UNIVERSITY OF MELBOURNE

COMP90055 Computing Project

Counting Cells from Microscopy Images Using Convolutional Neural Networks

Author:

Jizhizi Li

Supervisors:

Andrey Kan

Xuan Vinh Nguyen

June, 2016

Academic Year 2015/2016

Declaration

I clarify that

- this thesis does not incorporate without acknowledgement any material previously submitted for a degree or diploma in any university; and that to the best of my knowledge and belief it does not contain any material previously published or written by another person where due reference is not made in the text.
- the thesis is 7000 words in length(excluding text in images, table, bibliographies and appendices)

Signed: _____Jizhizi Li_____

Data:_____05/06/2016_____

Abstract

Microscopy images are widely used in biology applications today. Our aim is to accurately estimate the count in microscopy image. Without spending time detecting or segmenting cells clearly, we cast the problem of estimating number of cells to neural network framework. In this paper, we propose a new Convolutional Neural Network framework for solving problem of counting cells from microscopy images. Inspired by LeNet model, our framework is build by improving LeNet model with some network inner changes. Data used for training in this model is generated based on limited amount of original data with ground truth. After generating training data and building complete framework, our model is able to predict the number of cells in a new microscopy image with the Pearson correlation accuracy between predicted and observed values around 0.9 after training. Many relative experiments have been done in this project in order to have a better understanding of the effects of varying CNN structure.

Acknowledgements

First and foremost, I have to thank my research supervisors, Dr. Andrey Kan, the postdoctoral researcher at Walter and Eliza Hall Institute of Medical Research and Dr. Xuan Vinh Nguyen, the data scientist at University of Melbourne. Without their assistance and dedicated involvement in every step throughout the process, this paper would have never been accomplished. I would like to thank you very much for all your support and help over this semester.

I would also like to thank my family and friends for supporting me spiritually through writhing this thesis and my life in general.

Table of Contents

List of Tables

List of Figures

1. Introduction

Counting problem refers to estimating the total number of one object in a stable image or some video frames. As one example of counting problems, counting cells from microscopy images has important applications in biology research. A reliable method to count cells is of paramount importance because it will often affect the performance of following steps. Numerous procedures in medical and biology research are with the need of counting cells, for example: the number of cell are used when doing analysis in drug screening, the changes of the cell number during the process of tracking cell cycle after adding drugs can reflect the medicinal property of the drug[12]. Cell number can also be used when building mathematical model which is useful to describe cell behaviours[7].

However, even with technology this day and age, cell counting is still largely implemented manually which makes it tedious and time-consuming[8]. Manually counting can also cause mistake due to some human error. Based on this motivation, we focused on building a machine learning method to be able to count the number of cells automatically, which in our case, is to use Convolutional Neural Networks.

Convolutional Neural Networks (CNNs) is a type of feed-forward artificial neural network. Multilayer networks with CNN is able to learn complex and non-linear mapping from large collections of training data to make them become obvious candidate for speech recognition or image recognition tasks[1]. More introduction and background about CNNs are described in section 2.1 of this report.

CNNs is also widely used in cell segmentation problem, there are many references discussing it which can be viewed in section 2.2. The reason for us to choose our research interests in cell counting but not cell segmentation is that cell segmentation is a more complex problem which is much more difficult to have ground truth than cell counting[9]. Since in many process like cell merging or cell devision, the margin of a cell is ambiguous and difficult to decide. But the number of cell is absolute at most time. Based on this, to produce a model that can count cells from microscopy images without to detect or segment prior object is our objective.

Original dataset used in our project consists of images in different frames within a time periods. Due to the character of the dataset, there are two kinds of challenges in our problem, one is the substantial variability in cell appearance across experiments. This means the shape and moving of cells differs with the changing of time.

In addition, as one of the prevalent neural network model, CNN also requires a larger dataset to train, which is also the other problem we have encountered. Even though ground truth for cell

counting is easier to acquire than for cell segmentation, it still consume a lot of time to do it manually, sometimes even unrealistic for entire video. For instance, we need around 1000 dataset to train a model well, but that is way too large for people to mark the ground truth of these data one by one. So in our case, we have 100 images as the original data, this is potentially not enough to train a model well, that makes another aim of our project is to generate more data.

To summarise, there are three principle aims in our project:

1. To generate more data based on the limited original ground truth we acquired. Specifically, the size of train data used in CNNs model need to be grown from hundreds to thousands.
2. To build a Convolutional Neural Network model which takes N dataset as input, and can be used to predict the number of cell in a new test image with a reasonable accuracy, for example 0.9 for Pearson correlation in our case.
3. To handle the variability from cell to cell by using data based on real time frame.

Besides these principle objectives, while developing the CNNs model in our project, we can also assess to what extend does the lack of data influence the ability to train a generalisable model. Experiments have been done to explore this relationship.

In the following sections, more detail of our project are shown from many aspects. In section 2, we review related work of the same topic, we also introduce CNNs in detail as background. Section 3 outlines our approach in this project and explains pre-processing we have implemented on original dataset. In section 4, we describe our architecture layer by layer by showing the specific functionality of each layer and their parameters. Section 5 discusses the improvement of our CNNs model. By presenting experiments results, comparing the performance of network by changing many factors we can get a better understanding of the architecture we have build. More evaluation and analysis are given in section 5. Section 6 is the conclusion and future work of this project.

2. Background

2.1 Convolutional Neural Network

Neural networks with the use of convolutional neural network performs better than normal, non-CNNs model in image recognition problem. The reason can be concluded into three aspects.

Firstly, images, when represented by pixel metric, often has large amount of features. If we use fully connected network on it without any processing on features, overfitting problem may happen because we need numerous hidden units in fully connected network to fit it. Besides, working with so many features and parameters is a heavy task for hardware implementations[10]. However, with

the use of convolutional layer, the number of features that needed to be processed will decrease dramatically by receptive field and parameter sharing.

Secondly, before sending images to fully connected network, they need to be size-normalised and entered in input fields[1]. This operation may cause the size or position variation for individual features of images which will be a problem in the following working. But convolutional networks has obtained the ability of shift invariance because replication of weight are required during the operation.

Thirdly, when turning image into 2D array used in the following fully connected network, the relative position of each pixel and their local relationship is lost[6]. This information can be sometimes very important during processing. To avoid this lost, receptive fields involved in convolutional network force the extraction of local features which will retain this information.

