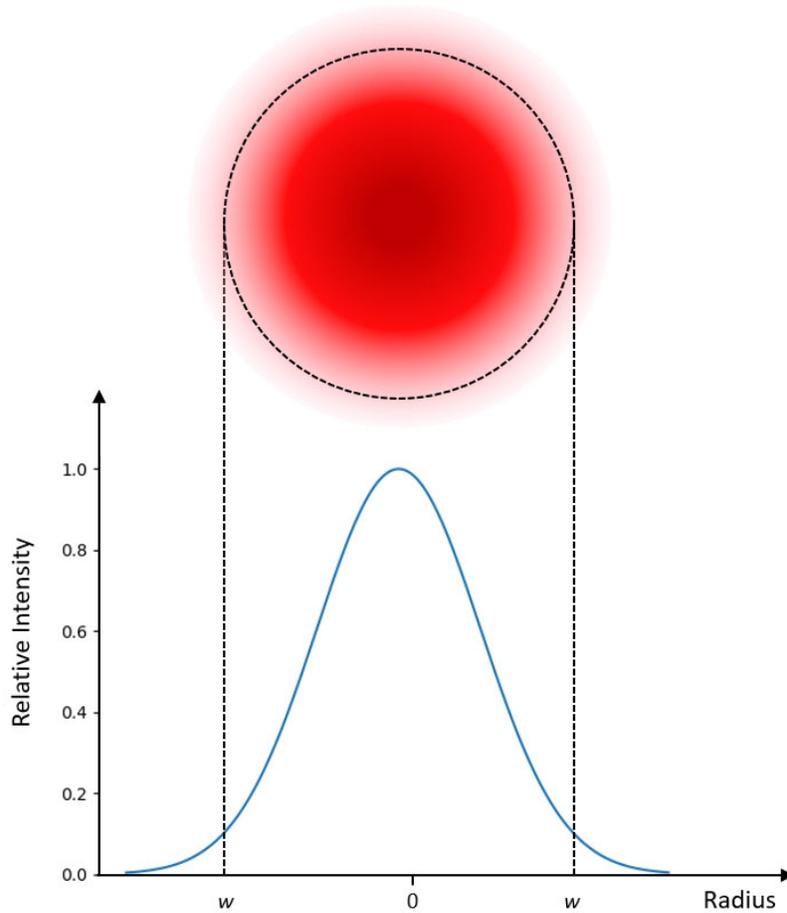
Fig. 3.7. Illustration of a gaussian beam in the transverse plane. The intensity follows a gaussian distribution, and at w, the intensity is $1/e^2$ of the intensity at the beam center.

The gaussian beam diameter reaches a minimum value at $z = 0$, where $w(z)$ is denoted as w_0, the beam waist. In the previous section, we assumed light can focus on a point. However, in gaussian beam, the beam is not linear. Near the beam waist, the rules in linear optics do not apply, but the linear approximation does apply when z is large. If the far field approximation of the beam divergence angle is θ, the size of beam waist is given by[30]:

$$w_0 = \frac{\lambda}{\pi n \theta} \tag{3.4}$$

where λ is the wavelength and n is the index of refraction. An illustration of a gaussian beam near the beam waist is given in Fig. 3.8.
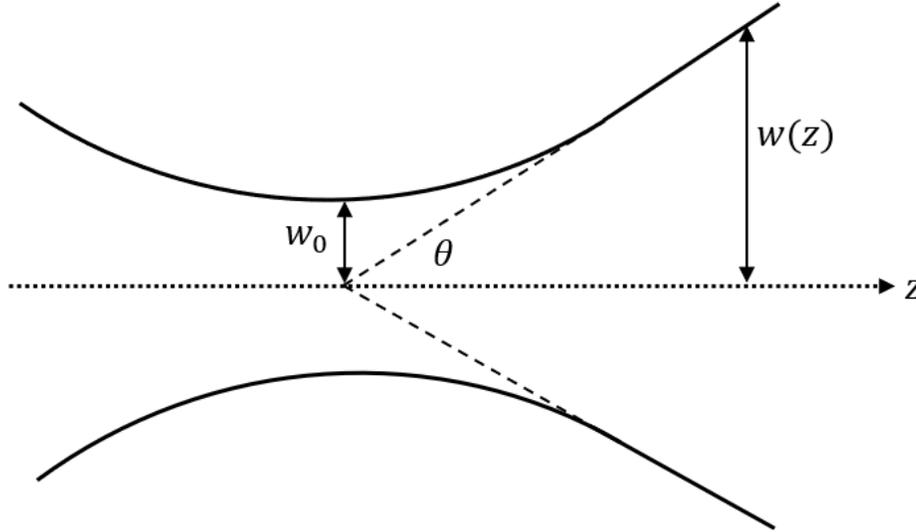
Fig. 3.8. Illustration of Gaussian beam with beam waist w_0 and divergence angle θ.

Both sides of the beam waist have the same rate of divergence or convergence. The beam size increases as a function of beam propagation axis z[31]:

$$w(z) = w_0 \sqrt{1 + (\frac{\lambda z}{\pi n w_0^2})^2} \tag{3.5}$$

The radius of the curvature of the wavefront is given by:

$$R(z) = z[1 + (\frac{\pi n w_0^2}{\lambda z})^2] \tag{3.6}$$

When z is large, the beam is almost linear, $R(z) \approx z$. At the area near the beam waist, the wavefront is nearly planar and $R(z)$ goes to infinity at the beam waist. z_R is defined as the point at which $R(z)$ is at its minimum. From Eq. (3.6), we get:

$$z_R = \frac{\pi n w_0{}^2}{\lambda} \tag{3.7}$$

Substituting into Eq. (3.5), we obtain $w(z_R) = \sqrt{2}w_0$. The range of $z = \pm z_R$ is called Rayleigh range, also known as depth of focus of the beam. Using this value, $w(z)$ can be represented as:

$$w(z) = w_0 \sqrt{1 + (\frac{z}{z_R})^2} \tag{3.8}$$

A gaussian beam passing through a perfect thin lens is still gaussian. With a gaussian beam refocused by a convex lens, the beam waist of the incident beam w_0 can be treated as the object, and the beam waist of the refocused beam w_0' can be treated as the image, shown in Fig. 3.9.
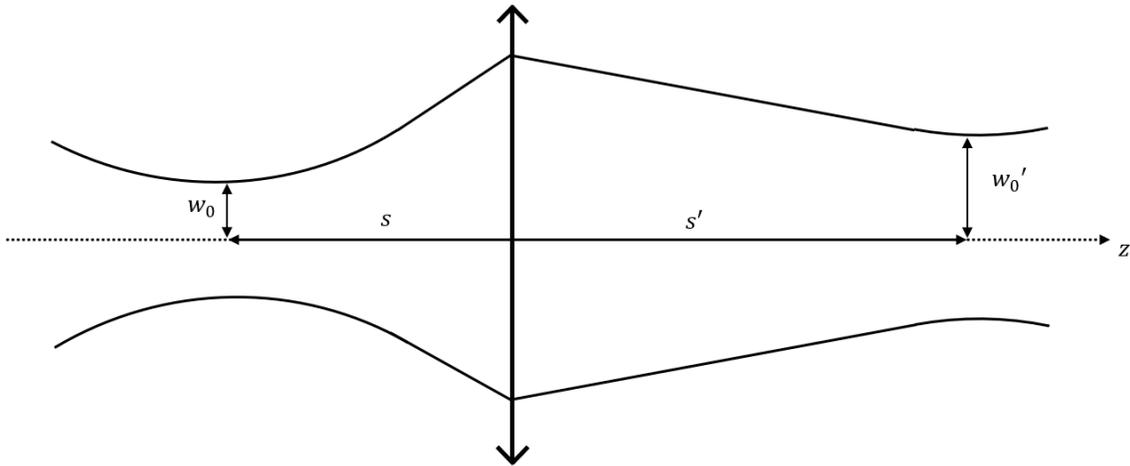
Fig. 3.9. Gaussian beam refocused by a thin convex lens.

From Fig. 3.9, the beam size at the lens should be same on both sides, and the magnification m is defined by $\frac{w_0'}{w_0}$. The magnification is given by[32]:

$$m = \frac{f}{\sqrt{(|s|-f)^2+z_R^2}} \tag{3.9}$$

The relationship between the magnification and the beam waist location is given by[32]:

$$m^2 = \frac{w_0'^2}{w_0^2} = \frac{s'-f}{|s|-f} \tag{3.10}$$

From Eq. (3.9) and Eq. (3.10) we have:

$$s' = f + m^2(|s| - f) = f + \frac{(|s|-f)f^2}{(|s|-f)^2+z_R^2} \tag{3.11}$$

If $|s| - f \ll z_R$ and $f \ll z_R$, from Eq. (3.11) we get $s' \approx f$. If $|s| - f \gg z_R$ and $|s| - f \gg f$, Eq. (3.11) still gives $s' \approx f$. This is because either at the position very far away from the beam waist, or very close to the beam waist, the curvature of the wave is near 0. The modified version of Eq. (3.2) for a gaussian beam is given by Self:

$$\frac{1}{s'} = \frac{1}{s+\frac{z_R^2}{s+f}} + \frac{1}{f} \tag{3.12}$$

A truly collimated beam with zero divergence is impossible to achieve. If the beam is collimated along the z axis, the photon momentum in the x and y directions is 0. According to Heisenberg Uncertainty Principle, the photon's position in the x and y directions is completely unknown. In this situation, only a beam with infinite width can be perfectly collimated. An approximately collimated beam can be achieved by making the distance between the lens and beam waist equal to the focal length.

Microscope

A microscope is an optical instrument that contains lenses and can enlarge the image of the sample placed in the focal plane. The two main components of a microscope are the objective and eyepiece[33]. The objective, made of compound lenses, is located closest to the sample under observation and forms a real image for the eyepiece. The eyepiece projects the image to the detector. If the light beam which emits from the object and passes through the objective directly forms a real image, then the design is called finite conjugate. If the light beam is collimated after passing through the objective, then it is an infinite conjugate optical design. To form a real image, the infinite conjugate microscope needs an extra tube lens between the objective and eyepiece. The infinite conjugate design is more convenient to add other optical components such as beam splitters, polarizers, and filters into the optical path.
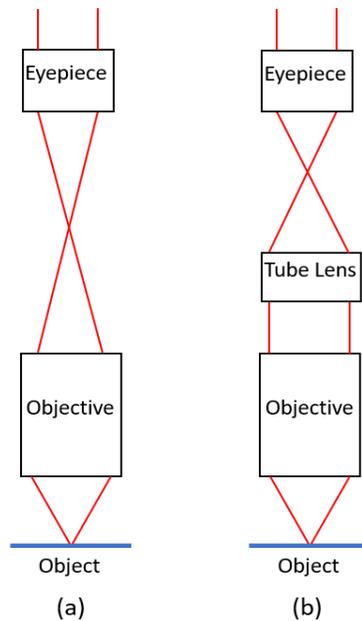
Fig. 3.10. (a) Finite conjugate and (b) infinite conjugate microscopes.

The magnification of the microscope system is equal to the product of the magnifications of the objective and eyepiece [Eq. (3.13)].

$$m_{system} = m_{objective} \times m_{eyepiece} \tag{3.13}$$

For an infinite conjugate microscope, the magnification of the objective is given by:

$$m_{objective} = \frac{f_{tubelens}}{f_{objective}} \tag{3.14}$$

where the $f_{tubelens}$ and $f_{objective}$ are the focal lengths of the tube lens and the objective.

A digital camera such as charge-coupled device (CCD) or complementary metal-oxide semiconductor (CMOS) camera can be used as the detector, and the eyepiece is not necessary. The camera sensor panel is located at the focal plane of the tube lens where the real image of the object is formed.
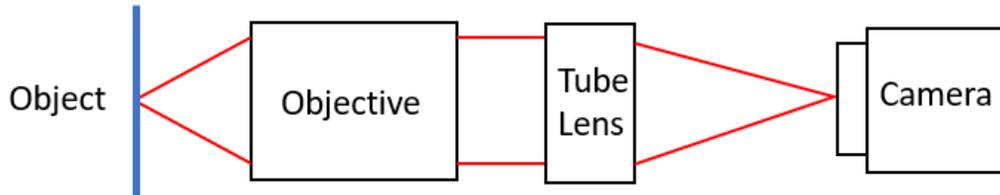
Fig. 3.11. Schematic of a microscope with digital camera as detector.

The resolving power R is defined as the minimum distance between two points which are separated and can still been recognized distinctly. The equation for R is given by:

$$R = \frac{1.22\lambda}{2\,\text{NA}} \tag{3.15}$$

where λ is the wavelength of the incident light and NA is the numerical aperture. The value of NA is given by[33]:

$$NA = n \sin \theta \qquad (3.16)$$

where n is the index of refraction, and θ is half of the vertical angle of the light cone.

Confocal Laser Scanning Microscope

As discussed in the last section, a microscope should have the object under observation placed in the focal plane of objective. However, for a thick sample, the light from above and below the focal plane will also been detected. This kind of light will blur the image. Confocal microscopy is a technique to reject the out-of-focus light and increase the image resolution. The idea was first introduced by Marvin Minsky[34], who designed a confocal microscope by adding two pinhole apertures before the light source and the detector. The detector can thus only detect the light from the focusing point.

Confocal laser scanning microscopy (CLSM)[35] is the most common type in commercial usage. It plays an important role in life sciences such as cell imaging[36] and tissue imaging[37]. CLSM is also useful in physical science areas such as point defects in semiconductors[38]. The light source is a laser, and usually the detector is a photomultiplier tube (PMT). With the pinhole in front of the detector, only the photons coming from the focal point will be detected. By moving the stage in the (*x, y*) plane, a resulting image can be formed. A schematic of the CLSM is shown in Fig. 3.12.
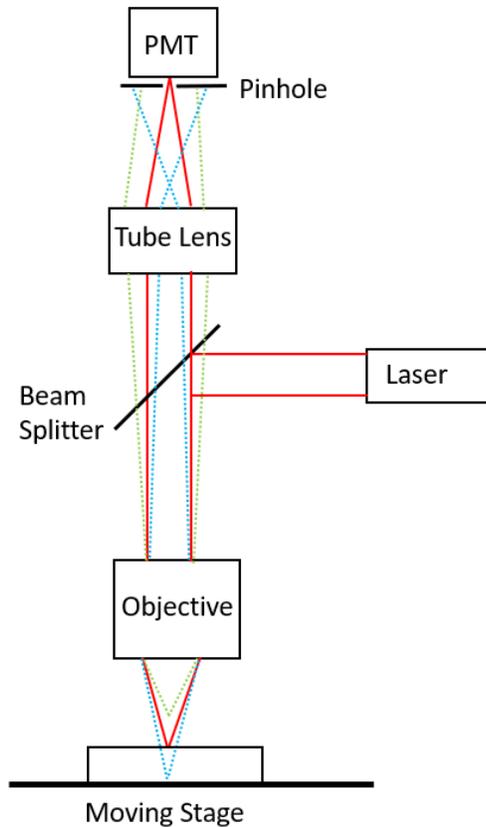
Fig. 3.12. Schematic of a CLSM. Light from the focal plane is detected by

the PMT, while the light from above and below the focal plane is rejected by the

pinhole.

The CLSM can form images with high resolution. However, to achieve this, the pinhole

aperture should be tiny, which makes the alignment of the optical components difficult. In

addition, the PMT requires a high-voltage power supply.

Ye and McCluskey[39] proposed a modular CLSM design which uses an off-the-shelf

digital camera (CCD or CMOS) to replace the physical pinhole and PMT. In their design, the

light emitted from the focal plane refocuses on the CCD sensor panel. A virtual pinhole can be

created by selecting the region of the interest on the recorded image. In a $N \times N$ pixel area cropped around the focusing center, the image moment is calculated to analyze the image. The equation of the image moment is given by[40]:

$$M_{pq} = \sum_x \sum_y x^q y^p I(x, y) \tag{3.17}$$

where $I(x, y)$ is the intensity of the pixel at (x, y). The CLSM with digital camera does not require precise alignment, as the position of virtual pinhole could be decided after the data are collected.
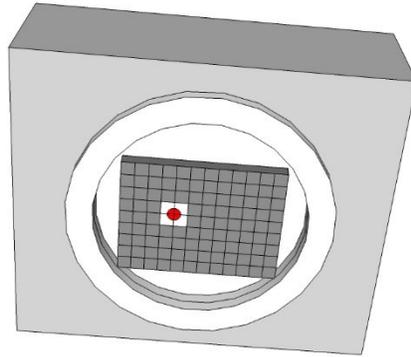
Fig. 3.13. A virtual pinhole is created by selecting certain pixels on the camera panel.

CHAPTER FOUR: CONFOCAL MICROSCOPY WITH A MICROLENS ARRAY

YINCHUAN YU,[1] XIANJUN YE,[1,2] MATTHEW D. MCCLUSKEY[1]

[1]*Department of Physics and Astronomy, Washington State University, Pullman, WA 99164-2814, USA*

[2]*Department of Anatomy and Structural Biology, Gruss Lipper Biophotonics Center, Albert Einstein College of Medicine, Bronx, NY 10461, USA*

Abstract

Confocal laser scanning microscopy (CLSM) is a preferred method for obtaining optical images with submicron resolution. Replacing the pinhole and detector of a CLSM with a digital camera (CCD or CMOS) has the potential to simplify the design and reduce cost. However, the relatively slow speed of a typical camera results in long scans. To address this issue, in the present investigation a microlens array (MLA) was used to split the laser beam into 48 beamlets that are focused onto the sample. In essence, 48 pinhole-detector measurements were performed in parallel. Images obtained from the 48 laser spots were stitched together into a final image.