LeNet Model

LeNet model, first appeared in Yan LeCun's work[5] in 1989. This model consists of 7 layers as shown in Fig.1. It takes a 32×32 pixel image as input, generated 6 features maps in layer C1 which is a convolutional layer. Then use a sub-sampling layer S2 to half the number of features in layer C1. Following by layer C3 and S4 and C5 to use convolutional function twice. Finally followed by a fully connected layer names as F6. This network has become the well-known digit classification tasks that have been used on million checks per day with a stable and high accuracy.

In our project, we have adapted original LeNet implementation to fit with our problem. Instead of using sigmoid activation in last layer, we use linear regression to make it work as a regression problem. More detail about architecture in our project is shown in following sections.

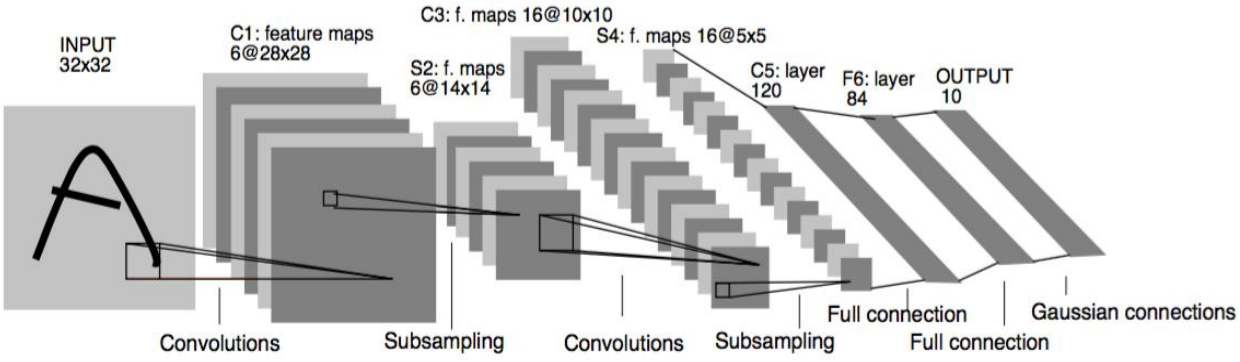

Fig.1 Construction used in LeNet Model

2.2 Related Work

With the developing of combining neural network with biology or medical research, many people have already done excellent research on cell counting or cell segmentation by using machine learning methods. Principle ideas in several typical relative papers are shown here as well as their research result.

Based on the similar aim to count cells from microscopy with the use of convolutional neural network, [2] which published in 2015, proposed a Fully Convolutional Regression Networks which allow end-to-end training for regression of images of arbitrary size. Inspired by [3], which developed a fully convolutional network for semantic labelling, [2] reinterpreted the fully-connected layer of a classification net as convolutional and fine-tuning unsampling filters, which can produce a correspondingly-sized output after trained on input with arbitrary size.

Two kinds of networks have been present in [2], called as FCRN-A and FCRN-B. They are shown in Fig.2. as blue and green one. The former one used 3×3 as the size of convolutional kernel and used three max-pooling layers to aggregate spatial information which is leading to an effective receptive filed with size 38 ×38. And the latter one used max-pooling layer after every two convolutional layers to avoid excessive loss of spatial information.

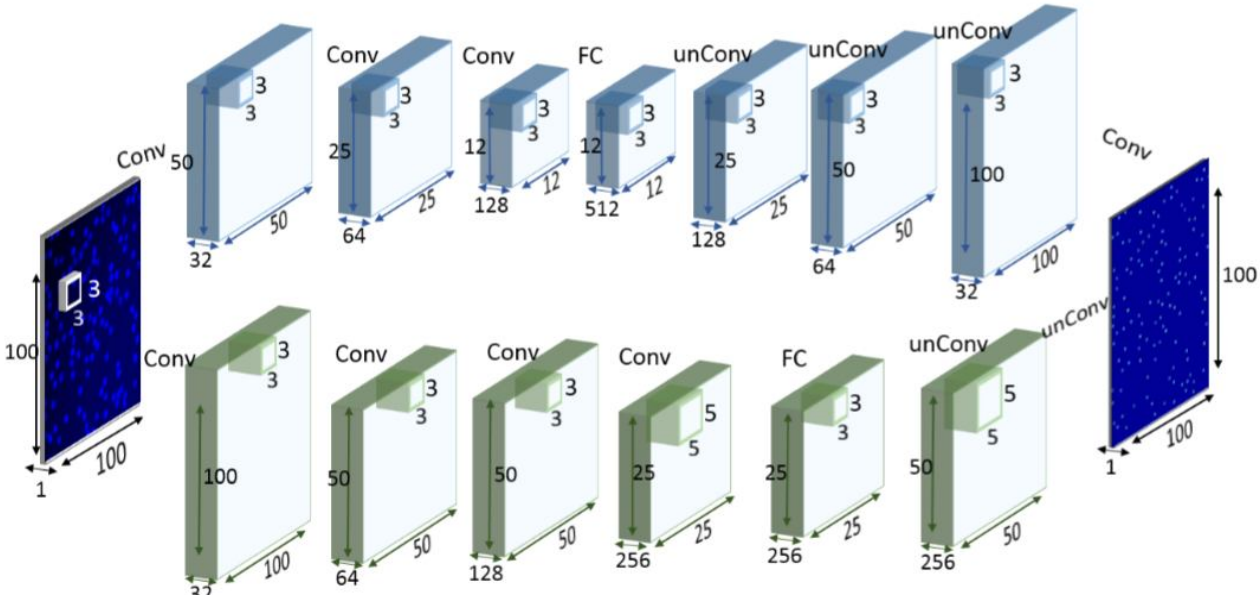

Fig.2 Constructs FCRN-A and FCRN-B used in [2]

After trained on 200 fluorescence microscopy cell images, this method got Ground-truth / Estimated count = 698/705 on test set as its final accuracy, which is a pretty good result. Some of the results can be shown in Fig.3 with the ground truth data and estimated density map.

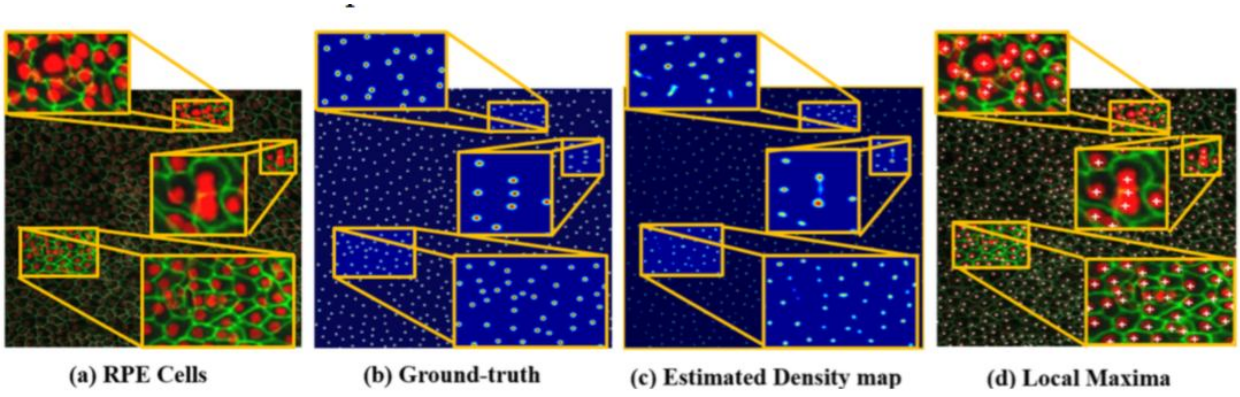

Fig.3 Ground truth and predicted result in [2]

[4] also presented methods to use CNNs to detect edge automatically. Network architecture they have used is one layer of convolution and three max-pooling after it to decrease the feature number of input image. Finally with a fully connected layer to output the final result. Evaluation has been done based on different epochs and also presented a good result.

[13] discussed about how to solve the problem of insufficient training data, they propose eight tangent vectors for i.e. scaling, rotation, X-translation, Y-translation, parallel hyperholic, diagonal, hyperbolic, thickness and modified thickness. These are all good ways to generate more data in our case. But we have only implemented cropping approach and in the interest of time we cannot explore other ways. They can be left to future work of this project.

Inspired by previous works shown above, we propose a complete Convolutional Neural Network with the combination of fully-connected layer and supervised layer based on the idea of LeNet model to achieve the objective to count the cell from microscopy images. Compared to [2], the fully-connected layer we have used can reserve global information. And bigger convolutional kernel(5×5) is used in our model to retrieve more information within image.

3. Methods Overview

3.1 Outline of Approach

In order to develop an approach for using Convolution Neural Network to count cells from microscopy images, we need to finish the following steps one by one.

First, we need to obtain enough data to be used as training set of our model. They need to be pre-processed well which means with the same size, format, representation way and reasonable label. The method we take to do this is shown in section 3.2.

Second, we need to build a complete network model with the use of Convolutional Neural Network. This model should have proper construction with justification for each layer. Information about architecture used in this project is present from section 4.1.

Third, after building the complete network model, it can be used to work. We can feed the dataset generated after pre-processing to this model and use optimisation method to learn the best parameters which are fit for our case. SGD(Stochastic Gradient Descent) is used in optimisation, more detail are described in section 4.2.

Fourth, after training and testing on test data during experiments, some problems or results will show up. By changing a number of factors related to this model, we can get the difference between performance on test data. After analysing on results, we will be able to improve our model. Experiments, results, analysis and improvement are discussed in section 5.

3.2 Pre Processing

Problem scenario in this project is shown in Fig. 4(left). It is one of the specific wells containing cells in Fig. 5. Fig.4(right) provides the ground truth value as dot annotations, where each of the dots correspond to one cell marked manually. The coordinates of these dots are provided as the X and Y coordinates of the absolute position of each cell. The number of the dots is the number of the cells.

However, our original data set is very limited, we have 100 images[1] like shown in Fig. 4(right). They are taken on the same well in different time frame. The problem is that these 100 images have very similar number of cells and the amount is too limited to train a model well. So we need to generate more dataset based on original data.

The original size of these 100 images is about 400 × 400. We cropped numerous smaller image from the original image randomly with the size defined by user. For example, we can get the smaller image with the size 50 × 50 after cropping. Because we can get the average radius of this kind of cell from ground truth data, and the coordinate value of each individual cell is also given. We define that if the distance between the boundary of the smaller image and the centre of the cell is larger than its radius, this cell is considered in the smaller image. When applied to another kind of cells, the radius need to be defined again. In our case, the average radius of this kind of cell is 5, if there is one cell with coordinate as (100,100), then we define the cropped image with the boundary as x1<95, x2>105, y1<95, y2>105 contains this cell.

[1] Original data used in this project is provided by Walter Eliza Hall Institute of Medical Research.

By this way, we can generate training data with any amount and any way as we want. For instance, by generating 20 smaller images for each original one, we can get 2000 smaller images constructing the training data, with the number of cells contained in each smaller images as their label. Or we can also generate 1000 smaller images on one of the original images and use them as training set. To ensure there are not overlapping between training set and test set, we split 100 images to two half, one to generate training set and another one for test set.

What we are going to do next is to build a CNN model that can predict the number of cell of a new image by training on training set we generated. Besides we are going to find out the relationship between the performance of the network and the way we generating the training data, and the amount of our training set. Fig. 6 shows some examples of the training set.