Introduction

Confocal microscopy[34] and multiphoton microscopy[42] are among the most popular

imaging modalities because of their superior optical sectioning capability[43]. These microscopies

have found their way into a wide range of life science[37, 44, 45 ,46, 47] and materials science[35, 48, 49, 50,

51] applications. The ability to detect single-molecule fluorescence has provided a wealth of

information and high-resolution images[52,53,54,55]. Multiphoton microscopy, despite its advantages

(deeper penetration depth and better signal to noise ratio), is expensive due to its requirement of

a high-intensity laser source[56]. A basic confocal microscopy system uses a relatively inexpensive

continuous wave laser[57]. However, a precise scanning/de-scanning system is required to guide

the emitted light through the pinhole, increasing alignment requirements and cost.

Confocal laser scanning microscopy (CLSM) can operate in fluorescent mode, where the

collected light has a longer wavelength than the laser, or reflection mode, where the laser light

itself is detected. The microscope in the present study operates in reflection mode. Ye and

McCluskey[39,58] proposed a modular CLSM design which uses an off-the-shelf digital camera

(CCD or CMOS) to replace the physical pinhole and photomultiplier tube. The confocal

microscope, as a popular base platform, can have other functionalities added to expand its

versatility. For example, a spectroscopic imaging module enables the scanning confocal

microscope to do photoluminescence mapping of two-dimensional nanomaterials[59].

Prior work showed that image moment analysis of properly cropped wide-field images in

CCD confocal microscopy can yield comparable performance to conventional confocal

microscopy[39], including optical sectioning[60]. Subtractive imaging together with Gaussian fits

provide further enhancement to the imaging quality[61]. A major drawback of this method is the

fairly slow scanning speed. In this work, we introduce a microlens array (MLA) into the incident

beam path. This splits the laser into a grid of beamlets, significantly shortening the scanning time.

MLAs have been used in wavefront sensors[62], light field microscopy[63], multifocal multiphoton microscopy[64], vibrational spectroscopy microscopy[65], and confocal microscopy[66,67,68]. In our work, the MLA is inserted into the incident beam path as an intermediate optical element of the CMOS confocal system, which uses a standard microscope objective and is (physical) pinhole-free. The entire area of the CMOS array is utilized.

Spinning-disk methods use an array of pinholes, rather than a single pinhole, and a detector records the light intensities from the various pinholes[69]. Favro et al.[70] and the Yokogawa Electrical Corporation[71] disclosed a microlens array disk coupled to a pinhole array disk in order to improve light collection efficiency. A similar approach to focus light through a pinhole array was described by Hell et al[72]. Our method dispenses with pinholes entirely and is thus distinct from these approaches. The lack of a spinning disk has the potential to reduce cost.

Experiment

The system is a modification of the confocal microscope described in Ref. 42. The apparatus was built with the Thorlabs 30 mm cage system. The light source is a 4.5 mW power, 532 nm wavelength collimated laser. A Keplerian style beam expander is placed after the light source to expand the beam in order to overfill the microlens array. The microlens array (Thorlabs MLA150-5C) consists of a 10 mm × 10 mm square grid of plano-convex lenses on a fused silica substrate. The distance between microlenses, or pitch, is 150 μm, and the focal length of each

microlens is 5.6 mm. The MLA splits the beam into a grid of beamlets and a 200 mm focal length lens is used to collimate these beamlets. The beamlets are then focused on the sample by an 8.2 mm focal length objective lens (20×, numerical aperture = 0.4).

After reflection by the sample, the laser beams are guided by beam splitter cubes into the camera detection module. There are two cameras, an Imaging Source DMK 33UP1300 monochrome camera (Camera1) placed on the vertical arm to collect the reflected laser spots, and a DFK 23U274 color camera (Camera2) on the horizontal arm for sample inspection. The sample is moved by the motion module, which contains a piezoelectric position stage controlling the x-axis and y-axis, a piezoelectric objective scanner controlling the z-axis, and a 3-axis manual stage for initial position control.

λ/2: half wave plate

BS: beam splitter

BE: beam expander

LF: line filter

LL: laser lens

M: mirror

MLA: microlens array

ND: neutral density filter
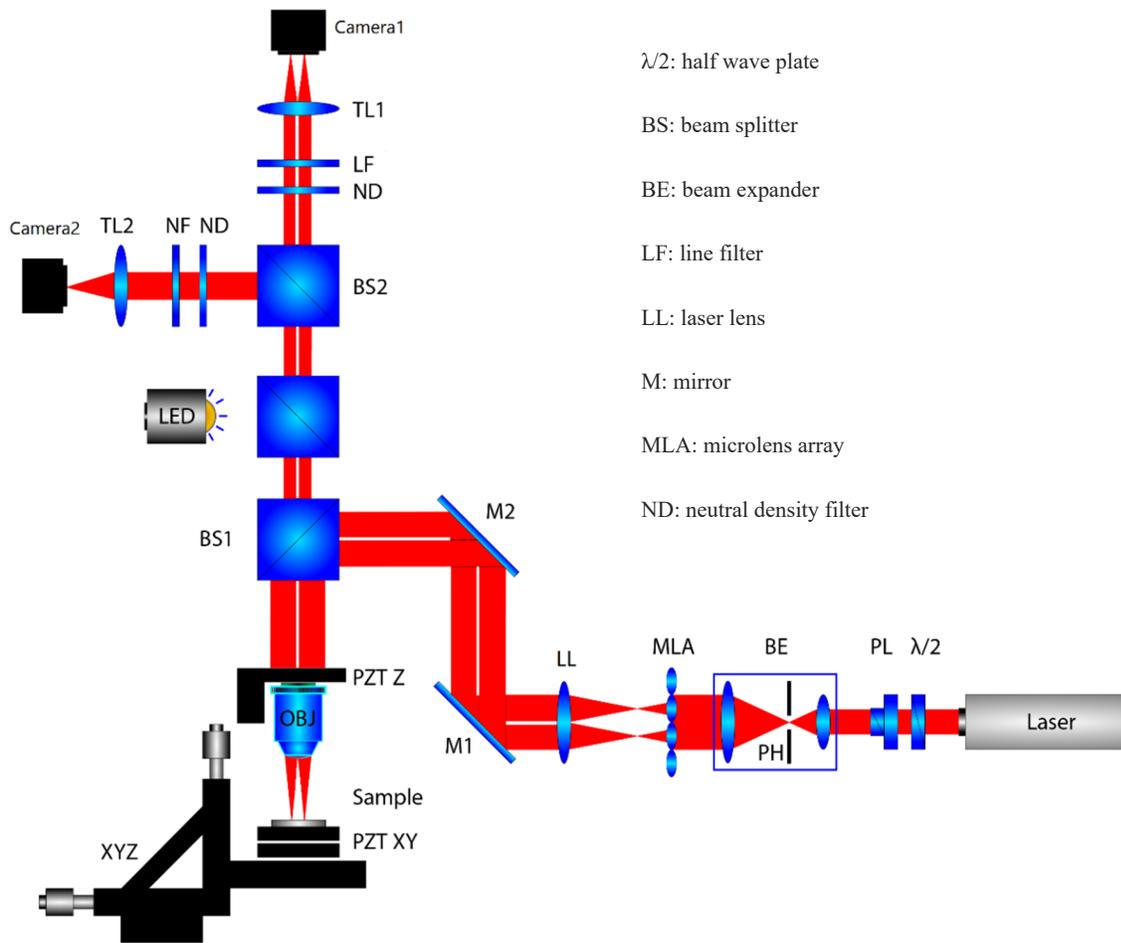
Fig. 4.1 Schematic diagram of the microscope system. A laser beam passes through the microlens array (MLA) and lens (LL). A beam splitter (BS1) directs the beamlets to the objective lens (OBJ), which focuses them on the sample. Camera2 is a color camera for widefield microscopy. Along with the light-emitting diode (LED), it is used for sample inspection; i.e., choosing the region of interest. Camera1 is a monochrome camera for acquiring images of reflected laser spots.

The sample was a US Air Force resolution target (USAF Ready Optics, California, up to group 11). Data acquisition and scanning processes are controlled by a program written in C++.

The exposure time was set as 1/500 s and the scanning step was 0.05 μm. The number of steps was 200×200, or 10×10 μm. On Camera1, an image of 48 reflected laser spots is collected (Fig. 4.2).

The first step is to find the correct Z position where the sample is in the focal plane of the microscope. This is done by turning on the light-emitting diode (LED), observing the sample with Camera2, and adjusting the objective Z height until the sample surface is in-focus. The manual stage is used to select the region of interest. Next, the laser is turned on and the sample is scanned. The scanning time depends on the setting of the camera frequency. In our experiment the frequency was set as 30 frames per second (fps), which resulted in a total scanning time of approximately 20 min.
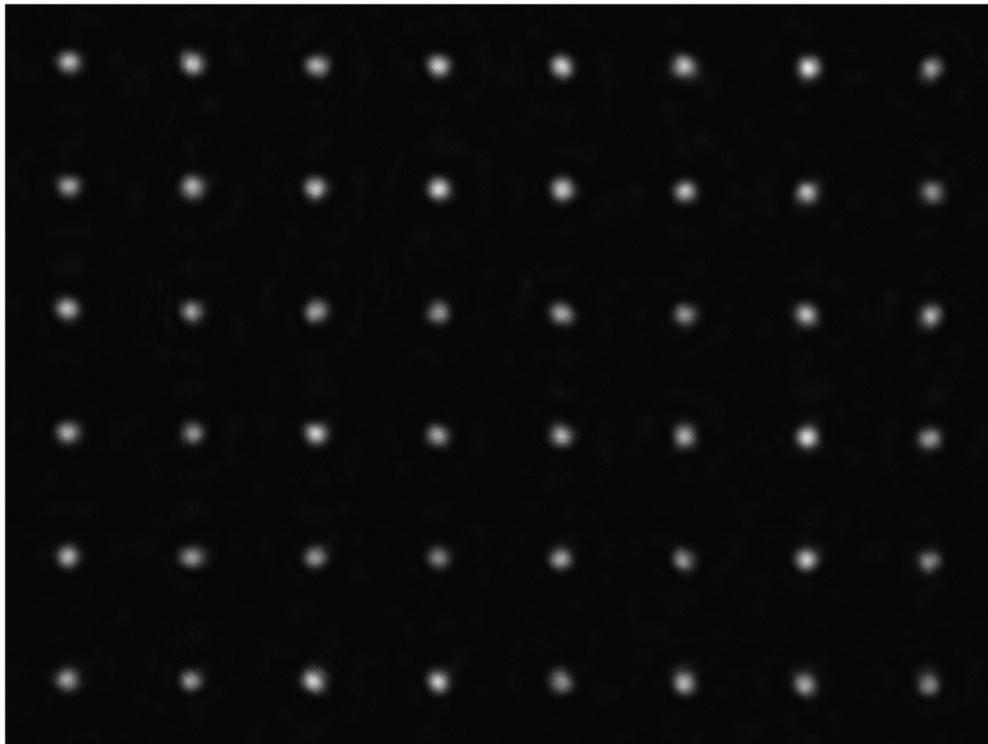
Fig. 4.2 Reflected laser spots imaged on Camera1.

Image Processing

Scanning and image moment

Before scanning the sample, a high-quality mirror was used to normalize the intensity of each laser spot. The distance between two lenses (pitch) on the microlens array is $d_l = 150$ μm. The distance between two laser spots on the sample is given by

$$d_s = d_l \frac{f_o}{f_l} \tag{4.1}$$

Here, f_l is the focal length of the laser lens and f_o is the focal length of the objective lens. For $f_l = 200$ mm and $f_o = 8.2$ mm, Eq. (4.1) yields a distance of $d_s = 6.15$ μm.

Since the step size is 0.05 μm, we need 123×123 steps for each laser spot to cover the whole picture. Because some overlap is required for stitching, however, we need to scan additional steps. In practice, 200×200 steps provide sufficient overlap between adjacent pieces. After scanning, each camera image is cropped evenly into forty-eight pieces, each of which is 160×160 pixels.

The image moment is defined as

$$M_{pq} = \sum_x \sum_y x^p y^q I(x, y) \tag{4.2}$$

where $I(x,y)$ is the intensity at pixel (x, y). The sum is performed over a 60×60 pixel region around the laser spot. From Eq. (4.2), the 0th order image moment M_{00} is the irradiance. By calculating each spot's 0th order image moment, forty-eight 200×200 matrices, or *graphs*, are formed.
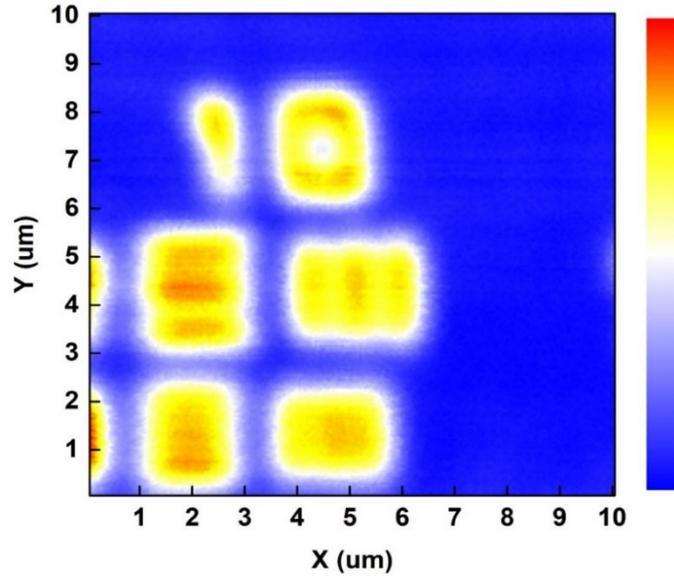
Fig. 4.3 Graph formed by one of the laser spots. Each pixel on the graph is determined by the value of 0th order image moment. The image shows part of the USAF target group 10.

Gaussian fits

An alternative analysis was performed by fitting the laser spots to Gaussian functions. The equation for a 2D Gaussian is:

$$g(x,y) = A \cdot \exp\left(-\frac{(x-\bar{x})^2}{2\sigma_x^{\,2}} - \frac{(y-\bar{y})^2}{2\sigma_y^{\,2}} \right)$$
(4.3)

where A is the peak intensity and $(\bar{x}, \bar{y})$ gives the spot's central position. An example of a fit to one laser spot is shown in Fig. 4.4. The difference between the experimental and simulated images, normalized to the maximum of the experimental image, is shown in Fig. 4.4(c).

(a)   (b)   (c)
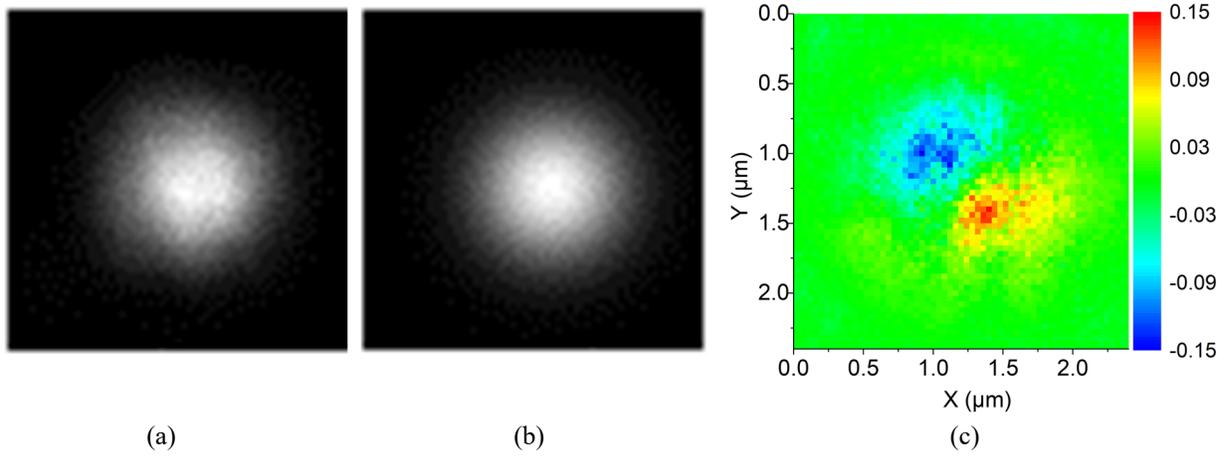
Fig. 4.4 (a) Laser spot obtained by Camera1. (b) 2D Gaussian fit. (c) False-color image of the normalized difference between the experimental spot and the fit.
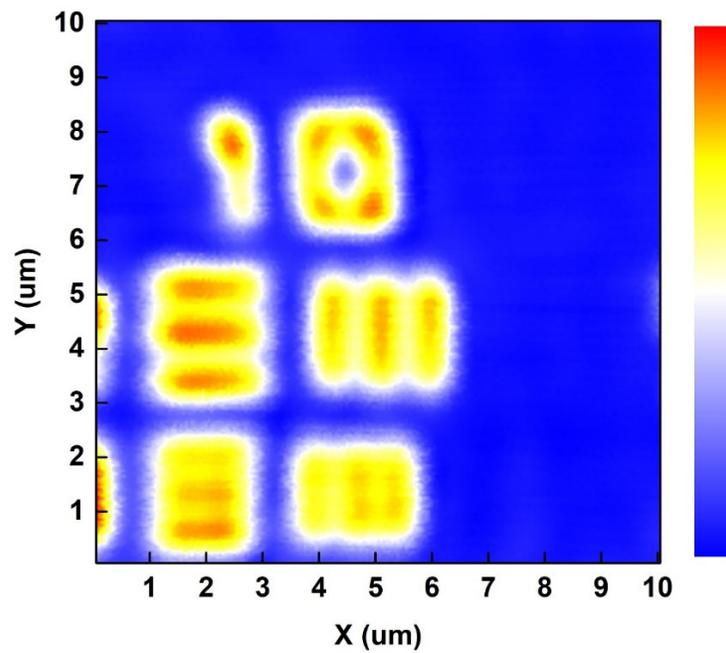
Fig. 4.5 Graph of the *A* value of the Gaussian fit. Each pixel on the graph is determined by the *A* value (amplitude) of a 2D Gaussian function. The scanning region is the same as Fig. 4.3.