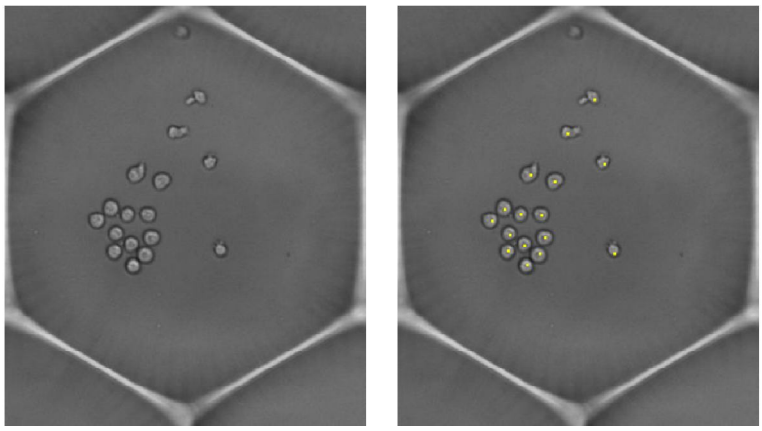

Fig.4 Original well image and the one with coordinates printed

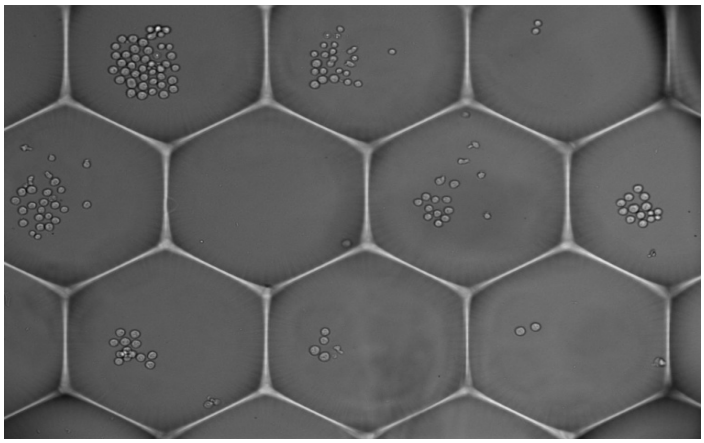

Fig.5 Microscopy image with all wells in it

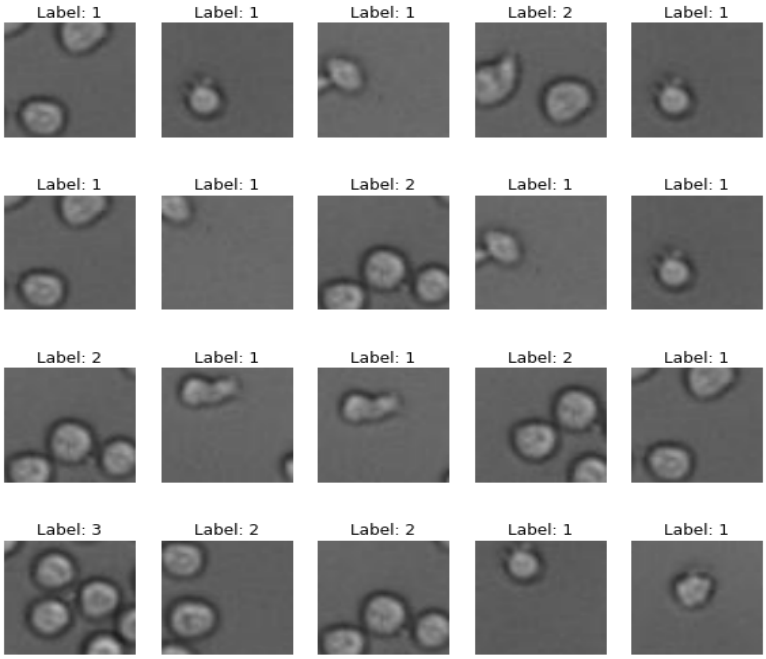

Fig.6 Some examples of training set

4. CNN Architecture

Based on the idea of LeNet, our initial architecture model is shown in Fig.7.

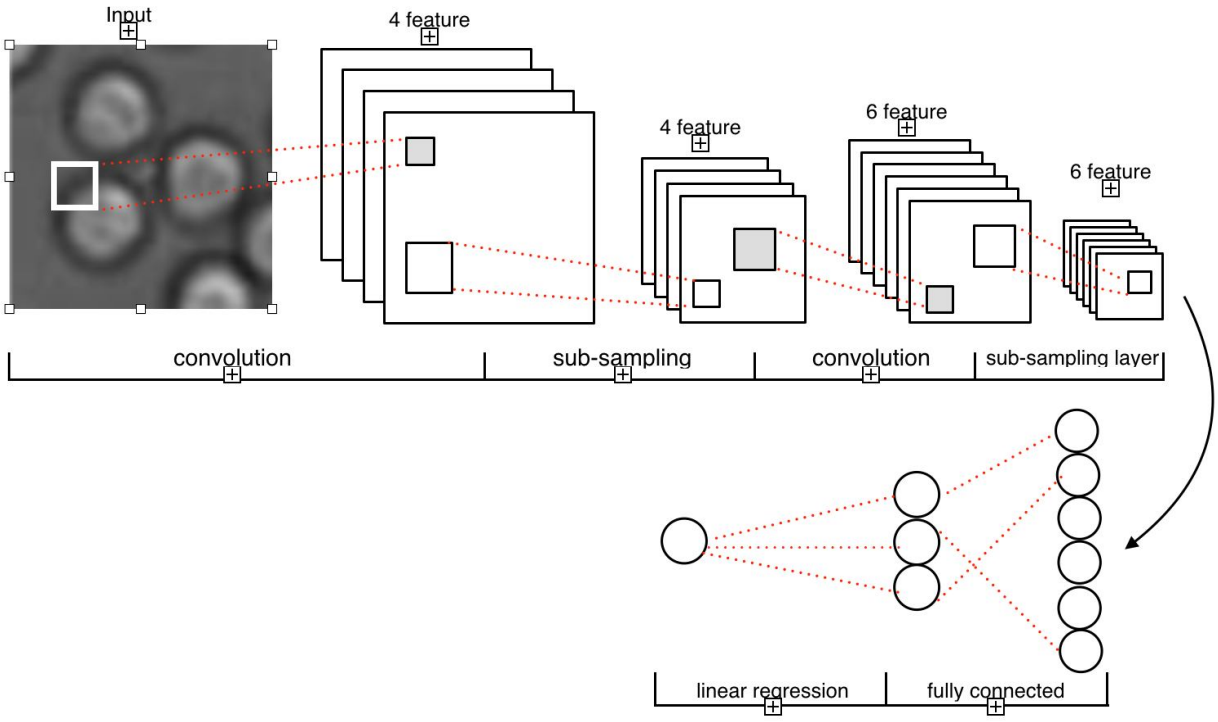

Fig.7 Architecture used in our model

In general, our model consists of a convolutional layer follower by a pooling layer, another convolutional layer followed by a pooling layer, and then one fully connected network layer, finally

the linear regression layer to generate the predicted output. Each layer is described specifically in the following part.

Theano in Python, which is a library that allows one to define, optimise, evaluate mathematical expression involving multi-dimensional arrays efficiently is been used to implement the model.

4.1 Layers in Architecture

4.1.1 Convolutional Layer

As shown in the first layer of Fig. 7, convolutional layer works directly on the features of original dataset that are approximately size-normalised and centred, process them perfectly before using fully connected layer later[11].

Normally, there are three components known in convolutional layer, they are local receptive fields, shared weights and spatial subsampling. With the help of these three components, the features of image will be extracted down to a limited amount.

Local connections, first appeared in early 60s with the concept of perceptrons, was known when Hubel and Wiesel discovered locally orientation-elective, locally sensitive neurone in cat's visual system[7]. After this, the concept of local connection have been continuously used in many neural models of compute vision. Like in the research of Yan LeCun's about LeNet model we introduced in section 2. The advantage of using local receptive fields is that many fundamental visual features like ending points, corners and oriented sides can all be extracted.

Besides, these features detected used on part of image can also be used on other part of image, even on the entire image because of the help of shared weights. To implement this, we just need to use identical weight vectors on different part of image, or replicate same filter across the entire visual field. Gradient descent or other optimisation method can still be used to find the best shared parameters, actually the gradient value of a shared weight is just the sum of the gradient of each weight that is been shared.

The advantage of using repetitive units in entire image is that features can be detected in spite of its absolute position in visual field, in this case, its approximate position relative to others is much more important than their exact locations. Additionally, by using share weight, the parameter used in network will be decreased significantly, the efficiency of model learning has also been increased.

The output after using local receptive fields and shared weights is called as feature map. By using multiple convolutional kernel in our project, multiple feature maps in same layer can be obtained to increase final accuracy. Theses feature maps will be used in the following layers.

Within operation, the formula of convolution in 2D can be described in Equation.1. After acquired multiple features maps in one layer, the weight W of a hidden layer can be saved as a 4D tensor variable that containing every combination of the destination feature map, source feature map, source horizontal position and source vertical position[6].

Another important concept of CNN model is max-pooling which is the second layer in our architecture, it takes the features maps we obtained before as input. The detail of max-pooling layer is described in the next section.

$$o[m,n] = f[m,n] \cdot g[m,n] = \sum_{u=-\infty}^{\infty} \sum_{u=-\infty}^{\infty} f[u,v] g[m-u, n-u]$$

Equation 1. Convolutional for a 2D object

4.1.2 Subsampling

Feature maps generated after first convolutional layer can be used for regression or classification problem in theory, but the number of features remained is still too large to be processed for upper layer. It will consume longer time, decrease learning efficiency and even cause problem of overfitting.

To solve this problem, another important concept of CNN network appears, that is max-pooling. Max-pooling is actually a way of non-linear down-subsampling. The way it did that is to split the original images into multiple non-overlapping rectangles and obtain the average value or maximum value of each sub-region.

There are two pros of using max-pooling layer after each convolutional layer. One is what we have pointed out before, by using max-pooling layer, non-maximal value will be eliminated (if maximum value is been use as threshold). In this way, the amount of feature will decreased, it can even be half of the original size if we use [2,2] as our max-pooling region. Computation task for following layer will be reduced distinctly.

The other advantage is with the use of max-pooling, the sensitivity of the output to shifts and distortions will also be reduced. For instance, after convolutional layer, each single pixel in the

input image will have 8 direction. But after a [2,2] max-pooling region, 3 of 8 possible configurations will end up with identical results at convolutional layer which means the number of direction feature is down from 8 to 5[6].

In Theano library, max-pooling is implemented by method of `theano.tensor.signal.downsample.max_pool_2d` which takes N dimensional tensor variable and a downscaling factor as input, then operate max-pooling over the 2 trailing dimensional of that variable.

max-pooling layer and convolutional layer construct a complete convolutional layer. They have a lot of parameters, like the size of convolutional kernel, the number of feature maps in one layer, the size of max-pooling region and so on. They can all be changed to test the difference performance of network.

In our architecture, layer 3 and layer 4 are convolutional layer and max pooling layer again. By using CNNs twice, the number of features for each map in one case can be decreased from 50×50 to 9×9 with the size of convolutional kernel set as [5,5], and the size of max-pooling region set as [2,2].

4.1.3 Fully-connected Layer

After using convolutional layer two times, we add one fully connected layer before using linear regression. The number of input parameters and output parameters can both be set due to different situation.

Like in any classical neural network, units in fully-connected layer performa a dot product between their input vector and weight vectors, bias is also added. The formula used in fully connected layer is shown in Equation.2. a_i represents weighted sum of unit i, a sigmoid squashing is used to produce the state of unit i which is reprinted as X_i. The squashing function in fully connected layer is a scaled hyperbolic tangent, with formula as shown in Equation.3.

Fully-connected layer is used here to ensure that the model also consider the global character of the image other than only considering local character.

After fully-connected layer, we flatten the output into 1D to be used for linear regression layer.

$$X_i = f(a_i)$$

Equation.2 Formula used in fully connected layer

$$f(a) = A\tanh(Sa)$$

Equation.3 Hyperbolic tangent used as squashing function in fully connected layer

4.1.4 Linear Regression

As the last layer in our architecture, linear regression layer is used to predict out the final output. With the formula as shown in Equation.4, linear regression takes the output of fully-connected layer as input, use least square as cost function to optimise the parameters β_i. The number i, which is the number of input features in this layer, can also be changed in network model.

$$y_i = \beta_i x_{i1} + \ldots + \beta_p x_{ip} + \varepsilon_i = x_i^T \beta + \varepsilon_i \qquad i=1,\ldots,n$$

Equation.4 Linear regression

4.2 Training the Model

After the complete network model has been build, the next step is to train the model based on training set generated in previous section. Because Theano is the library we are using to implement the CNNs model and parallel computing is needed when dealing with large amount of dataset, we use theano.shared to store the data as shared type, so that they can be copied into GPU and get speed increased[2].

The dataset is spliced into three parts, first is used for training which is called as training set, second is used for validation which is called as validation set, third is the test set which is used for testing.

Each layer is programmed as a python class with their cost function, activation function and parameters defined within class.