This fit is performed for each step, and the A value (amplitude) is plotted in a graph. Fig. 4.5 shows a graph of the A value of the Gaussian fit, for the same scanning region as Fig. 4.3. Comparing Fig. 4.3 and Fig. 4.5, it is apparent that the graph of A is sharper than that of M_{00}. A 1-D slice of the image is plotted in Fig. 4.6. The highlighted region is the intensity of a line across three stripes from USAF group 10, element 2. These rectangular stripes and the gaps between them are each 0.435 μm wide[73]. A qualitative assessment of Fig. 4.6 indicates that the plot of the A value has lower noise and higher contrast.
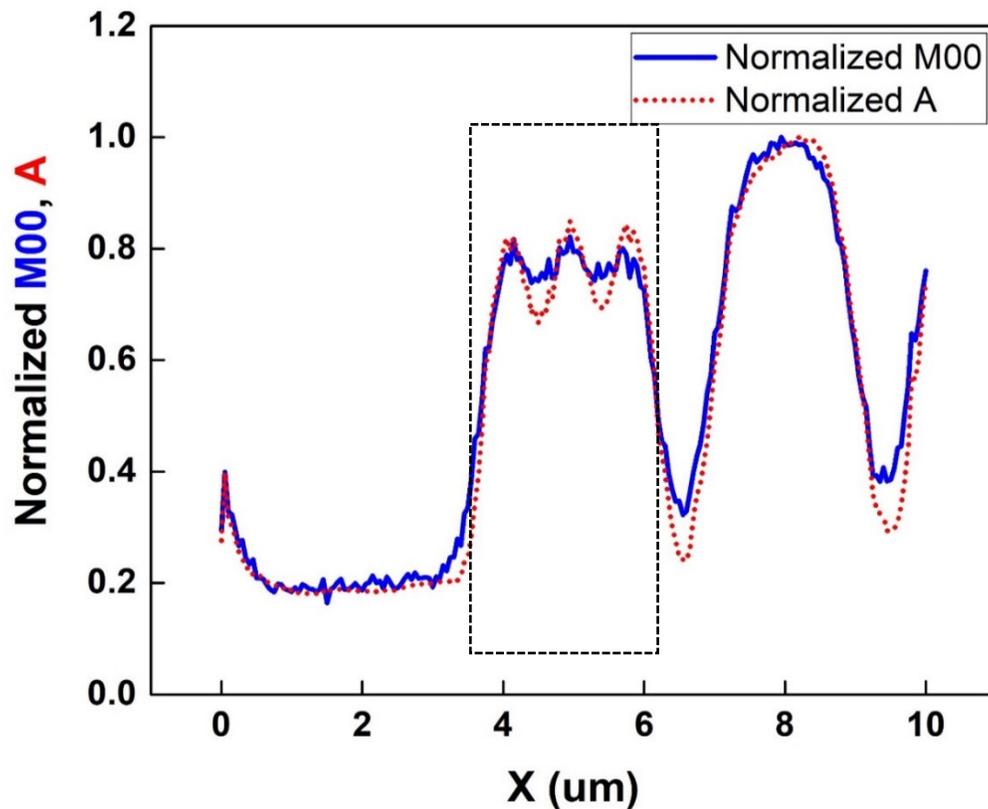
Fig. 4.6 Comparison of graphs made by the A value of the Gaussian fit and the 0th order image moment. The rectangular region is group 10, element 2.

Stitching Algorithm

The lenses on the microlens array are not perfectly uniform, which causes the laser spots to have position and intensity deviations. Simply combining 48 pieces will cause boundary discontinuities. To correct this artifact, an algorithm was introduced to smoothly join, or *stitch*, the 48 images. Each image was first multiplied by a normalization constant obtained from the mirror scan. For two neighboring pieces, rectangular portions were selected that should overlap (40×160 pixels for a vertical boundary, as shown in Fig. 4.7, 160×40 pixels for a horizontal boundary).

Plotting values of each pixel of one rectangular portion versus the values of the neighbor's portion, a linear regression was constructed [Fig. 4.8(a)]. The rectangular portion on the neighboring graph was moved until the best linearity was found, which returned a maximized r-squared value[74]. This relative position indicates how the neighboring piece should be translated. The r-squared value itself is a statistical measure of the quality of the linear regression.
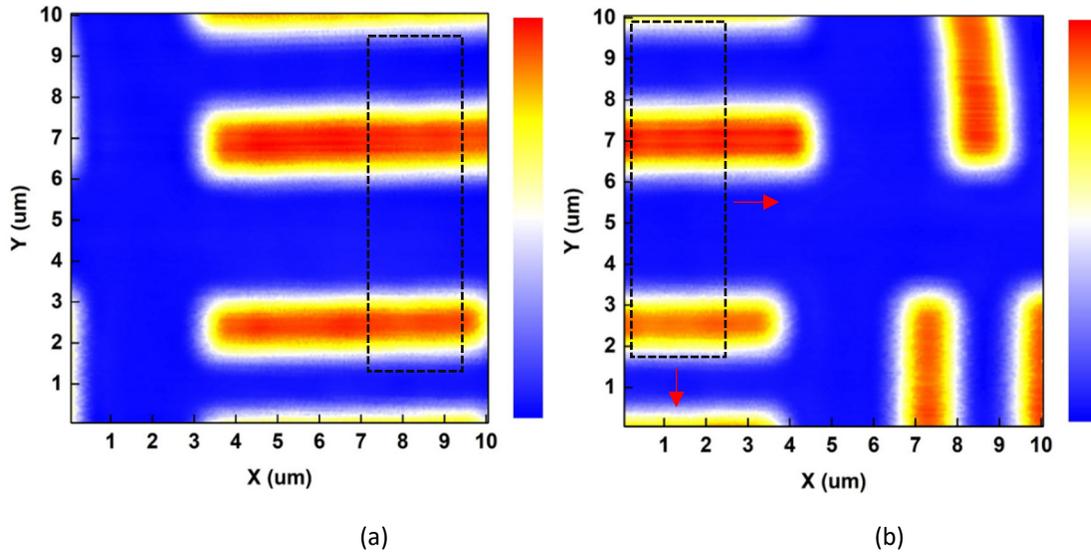
Fig. 4.7 Rectangular portions on two neighboring pieces. One of the

portions is moved until the best overlap is achieved.

(a)

(b)
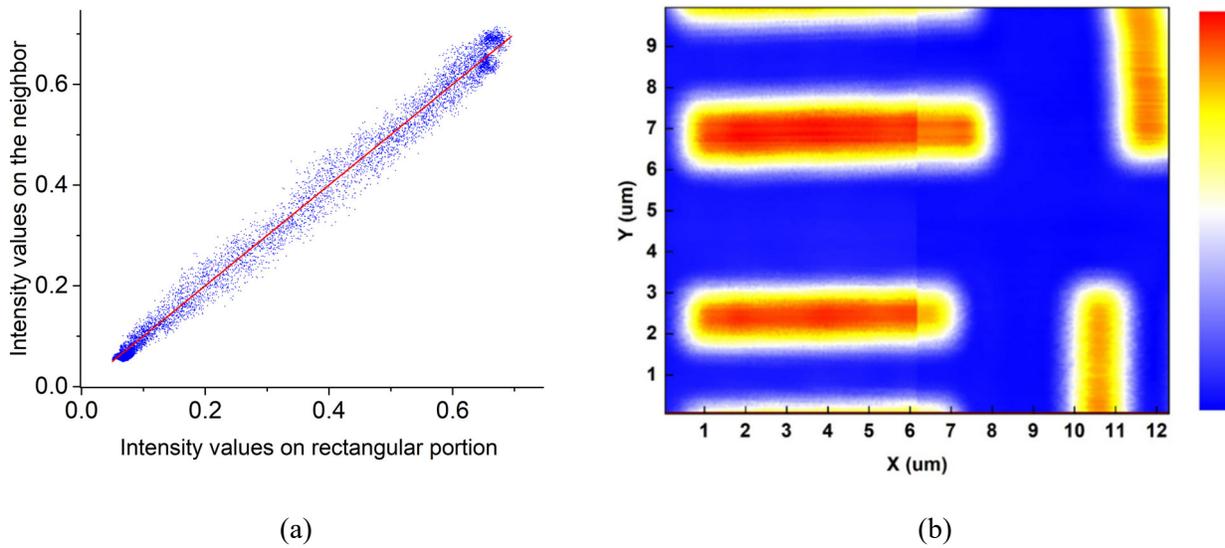
Fig. 4.8 (a) Linear fit with r-squared value closest to 1. (b) Result of

combining the two neighboring pieces based on the result.

Repeating this procedure for the boundaries of all 48 pieces, the whole graph is formed piece by piece, like a puzzle. In order to combine all pieces into a whole graph with minimum discontinuity and highest position accuracy, the piece that yields the highest r-squared value is added first. For example, consider the 1st piece, which is in the upper-left corner. There are two neighboring pieces, below and to the right. The linear regression procedure is performed for both of these neighboring pieces. Whichever one yields the highest r-squared value is added to the board. This procedure is repeated until the 48th piece is added.

Finally, boundary discontinuities are smoothed by adding a gradient value across a width of 20 pixels. The smoothing equation is:

$$I = I_0 + \tfrac{1}{2} D\left(1 - \tfrac{x}{10}\right) \tag{4.4}$$

where I is the adjusted intensity, I_0 is the original intensity, D is the difference between the intensity of two pixels on each side of the boundary, and x is the number of pixels away from the boundary. The graph of Fig. 4.8(b) after smoothing is shown in Fig. 4.9. The whole graph is shown in Fig. 4.10.
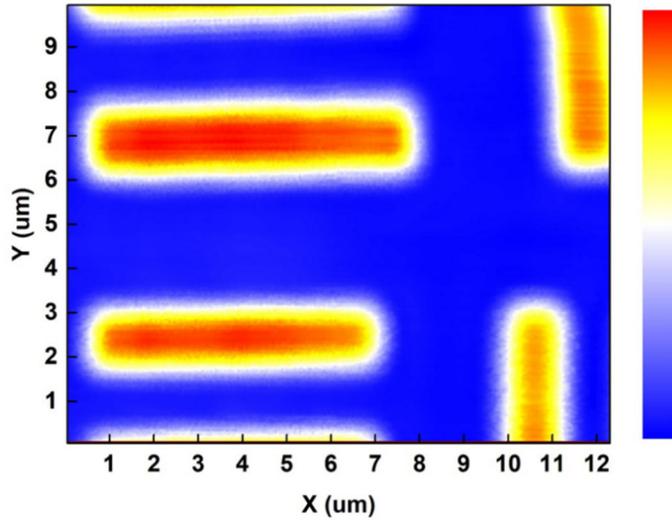
Fig. 4.9 The same graph as Fig. 4.8(b), with the boundary smoothed via Eq.

(4.4).
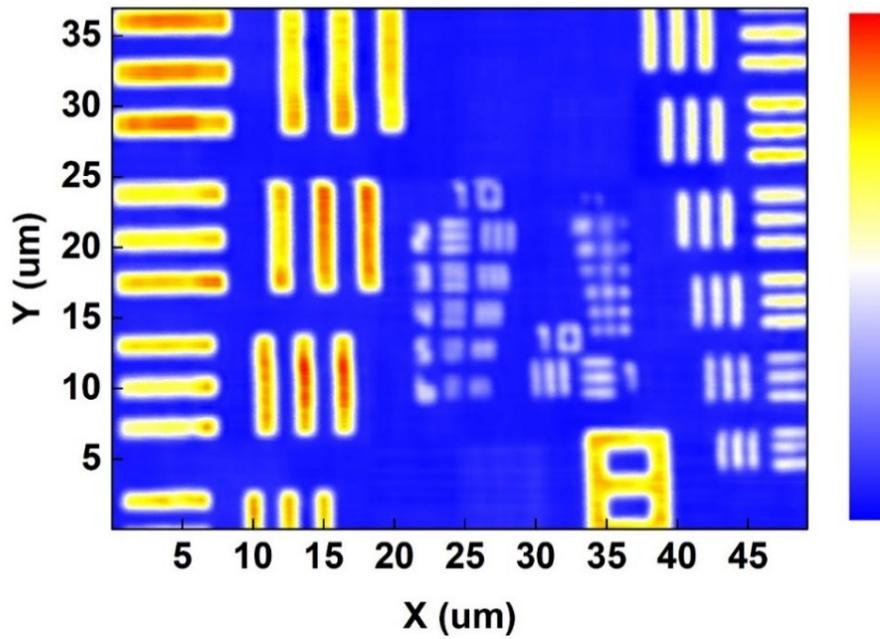
Fig. 4.10 The whole graph formed by the stitching method.

Conclusions

We have demonstrated that a microlens array confocal microscope produces images with submicron spatial resolution. In principle, using a microlens array could reduce scanning times arbitrarily, limited only by the camera frame rate and number of laser spots in the field of view. In our experiment, graphs made by Gaussian fits have higher contrast than those obtained using M_{00} (irradiance). Rectangular stripes separated by 0.4 μm can be resolved via the Gaussian-fit method. Stitching methods were used to minimize the boundary discontinuities. This method can be applied to fluorescence microscopy by placing an appropriate filter in front of the camera.

CHAPTER FIVE: CLASSIFICATION OF SEMICONDUCTORS BY PHOTOLUMINESCENCE SPECTROSCOPY AND MACHINE LEARNING

YINCHUAN YU,[1] MATTHEW D. MCCLUSKEY[1]

[1]*Department of Physics and Astronomy, Washington State University, Pullman, WA 99164-2814, USA*

Abstract

Photoluminescence (PL) spectroscopy is a non-destructive optical method that is widely used to characterize semiconductors. In the PL process, a substance absorbs photons and emits light with longer wavelengths. This paper discusses a method for identifying substances from their PL spectra using machine learning, a technique that is efficient in making classifications. Neural networks were constructed by taking simulated PL spectra as the input and the identity of the substance as the output. In this paper, six different semiconductors were chosen as categories: gallium oxide (Ga_2O_3), zinc oxide (ZnO), gallium nitride (GaN), cadmium sulfide (CdS), tungsten disulfide (WS_2) and cesium lead bromide ($CsPbBr_3$). The developed algorithm has a high accuracy (>90%) for assigning a substance to one of these six categories from its PL spectrum.

Introduction

Optical spectroscopy is a fast, nondestructive method for analyzing chemical compounds. Absorption or emission spectra provide a "fingerprint" of the material as well as its defects.

Typical wavelengths range from the UV to near-IR (200 to 1100 nm). When coupled with a microscope, spectra can be obtained with high spatial resolution. Types of optical spectroscopy include transmission, reflection, fluorescence, and Raman scattering[76].

The interpretation of spectra traditionally relies on the user's experience. A senior researcher might say, "This fluorescence spectrum has a broad green band and a sharp peak in the near-UV. That looks like zinc oxide, or maybe titania." That statement contains several concepts, including "spectrum," "green band," and "UV peak." These concepts led the researcher to assign high probabilities to zinc oxide and titania. Students or postdoctoral researchers operate in an apprentice mode and gradually become familiar with a set of spectral features relevant to their research.

In contrast to mass spectrometry and Raman spectroscopy databases[77], there are few tools available to identify fluorescence (or photoluminescence, PL) spectra of inorganic compounds. The main problem is that room-temperature PL spectra typically consist of broad, overlapping bands. The intensity and peak positions depend on excitation intensity, wavelength, and defects in the sample. Peak fitting or principal component analysis (PCA)[78] may not be optimal for such spectra. Machine learning (ML), which can readily group images or other datasets into general categories[79], has the potential to speed up identification of unknown substances. It can also provide a way to organize large collections of spectra, a valuable aid for researchers in chemistry and chemical engineering.

In the present work, a neural network was developed to identify PL spectra. Hornik[80] showed that multilayer feed-forward architecture gives neural networks the potential of being

universal approximators. Yarotsky[81] proved that deep Rectified Linear Unit (ReLU)[4] networks are more efficiently approximate functions than shallow networks. We may assume there is a function for which the input is the PL spectrum and the output is the substance category. Then the task is to build a deep ReLU neural networks to provide this function.

Methods

Simulated PL Spectra

The ML algorithm needs a large amount of data to get trained. To provide enough training data, simulated PL spectra were used. In general, the PL spectrum of substance has several emission bands, some of which are intrinsic features while others are caused by defects. To generate simulated PL data, the emission band wavelengths, and their relative strength and width, need to be described. In this work, the bi-gaussian function is used to describe each emission peak:

$$I = \begin{cases} A \exp\left(\frac{-(x-x_0)^2}{2\sigma_1^2}\right) & x < x_0 \\ A \exp\left(\frac{-(x-x_0)^2}{2\sigma_2^2}\right) & x > x_0 \end{cases} \tag{5.1}$$

where A is the relative peak height, x_0 is the peak center position, σ_1 and σ_2 are the left and right standard deviations that describe the asymmetric width of the peak.