Convolutional layer and max-pooling layer are combined as one layer, which takes several input like parameter *rng* with the type of bumpy.random.RandomState, used to generate initial weight random number. Parameter *input*, with type as theano.tensor.dtensor4 and size defined in advance, used in this layer. Parameter *filter_shape*, with type as tuple, is defined to describe the size of convolutional kernel used in this layer, and the number of features maps of last layer and this layer. Parameter *image_shape* is used to illustrate mini batch size used in GPU implementation, number of feature maps in last layer, the height and width of input image. Parameter *poolsize* is a tuple with the length of 2, defined to rule the row and column of downsampling factor.

[2] Code used in this project is open-source, can be viewed at https://github.com/JizhiziLi/CellCounting_CNN

Fully-connected layer is a typical hidden layer of a MLP, non-linear activation function used in our model is tanh. Weights are also initialised as random number, number of hidden units and output units are both the input of fully-connected layer.

Cost we minimise during the training process is the cost function of final linear regression layer, which is least square in our case. Then we create a function to compute the mistake made by the model on test set and validation set.

Next, parameters in each layer are put into a list params to be fit by optimisation, which is gradient descent in our model. *Grads* is another list of gradients calculated in each layer. During parameters updating, we use SGD since the number of parameter in this model is large. Instead of manually create an update rule for each parameter which is a tedious task, the updates list we are using is to automatically looping over all (*params[i], grads[i]*) pairs.

In the process of training, patience is defined as the number of examples been calculated regardless, it will be updated within the iteration. In every epoch, we check on validation set to get the error rate. If there is no longer new best appears after waiting patience_increase times, we choose early stop to get our final validation error and test error.

All the parameters defined above can effect the performance of network. The relationship between the network parameters and performance, the evolution way to judge the performance of network are described in section 4.

4.3 Evaluating Model

After training model, we save the parameters of each layer in disk as pickle file. Then we generate totally new test data just as the way we generate training data. We then build same network as the one used to be trained, take test data as input, fit parameters saved before to each layer, predict one final output as the predicted label of test data. By comparing the predicted label with true label of data, we can get the performance of our model constructed and our training process.

Two methods for evaluating we are using are RMSE and Pearson Correlation. RMSE, which is short for root-mean-square error. RMSE is used to measure the difference between values predicted by a model and the value actually observed. Pearson Correlation is used to measure the linear correlation between two variables, in this case are real value and predicted value. Relative experiment example and result analysing are shown in section 5.

5. Experiments and Results

In this section, we first determine how our CNNs network performs comparing with simple linear regression model. Then we use multiple ways to generate our training data to see what difference can it make. Next we change the construction of our model by trying different number of layers, different kernel size or other setting of the configuration are used to see how if affect our performance.

5.1 Compare with Simple Linear Regression without CNN

In order to show the significance of CNN, we train the dataset by normal simple linear regression and our model to compare. Same train set and test set are used for two models.

Generating 20 smaller images for first half(50 images) of our original images makes 1000 dataset to train our model. Test set is generated from the second half of the original images and has the same size as training set. 10-fold Cross-validation are used in both model to increase accuracy and avoid problem of overfitting.

We use root mean square error and Pearson correlation to reflect the performance of the results. Patience and learning rate in our model are set to 8000 and 0.0001.

Table 1 shows the result of RMSE and Pearson correlation result of using the trained model on training set and testing. Seen from the table, with the use of simple linear regression without CNNs, even though the trained model has 0.77 as the Pearson correlation on training set, but when using on test set, the Pearson correlation decreases to 0.06 which means the model does not work. However, after trained by our CNN model, the Pearson correlation is 0.9036 on training set and 0.9003 on test set. Same comparison is shown from RMSE value. Simple linear regression generates 1.8951 for test set while RMSE of our model on test set is 0.4404.

Table 1 Comparison of RMSE and Pearson Correlation with simple linear and our model

	Train Set		**Testing Set**		
	RMSE	Pearson	RMSE	Pearson	Train Time(Secs)
Simple Linear	0.7337	0.7682	1.8951	0.0644	0.89
Our model	0.4450	0.9036	0.4404	0.9003	207.54

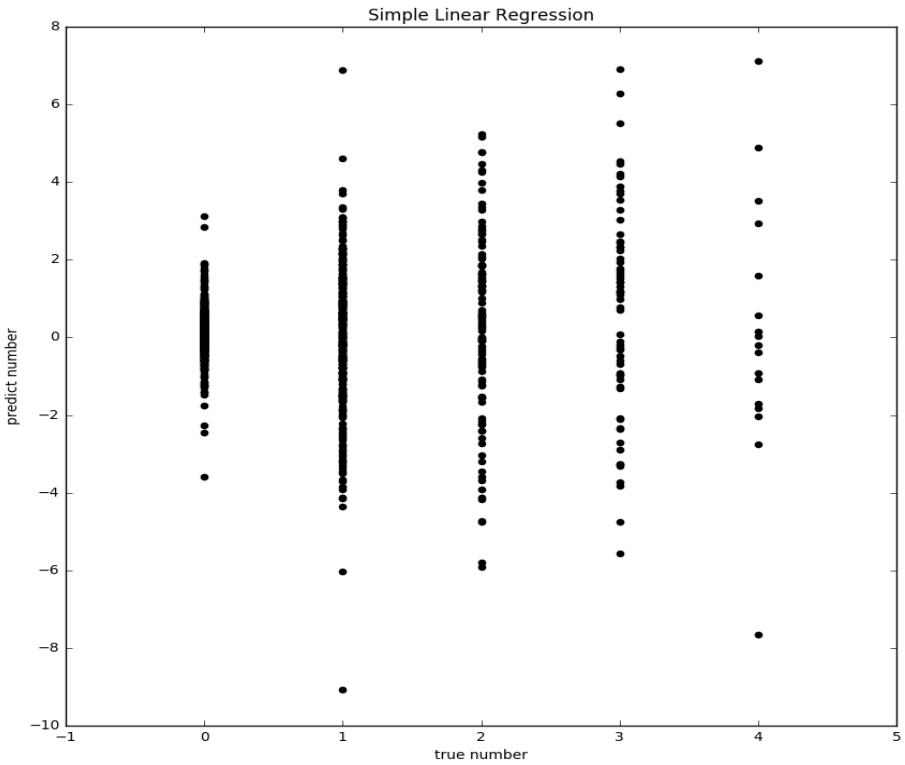

Fig.8 Scatter plot with simple linear

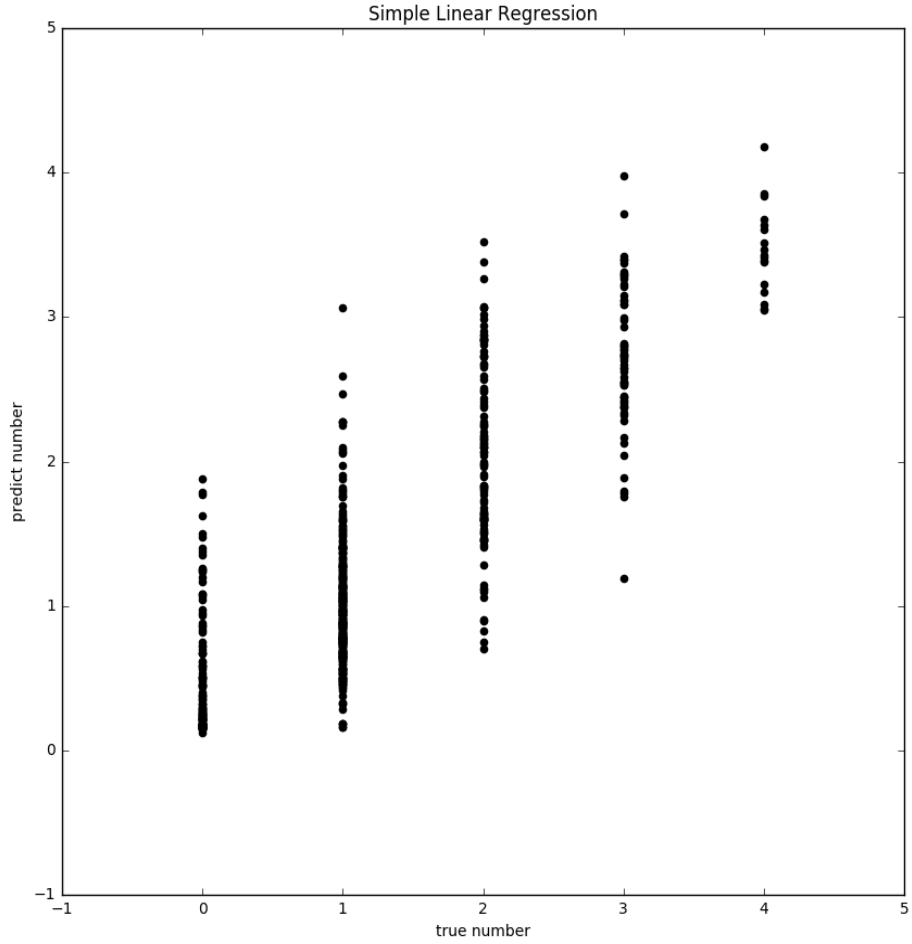

Fig.9 Scatter plot with our model

Fig.8 and Fig.9 shows the comparisons explicitly by plotting the scatter plot based on test set. X axis of the plot is the true number of cells in image, Y axis is the number of cells in image predicted by model. Fig.8 is the result by using simple linear regression, which shows non linear relationship between the true number and the predicted number. However in Fig.9, which is the result by using model with CNNs, the linear relationship between X and Y is obvious. The comparison of the results between using two kinds of model has shown the importance and meaning of using CNNs.

5.2 Use Different Ways to Generate Training Data

Ways of generating training data may affect the performance of the result. In previous section, we generate the training data on 50 images with 20 smaller images on each to make full use of the whole original dataset. In this section we will generate training data based on only one image of original data and split 1000 smaller images on it to see the difference. Both training sets are of the same size and same testing set are used to test the result. Patience and learning rate are set to 8000 and 0.0001. Same model as introduced in section4 is used to test this effect.

Seen from table 2, the result of generating training data only with one image has lower Pearson correlation value than generating based on 50 images. The reason for this is because data from only one image is too limited to train a model well. more images used could provide various sample to train.

Table 2 Comparison of using different way to generate training data

	Train Set		**Testing Set**		
	RMSE	Pearson	RMSE	Pearson	Train Time(Secs)
50 images	0.4450	0.9036	0.4404	0.9003	207.54
1 image	0.4225	0.8878	0.4494	0.8890	182.13

5.3 Changing the Construction of Model

In this sub section, construction of the model are changed from multiple aspects. Section 5.3.1 shows the result of using different number of (convolutional + max-pooling) layer in our model. Section 5.3.2 tries different options of convolutional kernel size and amount of feature maps in each layer. Section 5.3.3 changes the number of output units in fully-connected layer, which is also the number of input units in last linear regression layer to see how it affect our results.

5.3.1 Different Number of Convolutional and Max-pooling Layer

Convolutional and max-pooling layers are the significant part of our model, twice of both are used in our original model. How will it change the result if we adding more convolutional layers or

deleting some convolutional layers is the problem we work on in this part. Patience and learning rate used are set as 8000 and 0.0001.

Seen from Table 3 and Diagram 1, when only use one layer of convolutional and max-pooling, we can get better Pearson Correlation in training set than two layers. But as for testing set, the performance of only one layer is worse than two layers. And the difference between testing on train set and testing set is larger in one layer than two. One possible reason for this is that since the original size of one input image is 50 × 50, after only one convolutional and max-pooling layer with the convolutional kernel size as 5 and max-pooling filter size as 2, there still exist 23×23[3] features sent to fully-connected layer. But with the use of two layers, this will change to 9×9. So in this case, too many features in one layer may cause the problem of overfitting which could explain the reason of the decrease in testing set Pearson correlation.