An important example is gallium oxide (β-Ga_2O_3), an emerging material for power applications[82]. The PL of β-Ga_2O_3 typically has broad peaks that fall into four categories. The first one is the UV band, which is caused by recombination of an electron with a self-trapped

hole (STH). The peak center is located near 360 nm. A second peak near 400 nm (UV') and a third peak near 430 nm (blue) are due to donor-acceptor pair (DAP) transitions. The last peak at about 520 nm (green), also assigned to a DAP transition, is correlated with high O_2 pressure during the growth. From these peak positions, and randomized peak intensities and widths, simulated Ga_2O_3 PL spectra were generated. Four examples are shown in Fig. 5.1. Gaussian noise (mean 0, standard deviation 0.01) was included in the simulated spectra.
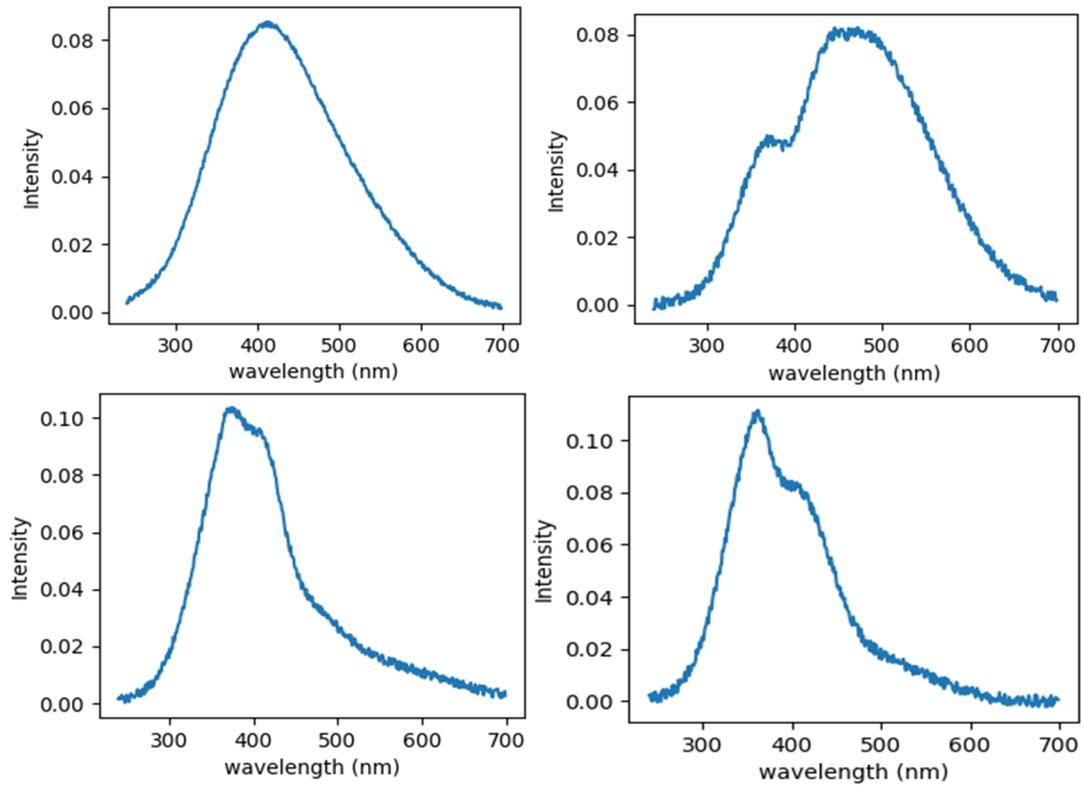
Fig. 5.1 Simulated PL spectra of Ga_2O_3 with randomized peak intensity and width.

The same idea was applied to other semiconductors with known emission properties. We restricted the data to PL spectra collected at room temperature. For ZnO, there is usually a UV peak at 378 nm due to free-exciton recombination and a broad green band near 530 nm due to defects[83]. GaN typically has a peak near 365 nm due to a band–to–band transition, and a small peak near 382 nm due to recombination of an electron in the conduction band with a hole bound to an acceptor[84]. Two broad yellow peaks near 482 nm and 525 nm caused by carbon-related defects are also observed[85]. The main peak of CdS[86], WS_2[87] and $CsPbBr_3$[88], at 508 nm, 630 nm, and 523 nm respectively, are due to exciton recombination.

Machine learning Algorithm

Simulated and real PL spectra are represented by column vectors. We focused on the PL spectrum between 250 nm to 649 nm with 1 nm as the interval, resulting in an $n = 400$ element vector. For spectra with less range, zero padding was used, where the intensity to was set 0 where values were not reported. To normalize the spectra, the vector was divided by its magnitude.

$$x = \frac{v}{\|v\|} \tag{5.2}$$

where x is the unit vector and provides the input for the training networks.

The output y is a vector, the dimension of which is the number of classes of substance used. The value of the element at the position of the known substance index is 1, and the rest are all 0. When m PL spectra are stacked one by one, a $n \times m$ matrix X is formed. Since the output target is already known, this ML approach is supervised learning, and a neural network is applied. There are two hidden layers. Each element in a layer is calculated by a linear

combination of all elements in the previous layer plus a bias value, followed by a nonlinear

activation function σ:

$$Z_l = W_l \cdot a_{l-1} + b_l \tag{5.3}$$

$$a_l = \sigma(Z_l) \tag{5.4}$$

where a_l is the vector of the lth layer, W_l is a parameter matrix with rows equal to the number of

elements in layer l and columns equal to the number in layer $l-1$, and b_l is a bias vector with

same number of elements as layer l.
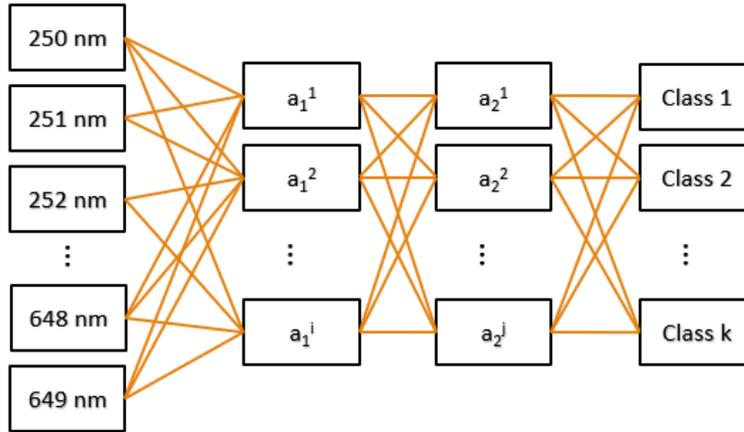
Fig. 5.2. Schematic diagram of neural networks used in this project.

The ReLU function is used as the activation function of the first two layers and a

SoftMax function is used as the activation function for the final layer:

$$\text{ReLU: } \sigma(x) = \max(0, x) \tag{5.5}$$

$$\text{SoftMax: } \sigma(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{k} e^{x_j}} \quad \text{for } i = 1, 2, 3, \ldots, k \tag{5.6}$$

The final layer is treated as the probability of the input belonging to each class; i.e., a prediction. For a known substance, the ideal output layer should be 1 on the corresponding class element and 0 for all other elements. We denote the output layer vector $\tilde{y}$ and the vector that represents the real class y. Both $\tilde{y}$ and y are vectors, the dimension of which is equal to the total number of classes. The Loss function describes how close the prediction $\tilde{y}$ is to the target y:

$$L = -\sum_{i=1}^{k}(y_i \ln(\tilde{y}_i)) \tag{5.7}$$

A small L value indicates a good prediction, with a perfect prediction giving $L = 0$. Averaging over all m training data, the total cost is

$$J = \frac{1}{m}\sum_{j=1}^{m} L_j \tag{5.8}$$

All W and b parameters are randomly generated at the beginning, so the predictions are far from the target and the cost is high. The method of gradient decent[89] is applied to push the Cost to 0. All parameters θ (W, b) are updated at the same time following the equation:

$$\theta := \theta - \alpha \cdot \nabla_\theta J \tag{5.9}$$

where α is the learning rate and needs to be set to an appropriately small value. Generating the Cost from the most recent parameters is a forward propagation. Updating all parameters from the most recent Cost is a backward propagation. One forward and backward propagation is one iteration of the training process. After enough iterations, the Cost converges to a constant, small value, and the parameters can be used to predict the class of a substance.

Results and Discussion

Each substance was represented by 1000 simulated spectra using bi-gaussian functions with the parameters shown in Table 5.1. The learning rate was set to be 0.01, the first hidden layer contained 50 elements, and second hidden layer contained 20 elements. After 20,000 iterations, the parameters were trained and could be used in predictions. A plot of the Cost as the function of number of iterations is shown in Fig. 5.3.

Substance	Peak Position (nm)	Relative Intensity	Width (nm)
Ga_2O_3	360	0 – 5	30 – 50
	400	0 – 5	40 – 60
	430	0 – 5	50 – 100
	520	0 – 1	25 - 100
ZnO	378	4 – 12	5 – 10
	530	1 – 12	30 – 60
GaN	365	6 – 16	5 - 10
	382	1 – 2	6 – 15
	525	1 – 12	25 – 50
CdS	508	1	5 – 10
WS_2	630	1	8 - 9
$CsPbBr_3$	523	1	10 – 30

Table 5.1. Parameters used in generating simulated PL spectra.
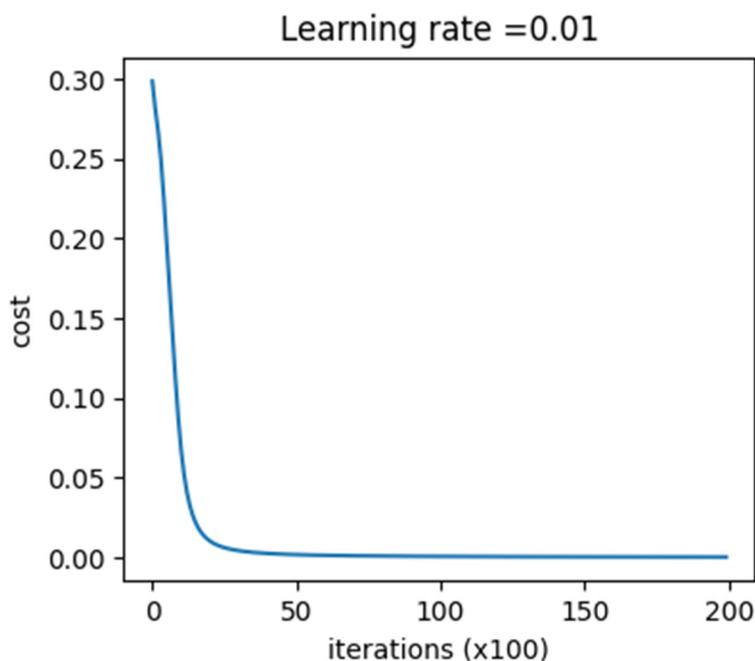
Fig. 5.3. Plot of the Cost as a function of iterations. The learning rate is 0.01 and 20,000 iterations were processed.

After the training process, the next task is to test how these parameters predict a new, unknown PL spectrum. The test is separated into two parts. The first part is to test with the PL spectra taken from literature, and the second part is to test with the PL collected in our lab using a Horiba Fluorolog spectrometer. By searching for papers with PL spectra, then digitalizing the PL with the GetData Graph[90] program, the PL data were collected. Applying the parameters obtained from the training process, the prediction of the PL spectra collected from literature approached 100% accuracy. Figs. 5.4-5.9 show the PL spectra collected from the literature.
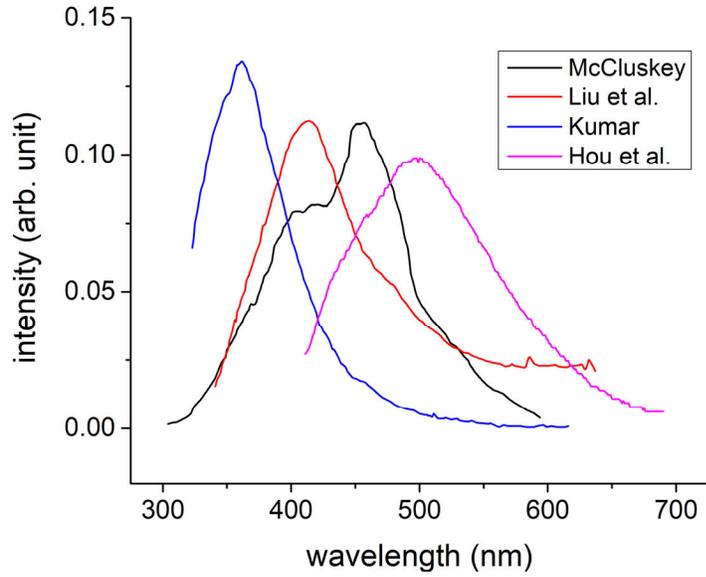
Fig. 5.4. PL spectra of Ga$_2$O$_3$ from McCluskey[82], Liu et al.[91], Kumar[92], and Hou et al.[93]
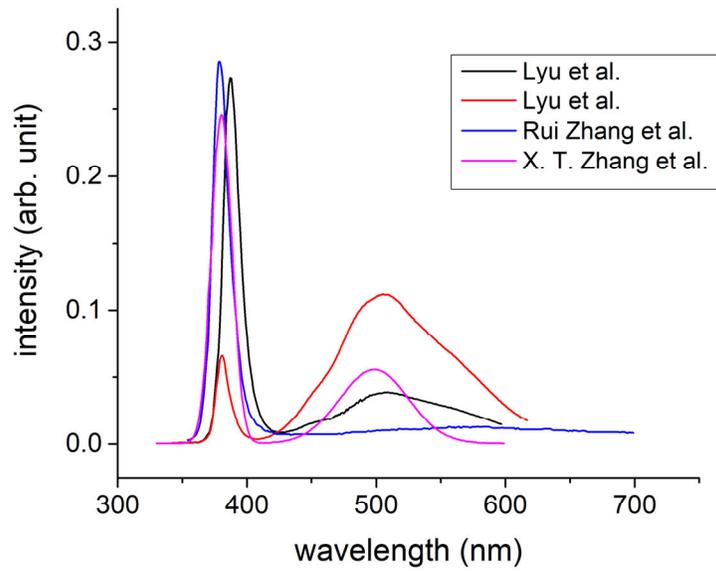
Fig. 5.5. PL spectra of ZnO from Lyu et al.[94], Rui Zhang et al.[83] ,and X. T. Zhang et al.[95]
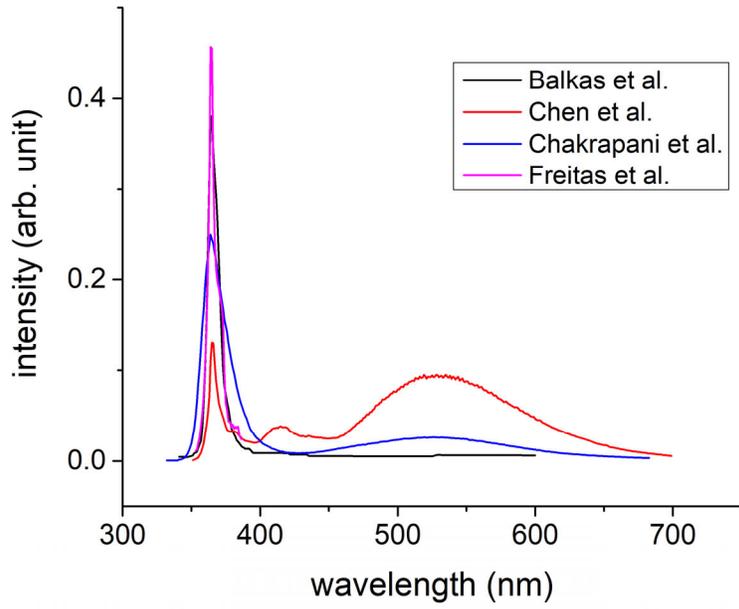
Fig. 5.6. PL spectra of GaN from Balkas et al.[96], Chen et al.[97], Chakrapani et al.[98], and Freitas et al.[84]
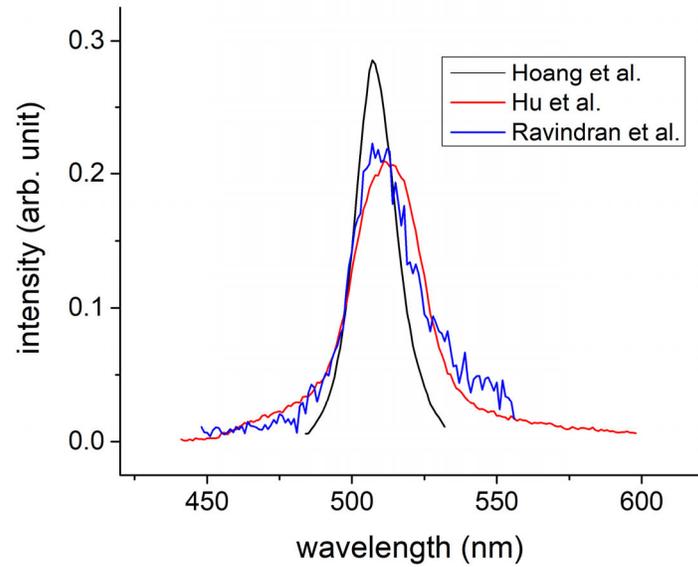
Fig. 5.7. PL spectra of CdS from Hoang et al.[86], Hu et al.[99], and Ravindran et al.[100]
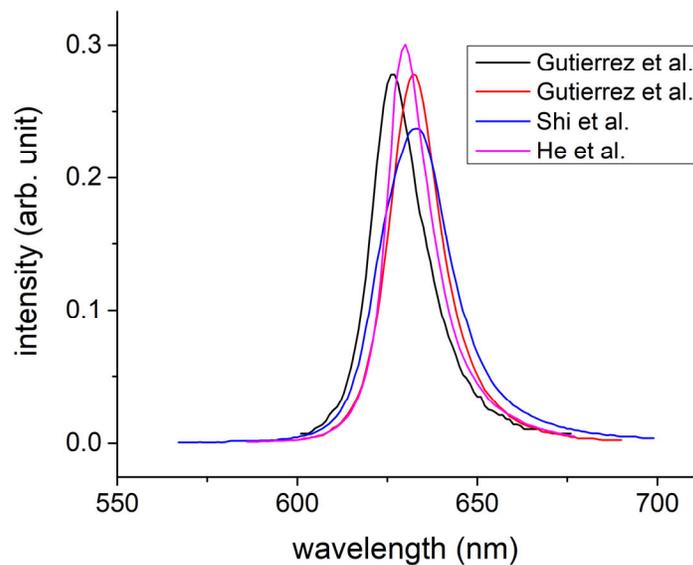
Fig. 5.8. PL spectra of WS_2 from Gutierrez et al.[87], Shi et al.[101], and He et al.[102]
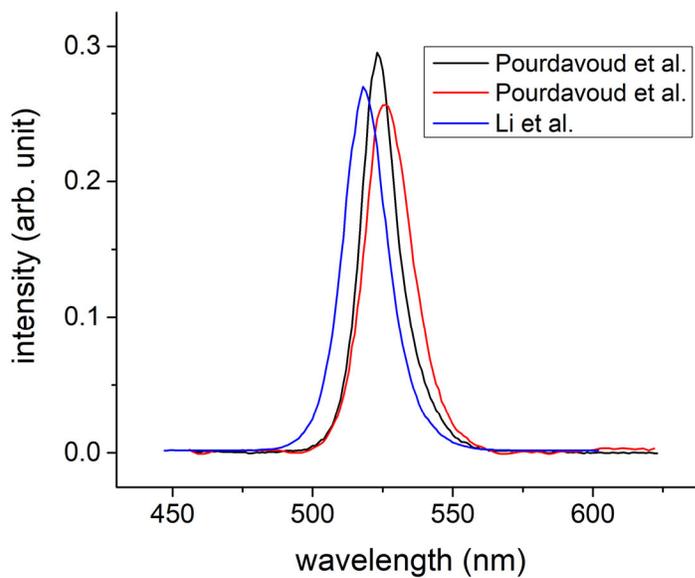
Fig. 5.9. PL spectra of $CsPbBr_3$ from Pourdavoud et al.[88] and Li et al.[103]

Category	Ga$_2$O$_3$	ZnO	GaN	CdS	WS$_2$	CsPbBr$_3$
	98.91%	0.09%	0.96%	0.02%	0.00%	0.02%
	0.63%	**99.29%**	0.00%	0.00%	0.00%	0.07%
	0.19%	**92.03%**	7.18%	0.00%	0.55%	0.04%
	0.02%	4.15%	**95.59%**	0.00%	0.24%	0.00%

	0.13%	0.02%	0.00%	**99.33%**	0.01%	0.51%

Table 5.2. PL collected from Horiba Fluorolog spectrometer and predictions given by the ML algorithm.

The second part was to use machine learning to identify semiconductors measured in our lab. The following excitation wavelengths were used: 270 nm (Ga2O3), 320 nm (ZnO #1 and GaN), 310 nm (ZnO #2), and 370 nm (CdS). Table 5.2 shows the plots of the PL spectra and the suggestions (predictions) given by the ML algorithm. The predictions include the percentage for the sample belonging to each category. As shown in bold, the percentages for correct categories exceeded 90%. To further test the ML algorithm, we performed a PL measurement on a Zn-doped Ga_2O_3 sample. The PL spectrum has signatures from Ga_2O_3 and ZnO.

Ga$_2$O$_3$	ZnO	GaN	CdS	WS$_2$	CsPbBr$_3$
61.09%	26.09%	12.42%	0.05%	0.05%	0.30%
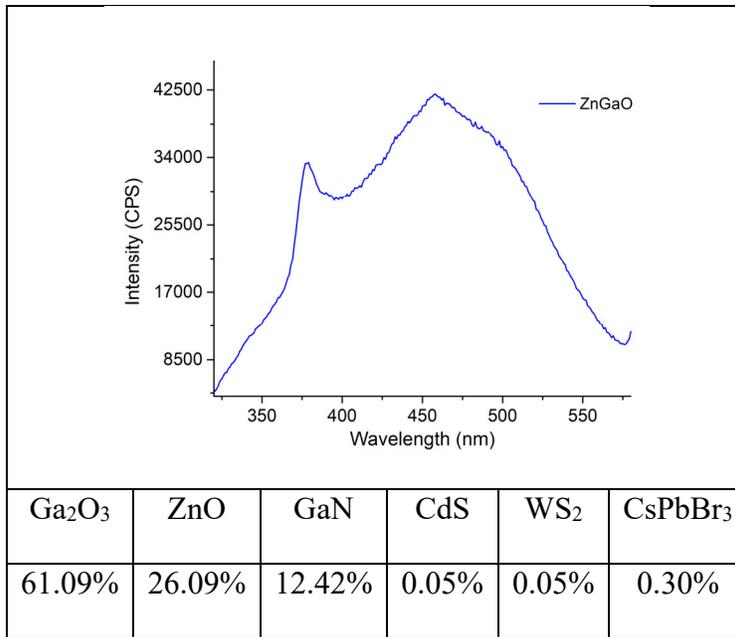
Table 5.3. PL of Zn doped Ga$_2$O$_3$ collected from Horiba Fluorolog spectrometer and the prediction given by the neural networks.

From Table 5.3, we can see that for an unknown PL spectrum, the predictions for the categories which contribute to the combined PL are higher than others. The prediction shows the probability of Ga$_2$O$_3$ is 61.09%, which is the highest. This make sense because the main constituent of the sample is Ga$_2$O$_3$. The probability for ZnO is 26.09%, which is the second highest. This result is sensible because the Zn-doped sample also shows the PL signature of phase-separated ZnO (the 378 nm peak).

Conclusions

PL spectroscopy combined with machine learning is a potential method for identifying unknown substances or providing an organization scheme for a large number of spectra. The forward network approach enables the user to assign probabilities to a substance belonging to different categories depending on its PL spectrum. Due to the relative lack of PL spectra in the literature, simulated PL spectra must be used to train the ML algorithm. Although only six categories were used in this preliminary study, the number of categories can be readily scaled up to provide suggestions for more substances and differentiate between substances with very similar PL spectra.

Acknowledgments

CHAPTER SIX: AUTOFOCUSING OF A CONFOCAL LASER MICROSCOPE USING CONVOLUTIONAL NEURAL NETWORKS

Abstract

Confocal laser scanning microscopy (CLSM) with a digital camera is a useful method for characterizing material surfaces. With a XY stage, a CLSM can scan a large area on a sample. Adjusting the height of the objective is necessary which made the laser beam could focus on the sample surface. However, if the surface of the sample is not flat, the laser spot will go in and out of focus, causing bad scanning results. Deep learning especially convolutional neural networks is an efficient way to treat images. It shows its success in the field of object detection[104], image classification[105], face recognition[106], etc. This chapter discusses using the deep learning techniques to design a model that predicts the out-of-focus distance with the image of laser spot. The model can develop to a system that could automatically focusing the CLSM in real time.

Introduction

The CLSM increases the image resolution by rejecting the light from above and below the focal plane. Therefore, maintaining the region of interest on the focal plane is important. Pre-adjusting the height of the objective is required to make the sample surface on the focal plane. During the scan, if the sample surface is not flat, and the changing height is larger than the depth of focus of the objective, the resulting image will be blurred. With a large scanning area and small scan step size, a whole scanning process could take hours or even days. Thus, an automatic system to adjust the objective height to maintain the sample on the focal plane is necessary. With

an off-the-shelf digital camera as detector, the CLSM could acquire the image of the laser spot continuously during the scan. Therefore, designing a method to predict the distance from the sample surface to the focal plane with the image of laser spot can solve this problem.

Some sample contains materials with different reflectivity locating at different area. At the position of height change, or boundary of different materials, the laser spot is affected and not in the shape of a circle. In such cases, a simple method such as using the intensity or calculating the image momentum may not work well. Deep learning with convolutional neural networks handles tasks with image as input very well. A model could be constructed with the image of laser spot as input and prediction of the out-of-focus distance as output.

Apparatus Setup and Sample

The system is a modification of the confocal microscope described in chapter 4 by removing the microlens array. The images of laser spot are collected by an Imaging Source DMK 33UP1300 monochrome camera (Camera 1), and there is a DFK 23U274 color camera (Camera2) for sample inspection. The light source is a 4.5 mW power, 532 nm wavelength collimated laser. The laser beams are focused on the sample by an 8.2 mm focal length objective lens (20×, numerical aperture = 0.4). The sample is placed on a piezoelectric position stage which has a *x-y* position control. The schematic diagram is shown in Fig. 6.1.

λ/2: half wave plate

BS: beam splitter

BE: beam expander

LF: line filter

M: mirror

ND: neutral density filter

NF: notch filter

PH: pinhole

PL: polarizer

PZT: piezo stage

TL: tube lens
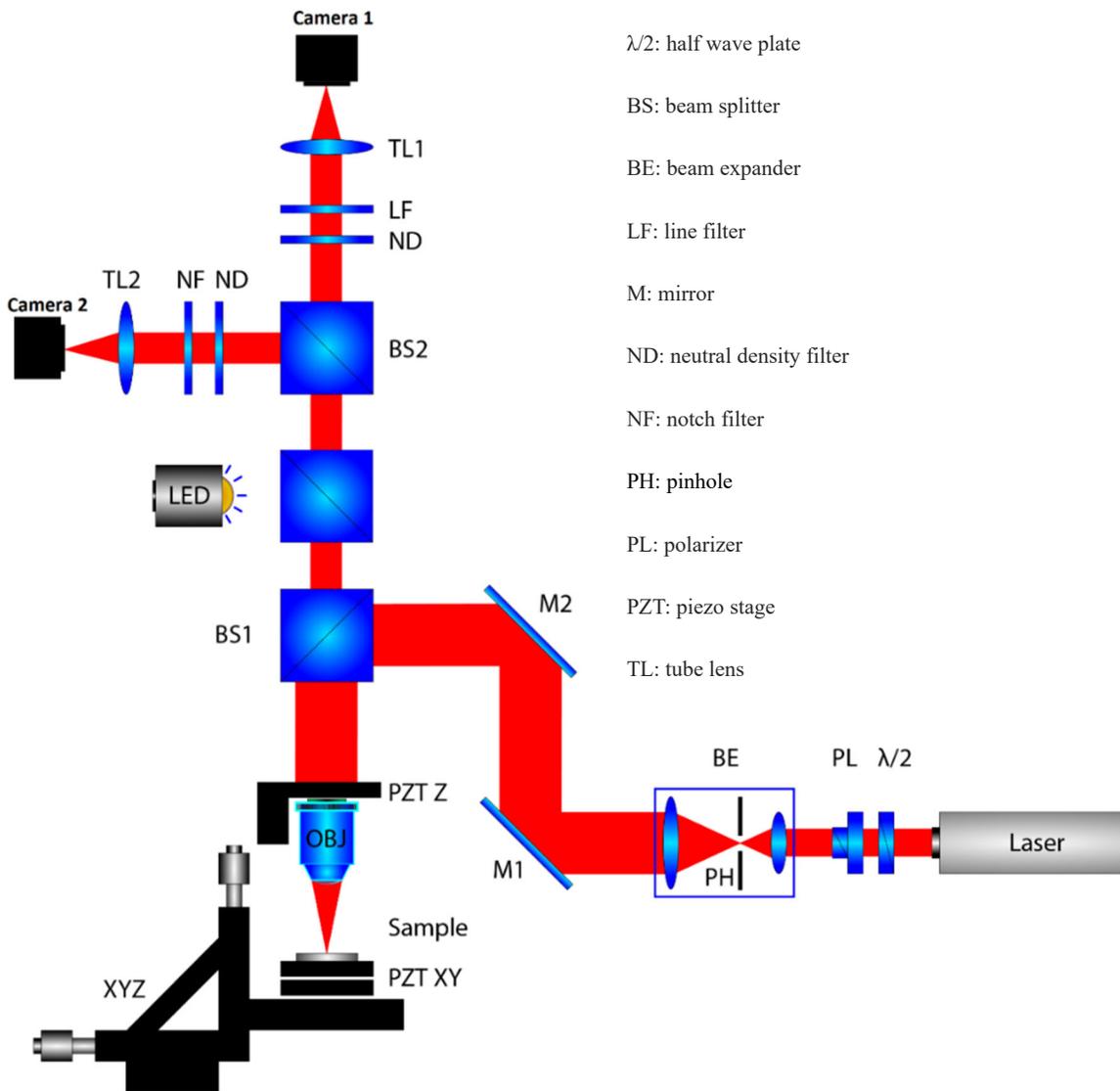
Fig. 6.1 Schematic diagram of the microscope system. It is modified of the system in Fig. 4.1 by removing the microlens array.

The sample we use is a semiconductor chip carrier that has gold patterns on silicon. Two areas with "cross" patterns were chosen, one used for training and one use for cross-validation[10] and testing (Fig. 6.2).

(a)　　　　　　　　　　　(b)　　　　　　　　　　　(c)
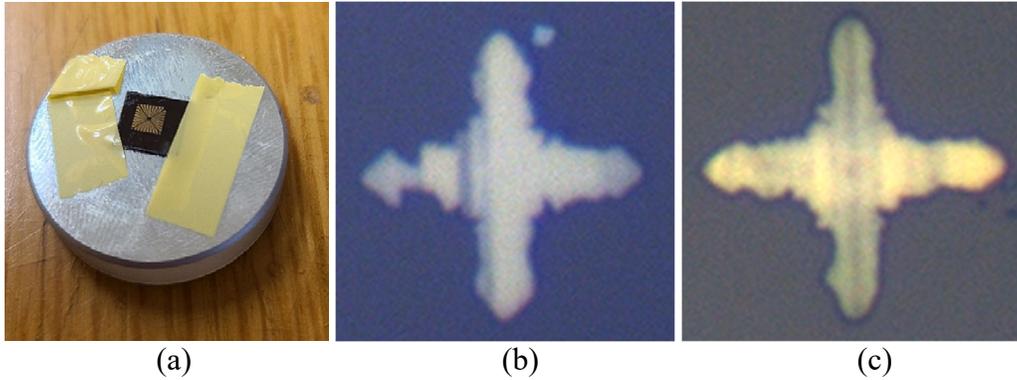
Fig. 6.2. (a) Photograph of the sample. (b) and (c) Images of two Golden

"cross" patterns at different positions on the sample collected by Camera 2.

Image Collection

The manual stage was adjusted until the surface of the sample was near the focal plane of

the objective. The piezoelectric position stage scanned 300 steps in each direction on the X and

Y axis, with a step size of 0.5 μm. The objective height was increased from -10 μm to 10 μm

about the focusing plane on the Z axis, with a step size of 0.1 μm for the training set and 0.5 μm

for the testing set. Therefore, there are 900 total (x, y) positions in a 15 μm by 15μm region on

the sample surface. On each (x, y) position, 200 images were taken for the training set and 20

images were taken for testing set. The horizontal steps were small such that the dataset included

laser spots reflecting on edges between the two materials. Examples of different laser spots are

shown in Fig. 6.3. Images of one laser spot at different heights are shown in Fig. 6.4.
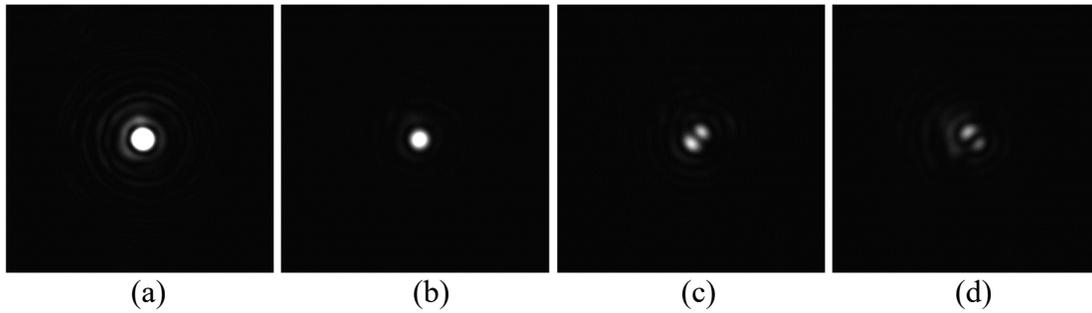
Fig. 6.3 Images of laser spots during the scan. (a) Laser reflected from gold. (b) Laser spot reflected from silicon. (c) and (d) Laser spots reflected from an edge between the two substances.
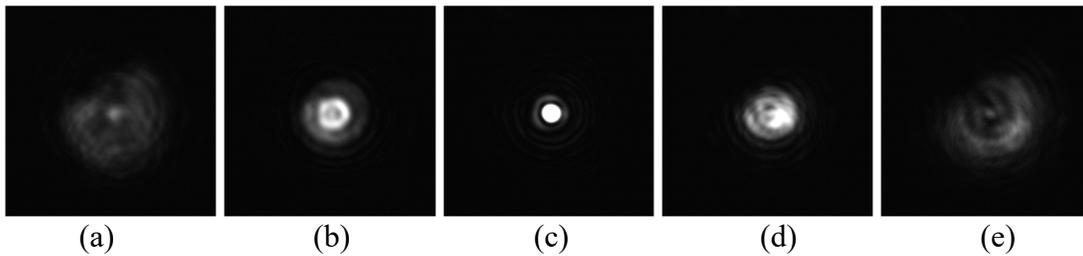
Fig.6. 4 Images of laser spots at different Z heights. They are (a) -6 μm, (b) -3 μm, (c) 0 μm, (d) 3 μm, (e) 6 μm from the focal plane.

Learning Model

In order to make an autofocusing system, the model needs to read an image of the laser spot and give a prediction of how far the sample surface is from the focal plane. The input of the model is a 400 by 400 pixel laser spot image, and the output is the distance from the focusing plane. As discussed in Chapter 1, the convolutional (conv) neural network is formed by layers including convolutional layers, max pooling layers and fully connected layers. In this model,

there are 4 groups of conv layers, and each group is followed by a max pooling layer. At the end

of the model, there are two fully connected layers which propagate the feature map to the output.

All convolutional layers are activated by the rectified linear unit (ReLU)[4] activation function.

Each convolution layer contains N $3 \times 3 \times n$ size conv filters. The value n is the number of

channels of the previous feature map (or image), and the number of conv filters N determines the

number of channels in the resulting feature map.
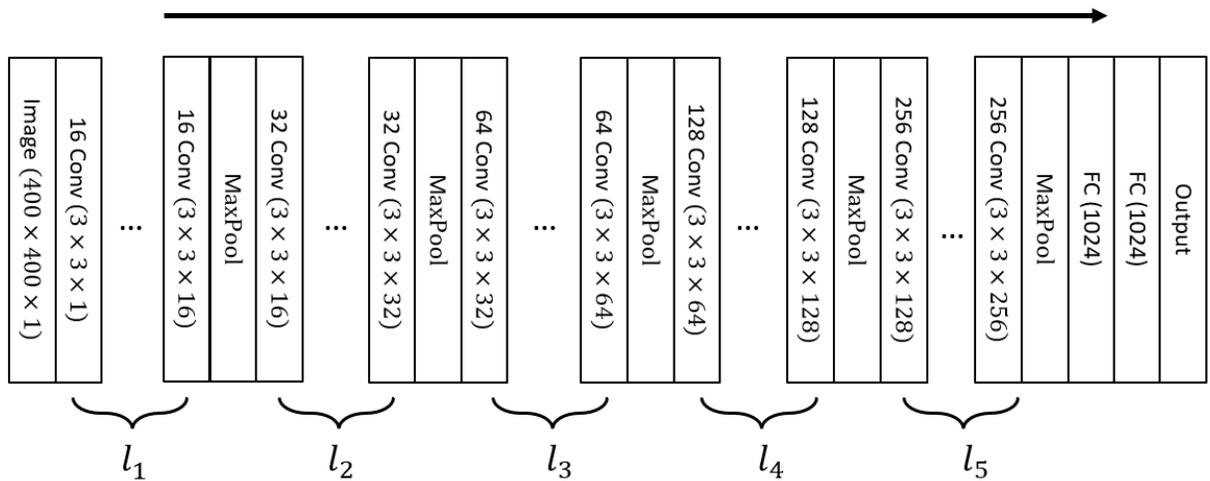
Fig. 6.5 Diagram of the convoulutional neural network model. There are 4

groups of convolutional layers (conv), 4 Max Pooling layers, and 2 fully connected

layers (FC). The number of conv layers in each group is l_1, l_2, l_3, l_4 and l_5.

In the first conv layers group, each convo layer contains 16 conv filters; second group

conv layer contains 32 conv filters; third group conv layer contains 64 conv filters; fourth group

conv layer contains 128 conv filters; fifth group conv layer contains 256 conv filters. The

numbers of conv layers in each group are hyperparameters which will decided after comparing

using cross-validation set. Padding size are all set as 2 and stride sizes are all 1. With this setup

and according to chapter 1, the conv layers will not change the size of the feature map. Max polling layers are put between every conv filter groups, such that the feature map will reduce the height and width by 2 after each group of conv layers. The number of elements of the two fully connected layers both set as 1024. The diagram of the neural network is shown in Fig. 6.5.

Training and Test

When applying the neural network mode to the image, an output $\tilde{y}$ could be used as the prediction of the distance from the focusing plane. To train the model efficiently, the images are divided into mini batches, each mini batch contains 64 images. The mean square error function [Eq. (6.1)] is used as loss function of the model.

$$L = \frac{1}{m}\sum_{i=1}^{m}(y_i - \tilde{y}_i)^2 \tag{6.1}$$

Where m is the total number of images in the batch and y is the real value of the distance from the focusing plane. From the loss calculated for a mini batch, the derivative of the loss over all parameters could be found, then all the trainable parameters θ could be updated to a new value [Eq. (6.2)].

$$\theta := \theta - \alpha \cdot \nabla_\theta J \tag{6.2}$$

α is the learning rate and the value is set to 0.0001. When all mini batches been used once, the training process has finished one epoch. The task of cross validation set is to figure the best combination of the hyper parameters. By monitoring the cross-validation loss, the model will be

saved when cross-validation loss improves. After 200 epochs training progress, result of best

cross-validation losses for all combinations are shown in Table. 6.1.

Model	l_1	l_1	l_1	l_1	l_1	Train loss	CV loss
a	1	1	1	1	1	1.10×10^{-3}	0.2197
b	2	1	1	1	1	1.25×10^{-3}	0.2171
c	2	2	1	1	1	2.91×10^{-3}	0.2083
d	2	2	2	1	1	8.20×10^{-4}	0.2265
e	2	2	2	2	1	1.58×10^{-3}	0.2316
f	2	2	2	2	2	2.16×10^{-3}	0.1610
g	1	2	2	2	2	7.27×10^{-3}	0.1933
h	1	1	2	2	2	5.47×10^{-4}	0.2032
i	1	1	1	2	2	4.66×10^{-3}	0.174
j	1	1	1	1	2	8.60×10^{-4}	0.2000

Table 6.1. The mean squared loss of tranning dataset and cross validation

dataset for different hyper parameters combination. The result is evaluated after

100 epochs trainning progress with learning rate $\alpha = 0.001$.

From Table 1, the combination of hyper parameters in model f is the most accurate one. It

is also the one which contains the most layer numbers. By applying the parameters learned from

the training process in this model, the loss of the test dataset is 0.2076. The average time cost for

predicting one image is 14 ms. Model a is the simplest model, only contains one layer in each

convo layer group, the test loss with model a is 0.2318. Depends on the trade-off between the accuracy and computing time, different models can be selected.

The 14 ms time consumption for predicting the out-of-focus distance is short enough for the camera in CLSM work with 60 fps. With this model, an autofocusing system, which has an accuracy in microns with a mean squared error near 0.2, could be developed.

CHAPTER SEVEN: CONCLUSION AND FUTURE WORKS

Chapter 4 shows that CLSM with a microlens array could reduce the scanning time. Chapter 5 shows the deep learning can apply to the PL spectrum and suggests the material's identity. In chapter 6, using the convolution neural networks could really predict the out-of-focus distance of the CLSM with a mean squared error near 0.2. The time consumption per prediction makes it possible to program the CLSM moving the objective during the scanning in real time.

The auto focusing algorithm gives the height information of the sample, which could also be used to produce the 3D map of the scanning region. Applying the algorithm in the CLSM with microlens array, each laser spot scanning its own region and collect the height information on the sample surface. By stitching method, a full 3D map can be formed.

Adding a PL spectrometer to the CLSM could makes the system collect the PL spectra during the scanning. Combined with the technique discussed in chapter 5, an image of sample surface can be formed with suggestions of material type at every point. The future work is to design a CLSM with PL spectrometer, that keep focusing the laser spot during the scan, analyze the PL spectrum from the sample scanning point, and predict the material's identity simultaneously.

Appendix

Python code for chapter 5

Background Functions

```
"""

Classification of Semiconductors by Photoluminescence Spectroscopy and Machine Learning

Yinchuan Yu

Washington State University

"""

import numpy as np

import math

import os

import matplotlib.pyplot as plt

import random


ROOT_DIR = os.path.dirname(os.path.abspath(__file__)) #define the root of folders


#some useful mathmatic functions
def sigmoid(x):

    s=1/(1+np.exp(-x))

    return s


def tanh(x):

    t=(np.exp(x)-np.exp(-x))/(np.exp(x)+np.exp(-x))

    return t


def relu(x):
```

```python
    r=np.maximum(0,x)
    return r


def softmax(x):
    expx=np.exp(x-np.max(x))
    return expx/expx.sum(axis=0, keepdims=True)


def bi_gaussian(x, x0, Cm,theta1, theta2):
    if x==x0:
        return Cm
    if x<x0:
        C=Cm*math.exp(-(x-x0)**2/(2*(theta1**2)))
    else:
        C=Cm*math.exp(-(x-x0)**2/(2*(theta2**2)))
    return C


class PL_generate():
    #The methods for generating a simulated Photoluminescent spectrum, including adding a peak, adding
a base line and adding noise
    def __init__(self, start, end):
        self.start=start
        self.end=end
        self.spectrum={}
        if end<start:
            print("impossible")
        for i in range(start,end):
            self.spectrum[i]=0


    def add_peak(self, peakcenter, height, leftwidth, rightwidth):
```

```python
        #adding peak using bi-gaussian method
        for wavelength in self.spectrum:
            self.spectrum[wavelength]+=bi_gaussian(wavelength, peakcenter, height, leftwidth, rightwidth)


    def add_noise(self, noise_type="normal",noise_level=0.01):
        #adding some gaussian noise
        if noise_type=="normal":
            for wavelength in self.spectrum:
                self.spectrum[wavelength]+=random.uniform(-noise_level, noise_level)
        else:
            for wavelength in self.spectrum:
                self.spectrum[wavelength]+=random.uniform(-noise_level,
noise_level)*self.spectrum[wavelength]


    def add_baseline(self, base):
        #adding base line to the PL
        for wavelength in self.spectrum:
            self.spectrum[wavelength]+=base


    def vectorize(self):
        #normalize the spectrum and store as an numpy vector
        vec=[]
        for i in range(self.start, self.end):
            vec.append(self.spectrum[i])
        vec=np.array(vec)
        vec=np.reshape(vec, ((self.end-self.start),1))
        normlized_vec=vec/(sum(np.square(vec)))**0.5
        return normlized_vec
```

```python
class Handmade:

    # The simulated photoluminscent spectra, including the method of made generate PL of
    "Ga2O3","ZnO","GaN","CdS","WS2","CsPbBr3".

    def __init__(self, classes_number=6, repeat_number=500, start=240, end= 700):

        self.X="init"

        self.Y=0

        self.pl={}

        substances=[]

        substances=["Ga2O3", "ZnO", "GaN","CdS","CsPbBr3","WS2"]

        for i, substance in enumerate(substances):

            for index in range(repeat_number):

                x = self.get_pl(substance, start, end)

                self.pl[substance+str(index)]=x

                y = [0]*i+[1]+[0]*(classes_number-1-i)

                y = np.array(y)

                y=np.reshape(y,(classes_number,1))

                if self.X=="init":

                    self.X=x

                    self.Y=y

                else:

                    self.X=np.hstack((self.X, x))

                    self.Y=np.hstack((self.Y, y))


    def get_pl(self, substance, start, end):

        #the method to get a new random generated pl spectrum for a certain substance

        PL=PL_generate(start, end)
```

```python
if substance == "Ga2O3":

    #Makeing the "Ga2O3" PL spectrum, with 360 nm, 400 nm, 430 nm, 520 nm peaks.

    peak1_center=random.gauss(360,3)

    peak1_height=random.uniform(0.01,0.5)

    peak1_left=random.uniform(30,50)

    peak1_right=random.uniform(30,50)


    peak2_center=random.gauss(400,5)

    peak2_height=random.uniform(0.01,0.5)

    peak2_left=random.uniform(40,60)

    peak2_right=random.uniform(40,60)


    peak3_center=random.gauss(430,10)

    peak3_height=random.uniform(0.01,0.5)

    peak3_left=random.uniform(50,100)

    peak3_right=random.uniform(50,100)


    peak4_center=random.gauss(520,10)

    peak4_height=random.uniform(0,0.1)

    peak4_left=random.uniform(25,100)

    peak4_right=random.uniform(25,100)


    PL.add_peak(peak1_center,peak1_height,peak1_left,peak1_right)

    PL.add_peak(peak2_center,peak2_height,peak2_left,peak2_right)

    PL.add_peak(peak3_center,peak3_height,peak3_left,peak3_right)

    PL.add_peak(peak4_center,peak4_height,peak4_left,peak4_right)

    PL.add_noise(noise_level=0.01)


if substance == "ZnO":
```

```python
##Makeing the "ZnO" PL spectrum, with 378 nm, 530 nm peaks.
peak1_center=random.gauss(378,3)
peak1_height=random.uniform(0.2,0.6)
peak1_left=random.uniform(5,10)
peak1_right=random.uniform(5,10)

peak2_center=random.gauss(530,20)
peak2_height=random.uniform(0.05,0.6)
peak2_left=random.uniform(25,50)
peak2_right=random.uniform(30,60)

PL.add_peak(peak1_center,peak1_height,peak1_left,peak1_right)
PL.add_peak(peak2_center,peak2_height,peak2_left,peak2_right)
PL.add_noise(noise_level=0.01)

if substance == "GaN":
    #Makeing the "Ga2O3" PL spectrum, with 365 nm, 382 nm, 430 nm, 525 nm peaks.
    peak1_center=random.gauss(365,3)
    peak1_height=random.uniform(0.3,0.8)
    peak1_left=random.uniform(5,10)
    peak1_right=random.uniform(5,10)

    peak2_center=random.gauss(382,3)
    peak2_height=random.uniform(0.05,0.1)
    peak2_left=random.uniform(6,15)
    peak2_right=random.uniform(6,15)

    peak3_center=random.gauss(525,10)
```

```python
        peak3_height=random.uniform(0.05,0.6)

        peak3_left=random.uniform(25,50)

        peak3_right=random.uniform(25,50)


        peak4_center=random.gauss(400,3)

        peak4_height=random.uniform(0.05,0.1)

        peak4_left=random.uniform(6,15)

        peak4_right=random.uniform(6,15)


        PL.add_peak(peak1_center,peak1_height,peak1_left,peak1_right)

        PL.add_peak(peak2_center,peak2_height,peak2_left,peak2_right)

        PL.add_peak(peak3_center,peak3_height,peak3_left,peak3_right)

        PL.add_peak(peak4_center,peak4_height,peak4_left,peak4_right)

        PL.add_noise(noise_level=0.01)


if substance =="CdS":

    #Makeing the "CdS" PL spectrum, with a 508 nm peak.

    peak1_center=random.gauss(508,3)

    peak1_height=random.uniform(0.4,1)

    peak1_left=random.uniform(5,10)

    peak1_right=random.uniform(5,10)


    PL.add_peak(peak1_center,peak1_height,peak1_left,peak1_right)

    PL.add_noise(noise_level=0.01)


if substance =="CsPbBr3":

    #Makeing the "CsPbBr3" PL spectrum, with a 523 nm peak.

    peak1_center=random.gauss(523,3)
```

```python
        peak1_height=random.uniform(0.4,1)

        peak1_left=random.uniform(8,9)

        peak1_right=random.uniform(8,9)


        PL.add_peak(peak1_center,peak1_height,peak1_left,peak1_right)

        PL.add_noise(noise_level=0.01)



    if substance =="WS2":

        #Makeing the "WS2" PL spectrum, with a 630 nm peak.

        peak1_center=random.gauss(630,10)

        peak1_height=random.uniform(0.4,1)

        peak1_left=random.uniform(10,25)

        peak1_right=random.uniform(15,30)


        PL.add_peak(peak1_center,peak1_height,peak1_left,peak1_right)

        PL.add_noise(noise_level=0.01)


    return PL.vectorize()


class Literature:

    #The object class used to read pl spectrum digitalized from literature by "GetData Graph", and
combined them into a dataset.

    def __init__(self, classes_number=6, start=240, end=700):

        self.X="init"

        self.Y="init"

        self.pl={}

        substances=["Ga2O3","ZnO","GaN","CdS","CsPbBr3","WS2"]

        k=0
```

```python
    for i, substance in enumerate(substances):
        index=1
        while True:
            try:
                readPATH = os.path.join(ROOT_DIR,"Raw_data", "Literature",substance)
                os.chdir(readPATH)
                open(str(index)+".txt","r")
            except:
                break

            x = self.get_pl(substance, index, start, end)
            self.pl[substance+str(index)]=x
            y = [0]*i+[1]+[0]*(classes_number-1-i)
            y = np.array(y)
            y=np.reshape(y,(classes_number,1))

            if self.X=="init":
                self.X=x
                self.Y=y
            else:
                self.X=np.hstack((self.X, x))
                self.Y=np.hstack((self.Y, y))
            index+=1
            k+=1

def get_pl(self, substance, index, start, end):
    raw=[]
    readPATH = os.path.join(ROOT_DIR,"Raw_data", "Literature",substance)
    os.chdir(readPATH)
```

```python
File=open(str(index)+".txt","r")

pairs=File.readlines()

checkpoint=pairs[0].split()

if float(checkpoint[0])>100:

    for pair in pairs:

        pair=pair.split()

        raw.append([float(pair[0]),float(pair[1])])

    raw=sorted(raw, key=lambda x: x[0])

else:

    for pair in pairs:

        pair=pair.split()

        raw.append([1242.18/float(pair[0]),float(pair[1])])

    raw=sorted(raw, key=lambda x: x[0])


spectrum=[0]*(end-start)

wavelength=start

i=0

while wavelength<end:

    if wavelength<raw[0][0]:

        wavelength+=1

        continue

    if wavelength>raw[-1][0]:

        break

    while i<=len(raw) and raw[i][0]<wavelength:

        i+=1

    if wavelength==raw[i][0]:

        spectrum[wavelength-start]=(raw[i][1])

    else:

        V=(raw[i][1]-raw[i-1][1])*(wavelength-raw[i-1][0])/(raw[i][0]-raw[i-1][0])+raw[i-1][1]
```

```python
            spectrum[wavelength-start]=V
        wavelength+=1


    spectrum = np.array(spectrum)
    normlized_spectrum=spectrum/(sum(np.square(spectrum)))**0.5
    normlized_spectrum=np.reshape(normlized_spectrum,(end-start,1))
    return normlized_spectrum



class SingleHorriba:
    # Make a unknown single PL spectrum into a vector, can use algrithm to predict its category.
    def __init__(self, class_name, index, start=240, end=700 ):
        self.X = self.get_pl(class_name, index, start, end)


    def get_pl(self, substance, index, start, end):
        PATH=os.path.join(ROOT_DIR,"Raw_data", "Single_Horriba", substance)
        os.chdir(PATH)
        spectrum=[0]*(end-start)
        filename=index+".txt"
        d=np.loadtxt(filename)


        for i in range(len(spectrum)):
            if i>=len(d) or d[i][0]>=end:
                break
            if d[i][0]<start:
                continue


            spectrum[int(d[i][0]-start)]=d[i][1]
```

```python
        spectrum=np.array(spectrum)
        spectrum=spectrum/sum((np.square(spectrum)))**0.5
        spectrum=np.reshape(spectrum,(start-end,1))
        return spectrum


class Parameter_SL:
    #save and Load the parameters trained by the algrithm into a txt file
    def save(self, parameters, version):
        PATH = os.path.join(ROOT_DIR,"Parameters",version)
        os.mkdir(PATH)
        os.chdir(PATH)
        W1 = parameters["W1"]
        b1 = parameters["b1"]
        W2 = parameters["W2"]
        b2 = parameters["b2"]
        W3 = parameters["W3"]
        b3 = parameters["b3"]


        np.savetxt('W1.txt', W1)
        np.savetxt('W2.txt', W2)
        np.savetxt('W3.txt', W3)
        np.savetxt('b1.txt', b1)
        np.savetxt('b2.txt', b2)
        np.savetxt('b3.txt', b3)



    def load(self, version):
        PATH = os.path.join(ROOT_DIR,"Parameters",version)
        os.chdir(PATH)
```

```python
        parameters={}
        parameters["W1"] = np.loadtxt('W1.txt', dtype=float)
        parameters["W2"] = np.loadtxt('W2.txt', dtype=float)
        parameters["W3"] = np.loadtxt('W3.txt', dtype=float)
        parameters["b1"] = np.loadtxt('b1.txt', dtype=float).reshape(-1,1)
        parameters["b2"] = np.loadtxt('b2.txt', dtype=float).reshape(-1,1)
        parameters["b3"] = np.loadtxt('b3.txt', dtype=float).reshape(-1,1)


        return parameters



class Learn:
    def initialize_with_zero(self, layer_dims):
        #initialize the parameters all Zeros
        parameters={}
        L=len(layer_dims)


        for l in range(1,L):
            parameters["W"+str(l)]=np.zeros((layer_dims[l],layer_dims[l-1]))
            parameters["b"+str(l)]=np.zeros((layer_dims[l],1))


        return parameters


    def initialize_with_random(self,layer_dims):
        #initialize the parameters with random values
        parameters={}
        L=len(layer_dims)


        for l in range(1,L):
```

```python
        parameters["W"+str(l)]=0.01 * np.random.randn(layer_dims[l],layer_dims[l-1])

        parameters["b"+str(l)]=np.zeros((layer_dims[l],1))


    return parameters


    def initialize_with_he(self, layer_dims):
        #initialize the parameters with HE method
        parameters={}
        L=len(layer_dims)


        for l in range(1,L):
            parameters["W"+str(l)]= np.random.randn(layer_dims[l],layer_dims[l-1]) *
np.sqrt(2/layer_dims[l-1])
            parameters["b"+str(l)]=np.zeros((layer_dims[l],1))
            print("W"+str(l)+"shape: " , parameters["W"+str(l)].shape)
        return parameters


    def forward_propagation(self, X, parameters):
        #The forward propagation, from the input PL vector to a softmax categorical output
        W1 = parameters["W1"]
        b1 = parameters["b1"]
        W2 = parameters["W2"]
        b2 = parameters["b2"]
        W3 = parameters["W3"]
        b3 = parameters["b3"]


        #First Layer, activated by ReLU
        Z1=np.dot(W1, X) + b1
        A1=relu(Z1)
```

```python
        #Second Layer, activated by ReLU
        Z2=np.dot(W2, A1) + b2
        A2=relu(Z2)
        #Output layer, activated by softmax
        Z3=np.dot(W3,A2) + b3
        A3=softmax(Z3)


        cache = (Z1, A1, W1, b1, Z2, A2, W2, b2, Z3, A3, W3, b3)
        return A3, cache


    def backward_propagation(self, X, Y, cache):
        #if no regulization, this function will be used
        #Backward propagation to calculate the gradient of all parameters
        m=X.shape[1]
        (Z1, A1, W1, b1, Z2, A2, W2, b2, Z3, A3, W3, b3)=cache


        dZ3=A3-Y
        dW3=1./m*np.dot(dZ3,A2.T)
        db3=1./m*np.sum(dZ3, axis=1,keepdims=True)


        dA2 = np.dot(W3.T, dZ3)
        dZ2 = np.multiply(dA2, np.int64(A2 > 0))
        dW2 = 1./m * np.dot(dZ2, A1.T)
        db2 = 1./m * np.sum(dZ2, axis=1, keepdims = True)


        dA1 = np.dot(W2.T, dZ2)
        dZ1 = np.multiply(dA1, np.int64(A1 > 0))
        dW1 = 1./m * np.dot(dZ1, X.T)
        db1 = 1./m * np.sum(dZ1, axis=1, keepdims = True)
```

```python
        gradients = {"dZ3": dZ3, "dW3": dW3, "db3": db3,
                "dA2": dA2, "dZ2": dZ2, "dW2": dW2, "db2": db2,
                "dA1": dA1, "dZ1": dZ1, "dW1": dW1, "db1": db1}


        return gradients


    def backward_propagation_L2(self, X, Y, cache, lamb):
        #if there are regulization term used, this function will be used
        #Backward propagation to calculate the gradient of all parameters
        m=X.shape[1]
        (Z1, A1, W1, b1, Z2, A2, W2, b2, Z3, A3, W3, b3)=cache


        dZ3=A3-Y
        dW3=1./m*(np.dot(dZ3,A2.T)+lamb*W3)
        db3=1./m*np.sum(dZ3, axis=1,keepdims=True)


        dA2 = np.dot(W3.T, dZ3)
        dZ2 = np.multiply(dA2, np.int64(A2 > 0))
        dW2 = 1./m * (np.dot(dZ2, A1.T)+lamb*W2)
        db2 = 1./m * np.sum(dZ2, axis=1, keepdims = True)


        dA1 = np.dot(W2.T, dZ2)
        dZ1 = np.multiply(dA1, np.int64(A1 > 0))
        dW1 = 1./m * (np.dot(dZ1, X.T)+lamb*W1)
        db1 = 1./m * np.sum(dZ1, axis=1, keepdims = True)


        gradients = {"dZ3": dZ3, "dW3": dW3, "db3": db3,
                "dA2": dA2, "dZ2": dZ2, "dW2": dW2, "db2": db2,
```

```python
                    "dA1": dA1, "dZ1": dZ1, "dW1": dW1, "db1": db1}


    return gradients


def update_parameters(self, parameters,grads,learning_rate):
    #update the parameters from the gradient getting from Backward propagation
    for i in range(len(parameters)//2):
        parameters["W"+str(i+1)]=parameters["W"+str(i+1)]-learning_rate * grads["dW"+str(i+1)]
        parameters["b"+str(i+1)]=parameters["b"+str(i+1)]-learning_rate * grads["db"+str(i+1)]
    return parameters


def predict(self, X, Y, parameters):
    #predict all the testing samples
    m = X.shape[1]
    a3, caches = self.forward_propagation(X, parameters)
    p= np.argmax(a3, axis=0)
    Y = np.argmax(Y, axis=0)
    for i in range(len(p)):
        if p[i]!=Y[i]:
            print(i," Substance ",Y[i]," prediction ",p[i])
    print("Accuracy: "  + str((p == Y).mean()) )
    return p


def predict_single(self,X,parameters):
    #predict a single vector, will show the probability in all categories
    a3, caches = self.forward_propagation(X, parameters)
    print("Ga2O3 :   ", a3[0])
    print("ZnO :     ", a3[1])
    print("GaN :     ", a3[2])
```

```python
        print("CdS :      ", a3[3])
        print("CsPbBr3 : ", a3[4])
        print("WS2 :      ", a3[5])




    def compute_cost(self, a3, Y):
        #calculate the cost, used to display in each training iteration
        cost = -np.mean(Y * np.log(a3+1e-8))
        return cost


    def compute_cost_with_regularization(self, A3, Y, parameters, lambd):
        #calculate the cost with regularization terms
        m = Y.shape[1]
        W1 = parameters["W1"]
        W2 = parameters["W2"]
        W3 = parameters["W3"]


        cross_entropy_cost = self.compute_cost(A3, Y)
        L2_regularization_cost = (1. / m)*(lambd / 2) * (np.sum(np.square(W1)) + np.sum(np.square(W2))
+ np.sum(np.square(W3)))
        cost = cross_entropy_cost + L2_regularization_cost
        return cost




    def model(self, X, Y, learning_rate = 0.01, num_iterations = 30000, print_cost = True, lambd = 0):
        #The training model with input X, 50 elements hidden layer 1, 20 elements hidden layer 2, and the
output
        grads = {}
        costs = []
```

```python
        m = X.shape[1]    #find the amount number of training samples

        layer_dims = [X.shape[0], 50, 20, 6]   # The dimension of each layer, [the input vector dim, 50
elements, 20 elements, 6 classes]

        parameters = self.initialize_with_he(layer_dims) #Initialize the trainning parameters with HE
method

        for i in range(0, num_iterations):

            #trainning with numbers of iterations set

            a3, cache = self.forward_propagation(X, parameters)

            if lambd == 0: #if not regulization term

                cost = self.compute_cost(a3, Y)

            else: # with regulization

                cost = self.compute_cost_with_regularization(a3, Y, parameters, lambd)

            if lambd == 0:

                grads = self.backward_propagation(X, Y, cache)

            else:

                grads = self.backward_propagation_L2(X, Y, cache, lambd)

            parameters = self.update_parameters(parameters, grads, learning_rate)

            if print_cost and i % 100 == 0:

                print("Cost after iteration {}: {}".format(i, cost))

            if print_cost and i % 100 == 0:

                costs.append(cost)


        #plot the cost during train, cost vs. iterations

        plt.plot(costs)

        plt.ylabel('cost')

        plt.xlabel('iterations (x100)')

        plt.title("Learning rate =" + str(learning_rate))

        plt.show()

        return parameters
```

Training

"""

Classification of Semiconductors by Photoluminescence Spectroscopy and Machine Learning

Yinchuan Yu

Washington State University

"""


```python
from PLcode import *


# Set the training set as the simulated PL, each substances generate 1000 spectra
Train=Handmade(6,10)
# first test sets as the PL from literature
Test=Literature(6)
learn=Learn()


#change the version number for yourself
previous_trained_version = 1
new_trained_version = 2


#if you have pre-trained parameters, uncommon next line
#parameters=sl.load("Version"+ str(previous_trained_version))


#training with 10000 iterations and learning rate as 0.01
parameters=learn.model(Train.X, Train.Y, learning_rate = 0.01, num_iterations = 20000, print_cost = True, lambd = 0.1)
sl=Parameter_SL()
sl.save(parameters,"Version"+str(new_trained_version))


print("train")
```

```python
learn.predict(Train.X, Train.Y, parameters)
print("test")
learn.predict(Test.X, Test.Y, parameters)


Prediction
"""
Classification of Semiconductors by Photoluminescence Spectroscopy and Machine Learning
Yinchuan Yu
Washington State University
"""


import matplotlib.pyplot as plt
import os
import numpy as np
from PLcode import *


learn=Learn()
sl=Parameter_SL()
parameters=sl.load("Version1")


substance, index="test", "1" #the subfolder in Single_Horriba, and the spectrum file name in txt format
Test=SingleHorriba(substance,index).X
Axis=[]
for i in range(410):
    Axis.append(i+240)
plt.plot(Axis,Test[0:410,0])
plt.ylabel('Intensity')
plt.xlabel('wavelength (nm)')
plt.show()
```

```python
print("Test of ", substance)

learn.predict_single(Test, parameters)
```

Python code for chapter 6

Learning Model

```python
"""

Autofocusing of a Confocal Laser Microscope using Convolutional Neural Networks

Yinchuan Yu

Washington State University

"""


import tensorflow as tf

from tensorflow.keras import datasets, layers, models, activations, regularizers


def my_model(pix_len, regularizer, layerNumbers):

    inputs = tf.keras.Input(shape=(pix_len, pix_len, 1))

    x = inputs

    #first convo layer group

    for _ in range(layerNumbers[0]):

        x = layers.Conv2D(16, 3, padding='same', kernel_regularizer=regularizers.l2(regularizer))(x)

        x = activations.relu(x)

    #max pooling

    x = layers.MaxPooling2D()(x)


    #second convo layer group

    for _ in range(layerNumbers[1]):

        x = layers.Conv2D(32, 3, padding='same', kernel_regularizer=regularizers.l2(regularizer))(x)
```

```python
    x = activations.relu(x)
#max pooling
x = layers.MaxPooling2D()(x)


# third convo layer group
for _ in range(layerNumbers[2]):
    x = layers.Conv2D(64, 3, padding='same', kernel_regularizer=regularizers.l2(regularizer))(x)
    x = activations.relu(x)
# max pooling
x = layers.MaxPooling2D()(x)


# forth convo layer group
for _ in range(layerNumbers[3]):
    x = layers.Conv2D(128, 3, padding='same', kernel_regularizer=regularizers.l2(regularizer))(x)
    x = activations.relu(x)
# max pooling
x = layers.MaxPooling2D()(x)


# fifth convo layer group
for _ in range(layerNumbers[4]):
    x = layers.Conv2D(256, 3, padding='same', kernel_regularizer=regularizers.l2(regularizer))(x)
    x = activations.relu(x)
# max pooling
x = layers.MaxPooling2D()(x)  # 12


#Fully connected layers
x = layers.Flatten()(x)
x = layers.Dense(1024, activation='relu', kernel_regularizer=regularizers.l2(regularizer))(x)
x = layers.Dense(1024, activation='relu', kernel_regularizer=regularizers.l2(regularizer))(x)
```

```python
    outputs = layers.Dense(1)(x)

    model = tf.keras.Model(inputs=inputs, outputs=outputs)

    return model
```

Training

```python
"""

Autofocusing of a Confocal Laser Microscope using Convolutional Neural Networks

Yinchuan Yu

Washington State University

"""

import tensorflow as tf

from root import *

from model import *




#loading the training set

train_path= os.path.join(ROOT_DIR, "DataSet","small_random","train.npz")

with np.load(train_path) as data:

    train_images = data['images']

    train_labels = data['labels']


#loading the cross-validation set

cv_path = os.path.join(ROOT_DIR, "DataSet","small_random","cross_val.npz")

with np.load(cv_path) as data:

    cv_images = data['images']

    cv_labels = data['labels']


train_images = train_images / 255.0
```

```python
cv_images = cv_images / 255.0


os.chdir(ROOT_DIR)


#define the number of layers in each convo group
layerNumbers=[2,2,2,2,2]
folderName="model"
for num in layerNumbers:
    folderName+="_"+str(num)
folderName+="/"
print(folderName)


save_path= "saved_model/sythesis/"+folderName
checkpoint_path = save_path
checkpoint_dir = os.path.dirname(checkpoint_path)


# Create a callback that saves the model's weights
cp_callback = tf.keras.callbacks.ModelCheckpoint(filepath=checkpoint_path,
                            monitor= 'val_loss',
                            save_weights_only=True,
                            save_best_only=True,
                            verbose=1)


model = my_model(400, 0, layerNumbers)
model.summary()
model.compile(optimizer=tf.optimizers.Adam(learning_rate=0.0001), loss='mean_squared_error')
#train the model
history = model.fit(x=train_images, y=train_labels,
            batch_size=64,
```

```
            shuffle=True,

            epochs=200,

            verbose=1,

            validation_data= (cv_images, cv_labels),

            callbacks=[cp_callback])




loss = np.array(history.history["loss"])

val_loss = np.array(history.history["val_loss"])


np.savetxt(save_path+"/loss.txt", loss)

np.savetxt(save_path+"/val_loss.txt", val_loss)
```

Testing

```
"""
Autofocusing of a Confocal Laser Microscope using Convolutional Neural Networks

Yinchuan Yu

Washington State University
"""


import tensorflow as tf

from root import *

from model import *

import time


#loading testing set

test_path = os.path.join(ROOT_DIR, "DataSet","small_random","test.npz")

with np.load(test_path) as data:
```

```python
    test_images = data['images']
    test_labels = data['labels']


test_images = test_images / 255.0


os.chdir(ROOT_DIR)


#define the model used for testing
layerNumbers=[2,2,2,2,1]
folderName="model"
for num in layerNumbers:
    folderName+="_"+str(num)
folderName+="/"
print(folderName)


model = my_model(400, 0, layerNumbers)
model.compile(optimizer=tf.optimizers.Adam(learning_rate=0.0001), loss='mean_squared_error')
model.load_weights("saved_model/sythesis/"+folderName)


test_loss = model.evaluate(test_images, test_labels, batch_size=64)
print('\nTest loss:', test_loss)
```

Bibliography

1. Nielsen, M. A. Neural Networks and Deep Learning. (2015).

2. McCulloch, W. S. & Pitts, W. A logical calculus of the ideas immanent in nervous activity. *Bull. Math. Biophys.* **5**, 115–133 (1943).

3. Hochreiter, S. The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions. *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.* **06**, 107–116 (1998).

4. Agarap, A. F. Deep Learning using Rectified Linear Units (ReLU). *ArXiv180308375 Cs Stat* (2019).

5. Rumelhart, D. E., Hinton, G. E. & Williams, R. J. Learning representations by back-propagating errors. *Nature* **323**, 533–536 (1986).

6. Glorot, X. & Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics* 249–256 (JMLR Workshop and Conference Proceedings, 2010).

7. He, K., Zhang, X., Ren, S. & Sun, J. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. *ArXiv150201852 Cs* (2015).

8. URL: https://commons.wikimedia.org/wiki/File:Saddle_point.svg.

9. Kingma, D. P. & Ba, J. Adam: A Method for Stochastic Optimization. *ArXiv14126980 Cs* (2017).

10. Berrar, D. Cross-Validation. in (2018). doi:10.1016/B978-0-12-809633-8.20349-X.

11. Ying, X. An Overview of Overfitting and its Solutions. *J. Phys. Conf. Ser.* **1168**, 022022 (2019).

12. Warde-Farley, D., Goodfellow, I. J., Courville, A. & Bengio, Y. An empirical analysis of dropout in piecewise linear networks. (2013).

13. Russakovsky, O. *et al.* ImageNet Large Scale Visual Recognition Challenge. *ArXiv14090575 Cs* (2015).

14. Deng, J. *et al.* ImageNet: A large-scale hierarchical image database. in *2009 IEEE Conference on Computer Vision and Pattern Recognition* 248–255 (2009). doi:10.1109/CVPR.2009.5206848.

15. Krizhevsky, A., Sutskever, I. & Hinton, G. E. ImageNet Classification with Deep Convolutional Neural Networks. in *Advances in Neural Information Processing Systems* vol. 25 (Curran Associates, Inc., 2012).

16. Simonyan, K. & Zisserman, A. Very Deep Convolutional Networks for Large-Scale Image Recognition. *ArXiv14091556 Cs* (2015).

17. Kronig, R. D. L., Penney, W. G. & Fowler, R. H. Quantum mechanics of electrons in crystal lattices. *Proc. R. Soc. Lond. Ser. Contain. Pap. Math. Phys. Character* **130**, 499–513 (1931).

18. McCluskey, M. D. & Haller, E. E. *Dopants and Defects in Semiconductors*. (CRC Press, 2018).

19. Stavrou, V. N. *Phonons in Low Dimensional Structures*. (BoD – Books on Demand, 2018).

20. Bhat, I. 3 - Physical properties of gallium nitride and related III–V nitrides. in *Wide Bandgap Semiconductor Power Devices* (ed. Baliga, B. J.) 43–77 (Woodhead Publishing, 2019). doi:10.1016/B978-0-08-102306-8.00003-4.

21. Frenkel, J. On the Transformation of light into Heat in Solids. I. *Phys. Rev.* **37**, 17–44 (1931).

22. Wannier, G. H. The Structure of Electronic Excitation Levels in Insulating Crystals. *Phys. Rev.* **52**, 191–197 (1937).

23. Fuller, C. S. & Ditzenberger, J. A. Diffusion of Donor and Acceptor Elements in Silicon. *J. Appl. Phys.* **27**, 544–553 (1956).

24. Smith, K. K. Photoluminescence of semiconductor materials. *Thin Solid Films* **84**, 171–182 (1981).

25. Gilliland, G. D. Photoluminescence spectroscopy of crystalline semiconductors. *Mater. Sci. Eng. R Rep.* **18**, 99–399 (1997).

26. URL: https://static.horiba.com/fileadmin/Horiba/Products/Scientific/Molecular_and_Microanalysis/Fluorolog/Fluorolog_Brochure.pdf.

27. Langenbucher, A. Law of Refraction (Snell's Law). in *Encyclopedia of Ophthalmology* (eds. Schmidt-Erfurth, U. & Kohnen, T.) 1042–1042 (Springer, 2018). doi:10.1007/978-3-540-69000-9_637.

28. Pedrotti, F. L., Pedrotti, L. M. & Pedrotti, L. S. Introduction to Optics. *Higher Education from Cambridge University Press* https://www.cambridge.org/highereducation/books/introduction-to-optics/8923BB7830091A58316177E029C5D420 (2017) doi:10.1017/9781108552493.

29. Yariv, A., Yariv, P. of E. E. A. & Amnon, Y. *Quantum Electronics*. (Wiley, 1989).

30. Self, S. A. Focusing of spherical Gaussian beams. *Appl. Opt.* **22**, 658–661 (1983).

31. Bandres, M. A. & Gutiérrez-Vega, J. C. Ince–Gaussian beams. *Opt. Lett.* **29**, 144–146 (2004).

32. Elements of Modern Optical Design | Wiley. *Wiley.com* https://www.wiley.com/en-us/Elements+of+Modern+Optical+Design-p-9780471077961.

33. McNamara, G., Difilippantonio, M. J. & Ried, T. Microscopy and Image Analysis. *Curr. Protoc. Hum. Genet. Editor. Board Jonathan Haines Al* **0 4**, Unit-4.4 (2005).

34. Minsky, M. Memoir on inventing the confocal scanning microscope. *Scanning* **10**, 128–138 (1988).

35. HOVIS, D. B. The use of laser scanning confocal microscopy (LSCM) in materials science. *J. Microsc.* **240**, 173–180 (2010).

36. White, J. G., Amos, W. B. & Fordham, M. An evaluation of confocal versus conventional imaging of biological structures by fluorescence light microscopy. *J. Cell Biol.* **105**, 41–48 (1987).

37. Helmchen, F. & Denk, W. Deep tissue two-photon microscopy. *Nat. Methods* **2**, 932–940 (2005).

38. Gruber, A. *et al.* Scanning Confocal Optical Microscopy and Magnetic Resonance on Single Defect Centers. *Science* **276**, 2012–2014 (1997).

39. Ye, X. & McCluskey, M. D. Modular Scanning Confocal Microscope with Digital Image Processing. *PLOS ONE* **11**, e0166212 (2016).

40. Hu, M.-K. Visual pattern recognition by moment invariants. *IRE Trans. Inf. Theory* **8**, 179–187 (1962).

41. Yu, Y., Ye, X., Ye, X. & McCluskey, M. D. Confocal microscopy with a microlens array. *Appl. Opt.* **59**, 3058–3063 (2020).

42. Denk, W., Strickler, J. H. & Webb, W. W. Two-Photon Laser Scanning Fluorescence Microscopy. *Science* **248**, 73–76 (1990).

43. Novotny, L. & Hecht, B. *Principles of Nano-Optics*. (Cambridge University Press, 2012).

44. Stephens, D. J. & Allan, V. J. Light Microscopy Techniques for Live Cell Imaging. *Science* **300**, 82–86 (2003).

45. Hickey, P. C., Swift, S. R., Roca, M. G. & Read, N. D. Live-cell Imaging of Filamentous Fungi Using Vital Fluorescent Dyes and Confocal Microscopy. in *Methods in Microbiology* vol. 34 63–87 (Academic Press, 2004).

46. Phan, T. G. & Bullen, A. Practical intravital two-photon microscopy for immunological research: faster, brighter, deeper. *Immunol. Cell Biol.* **88**, 438–444 (2010).

47. Ettinger, A. & Wittmann, T. Chapter 5 - Fluorescence live cell imaging. in *Methods in Cell Biology* (eds. Waters, J. C. & Wittman, T.) vol. 123 77–94 (Academic Press, 2014).

48. Hoheisel, W., Jacobsen, W., Lüttge, B. & Weiner, W. Confocal Microscopy: Applications in Materials Science. *Macromol. Mater. Eng.* **286**, 663–668 (2001).

49. M. Chizhik, A. *et al.* Optical imaging of excited-state tautomerization in single molecules. *Phys. Chem. Chem. Phys.* **13**, 1722–1733 (2011).

50. Barnard, E. S. *et al.* Probing carrier lifetimes in photovoltaic materials using subsurface two-photon microscopy. *Sci. Rep.* **3**, 2098 (2013).

51. Sadeghi, S., Khabbaz Abkenar, S., Ow-Yang, C. W. & Nizamoglu, S. Efficient White LEDs Using Liquid-state Magic-sized CdSe Quantum Dots. *Sci. Rep.* **9**, 10061 (2019).

52. Sick, B., Hecht, B. & Novotny, L. Orientational Imaging of Single Molecules by Annular Illumination. *Phys. Rev. Lett.* **85**, 4482–4485 (2000).

53. Lounis, B. & Moerner, W. E. Single photons on demand from a single molecule at room temperature. *Nature* **407**, 491–493 (2000).

54. Débarre, A. *et al.* Quantitative determination of the 3D dipole orientation of single molecules. *Eur. Phys. J. - At. Mol. Opt. Plasma Phys.* **28**, 67–77 (2004).

55. Boichenko, S. Theoretical investigation of confocal microscopy using an elliptically polarized cylindrical vector laser beam: Visualization of quantum emitters near interfaces. *Phys. Rev. A* **97**, 043825 (2018).

56. So, P. T. C., Dong, C. Y., Masters, B. R. & Berland, K. M. Two-Photon Excitation Fluorescence Microscopy. *Annu. Rev. Biomed. Eng.* **2**, 399–429 (2000).

57. Pawley, J. *Handbook of Biological Confocal Microscopy*. (Springer Science & Business Media, 2013).

58. McCluskey, M. D. Digital confocal optical profile microscopy. US Patent 9,891,422 (2018).

59. Wang, Q. *et al.* Phase-Defined van der Waals Schottky Junctions with Significantly Enhanced Thermoelectric Properties. *J. Phys. Chem. Lett.* **8**, 2887–2894 (2017).

60. Sánchez-Ortiga, E. *et al.* Subtractive imaging in confocal scanning microscopy using a CCD camera as a detector. *Opt. Lett.* **37**, 1280–1282 (2012).

61. Ye, X. & Mccluskey, M. Subtractive Imaging Using Gaussian Fits and Image Moments in Confocal Microscopy. **1**, (2017).

62. Artzner, G. E. Microlens arrays for Shack-Hartmann wavefront sensors. *Opt. Eng.* **31**, 1311–1322 (1992).

63. Levoy, M., Ng, R., Adams, A., Footer, M. & Horowitz, M. Light field microscopy. *ACM Trans. Graph.* **25**, 924–934 (2006).

64. Bewersdorf, J., Pick, R. & Hell, S. W. Multifocal multiphoton microscopy. *Opt. Lett.* **23**, 655–657 (1998).

65. Minamikawa, T., Hashimoto, M., Fujita, K., Kawata, S. & Araki, T. Multi-focus excitation coherent anti-Stokes Raman scattering (CARS) microscopy and its applications for real-time imaging. *Opt. Express* **17**, 9526–9536 (2009).

66. Tiziani, H. J. & Uhde, H.-M. Three-dimensional analysis by a microlens-array confocal arrangement. *Appl. Opt.* **33**, 567–572 (1994).

67. Orth, A. & Crozier, K. Microscopy with microlens arrays: high throughput, high resolution and light-field imaging. *Opt. Express* **20**, 13522–13531 (2012).

68. George, J. S. Scanning computed confocal imager. US Patent 6,038,067 (2000).

69. Toomre, D. & Pawley, J. B. Disk-Scanning Confocal Microscopy. in *Handbook Of Biological Confocal Microscopy* (ed. Pawley, J. B.) 221–238 (Springer US, 2006). doi:10.1007/978-0-387-45524-2_10.

70. Favro, L. D., Thomas, R. L., Kuo, P.-K. & Chen, L. Confocal microscope. US Patent 5,162,941 (1992).

71. Isozaki, K. & Kenta Mikuriya, Y. Confocal Microscope. European Patent Application 1245986 (2000).

72. Hell, S. & Pick, R. Confocal Microscope Comprising Two Microlens Arrays and a Pinhole Diaphragm. US Patent Application 20050094261 (2003).

73. URL: https://readyoptics.com/index_files/Page848.htm.

74. Bevington, P. R., Robinson, D. K., Blair, J. M., Mallinckrodt, A. J. & McKay, S. Data Reduction and Error Analysis for the Physical Sciences. *Comput. Phys.* **7**, 415–416 (1993).

75. Yu, Y. & McCluskey, M. D. Classification of Semiconductors Using Photoluminescence Spectroscopy and Machine Learning. *Appl. Spectrosc.* 00037028211031618 (2021) doi:10.1177/00037028211031618.

76. Burgess, C. OPTICAL SPECTROSCOPY | Radiation Sources. in *Encyclopedia of Analytical Science (Third Edition)* (eds. Worsfold, P., Poole, C., Townshend, A. & Miró, M.) 103–109 (Academic Press, 2005). doi:10.1016/B978-0-08-101983-2.00429-1.

77. Vandenabeele, P., Edwards, H. G. M. & Moens, L. A Decade of Raman Spectroscopy in Art and Archaeology. *Chem. Rev.* **107**, 675–686 (2007).

78. Beattie, J. R. & Esmonde-White, F. W. L. Exploration of Principal Component Analysis: Deriving Principal Component Analysis Visually Using Spectra. *Appl. Spectrosc.* **75**, 361–375 (2021).

79. Kotsiantis, S. B., Zaharakis, I. D. & Pintelas, P. E. Machine learning: a review of classification and combining techniques. *Artif. Intell. Rev.* **26**, 159–190 (2006).

80. Hornik, K. Approximation capabilities of multilayer feedforward networks. *Neural Netw.* **4**, 251–257 (1991).

81. Yarotsky, D. Error bounds for approximations with deep ReLU networks. *Neural Netw.* **94**, 103–114 (2017).

82. McCluskey, M. D. Point defects in Ga2O3. *J. Appl. Phys.* **127**, 101101 (2020).

83. Zhang, R., Yin, P.-G., Wang, N. & Guo, L. Photoluminescence and Raman scattering of ZnO nanorods. *Solid State Sci.* **11**, 865–869 (2009).

84. Freitas, J. A., Tischler, J. G., Garces, N. Y. & Feigelson, B. N. Optical probing of low-pressure solution grown GaN crystal properties. *J. Cryst. Growth* **312**, 2564–2568 (2010).

85. Reshchikov, M. A., McNamara, J. D., Helava, H., Usikov, A. & Makarov, Y. Two yellow luminescence bands in undoped GaN. *Sci. Rep.* **8**, 8091 (2018).

86. Hoang, T. B. *et al.* Temperature dependent photoluminescence of single CdS nanowires. *Appl. Phys. Lett.* **89**, 123123 (2006).

87. Gutiérrez, H. R. *et al.* Extraordinary Room-Temperature Photoluminescence in Triangular WS2 Monolayers. *Nano Lett.* **13**, 3447–3454 (2013).

88. Pourdavoud, N. *et al.* Room-Temperature Stimulated Emission and Lasing in Recrystallized Cesium Lead Bromide Perovskite Thin Films. *Adv. Mater.* **31**, 1903717 (2019).

89. Ruder, S. An overview of gradient descent optimization algorithms. *ArXiv160904747 Cs* (2017).

90. Digitize graphs and plots - GetData Graph Digitizer - graph digitizing software. http://getdata-graph-digitizer.com/index.php.

91. Liu, H., Xu, C., Pan, X. & Ye, Z. The Photoluminescence Properties of β-Ga2O3 Thin Films. *J. Electron. Mater.* **49**, (2020).

92. Kumar, S. Nanofunctional gallium oxide (Ga2O3) nanowires/nanostructures and their applications in nanodevices. *Phys. Status Solidi RRL - Rapid Res. Lett.* (2013) doi:10.1002/pssr.201307253.

93. Hou, Y. *et al.* Photocatalytic performance of α-, β-, and γ-Ga2O3 for the destruction of volatile aromatic pollutants in air. *J. Catal.* **250**, 12–18 (2007).

94. Lyu, S. C. *et al.* Low temperature growth and photoluminescence of well-aligned zinc oxide nanowires. *Chem. Phys. Lett.* **363**, 134–138 (2002).

95. Zhang, X. T. *et al.* Structure and photoluminescence of Mn-passivated nanocrystalline ZnO thin films. *J. Cryst. Growth* **254**, 80–85 (2003).

96. Balkaş, C. M. *et al.* Growth and characterization of GaN single crystals. *J. Cryst. Growth* **208**, 100–106 (2000).

97. Chen, C., Sun, S., Chou, M. M. C. & Xie, K. In situ inward epitaxial growth of bulk macroporous single crystals. *Nat. Commun.* **8**, 2178 (2017).

98. Chakrapani, V. *et al.* Electrochemical Pinning of the Fermi Level: Mediation of Photoluminescence from Gallium Nitride and Zinc Oxide. *J. Am. Chem. Soc.* **130**, 12944–12952 (2008).

99. Hu, C., Zeng, X., Cui, J., Chen, H. & Lu, J. Size Effects of Raman and Photoluminescence Spectra of CdS Nanobelts. *J. Phys. Chem. C* **117**, 20998–21005 (2013).

100. Ravindran, T. R., Arora, A. K., Balamurugan, B. & Mehta, B. R. Inhomogeneous broadening in the photoluminescence spectrum of CdS nanoparticles. *Nanostructured Mater.* **11**, 603–609 (1999).

101. Shi, W. *et al.* Raman and photoluminescence spectra of two-dimensional nanocrystallites of monolayer WS 2 and WSe 2. *2D Mater.* **3**, 025016 (2016).

102. He, Z. *et al.* Revealing Defect-State Photoluminescence in Monolayer WS 2 by Cryogenic Laser Processing. *ACS Nano* **10**, 5847–5855 (2016).

103. Li, C. *et al.* Highly pure green light emission of perovskite $CsPbBr_3$ quantum dots and their application for green light-emitting diodes. *Opt. Express* **24**, 15071–15078 (2016).

104. Zhao, Z.-Q., Zheng, P., Xu, S.-T. & Wu, X. Object Detection With Deep Learning: A Review. *IEEE Trans. Neural Netw. Learn. Syst.* **30**, 3212–3232 (2019).

105. Pak, M. & Kim, S. A review of deep learning in image recognition. in *2017 4th International Conference on Computer Applications and Information Processing Technology (CAIPT)* 1–3 (2017). doi:10.1109/CAIPT.2017.8320684.

106. Wang, W., Yang, J., Xiao, J., Li, S. & Zhou, D. Face Recognition Based on Deep Learning. in *Human Centered Computing* (eds. Zu, Q., Hu, B., Gu, N. & Seng, S.) 812–820 (Springer International Publishing, 2015). doi:10.1007/978-3-319-15554-8_73.