Table 3 Comparison of changing number of convolutional and max-pooling layer

	Train Set		**Testing Set**		
	RMSE	Pearson	RMSE	Pearson	Train time(secs)
One layer	0.4222	0.9129	0.4546	0.8944	103.38
Two layers	0.4450	0.9036	0.4404	0.9003	207.54
Three layers	0.4828	0.8787	0.4478	0.8899	247.37

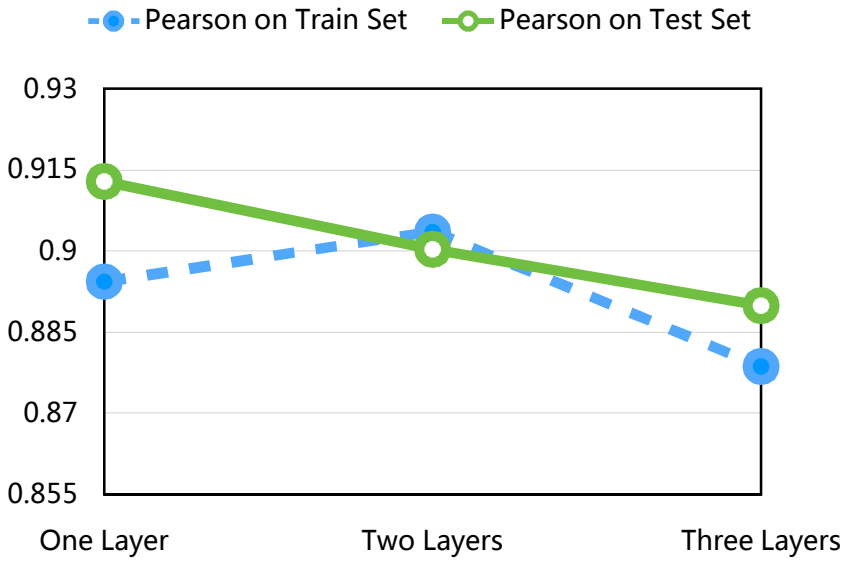

Diagram 1 Comparison of Pearson Correlation of train set and test set on three options

[3] The original size is 50, after first convolutional layer, size = (50-5+1)/2=23

As for three layers, the number of features in each individual image is 2×2 after three times of convolution and max-pooling. This number s way too small to sent to fully connected layer, which may cause the problem of under-fitting. This could also explain why Pearson correlation in both train set and testing set for three layers are lower than the other two.

Based on this experiment, two layers of convolutional and max-pooling is the best choice for our case, with the size of input image as 50 × 50, convolutional kernel is 5 and max-pooling filter is 2.

5.3.2 Changing Filter Shape and Number of Filters

Filter shape is the size of convolutional kernel used in each convolutional layer. It decides how big the size of small potion of the image is being processed each time. By changing the filter shape, the number of features after processing is different. The smaller the kernel is, the more the features reserved.

Number of filters decides how many feature maps generate in per layer. The larger the number of filters is, the more complete the information we retrieved from image is. However, because computing activations of a single convolutional filters is much more expensive and complicate than traditional MLP[6], the number of filters used in CNNs are usually set less than units of hidden layer.

Table 4. Comparison of RMSE and Pearson based on different filter number/shape

Filter Shape	Number of Filters	Train Set		Testing Set		
		RMSE	Pearson	RMSE	Pearson	Train time(Secs)
[3, 3]	[5, 10]	0.4708	0.8865	0.4425	0.8939	160.7
[3, 3]	[20,30]	0.4198	0.9120	0.4123	0.9081	893.12
[5, 5]	[5, 10]	0.4526	0.8983	0.4495	0.8971	180.9
[5, 5]	[20,30]	0.4431	0.9015	0.4208	0.9044	1238.75

Here we try three choices of filter shape and two choices of number of filters for experiment. Learning rate is set based on individual case. We can seen from Table 4, the more number of filters we have, the better the performance is , but the cost is a much longer time for the model to be trained. There is not very big difference between use 3 as size of kernel or use 5, they perform as similar level of RMSE and Pearson correlation.

5.3.3 Changing Number of Output Units in Fully-connected Layer

Output units in fully-connected layer is the input units for linear regression layer. How could it effect the final result is what we working in this part. Five choices have been taken to evaluate the performance of network. patience value is set based on individual case.

Seen from Table 5 and Diagram 2, the Pearson correlation is growing for both the train set and test set, and the RMSE is decreasing with the growing of units number from 5 to 500. This is understandable since more units can store more information of image. However the accuracy decrease at the number of 1000, which may because 1000 is too large as input of linear regression layer. Besides, time spent also grows as units number increasing. Based on all, 50-500 is good choice for this case.

Table 5 Comparison of RMSE and Pearson based on different output units

Output Units	Train Set		Testing Set			
	RMSE	Pearson	RMSE	Pearson	Patience	Train Time(secs)
5	0.4851	0.8848	0.4559	0.8899	0.00001	196.82
20	0.4526	0.8983	0.4495	0.8971	0.00001	180.9
50	0.4450	0.9036	0.4404	0.9003	0.00001	216.0
100	0.3209	0.9487	0.4049	0.9157	0.00005	216.93
500	0.2810	0.9655	0.3932	0.9243	0.00005	230.13
1000	0.4031	0.9257	0.4447	0.9067	0.00005	396.61

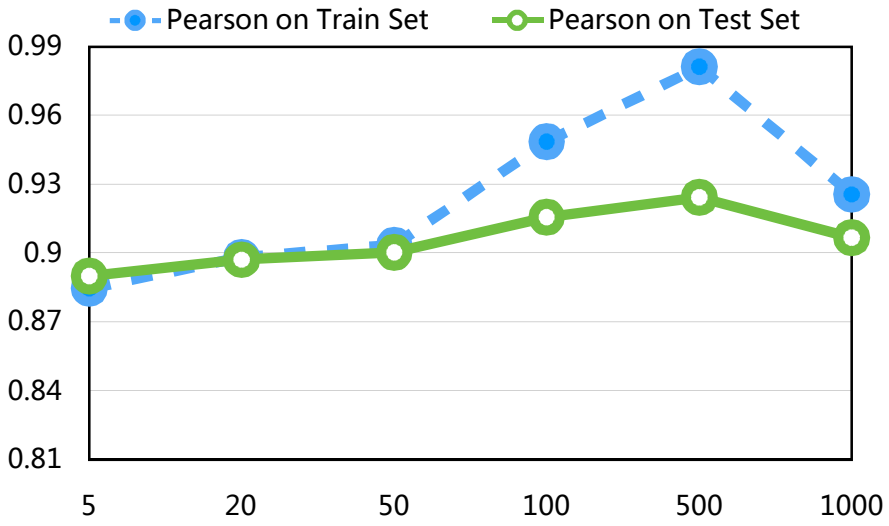

Diagram 2 Comparison of Pearson Correlation of train set and test set on different output units

5.4 Changing Patience Value and Learning Rate

5.4.1 Change Patience Value

Patience value is one of the parameters that could affect the training process. It is a way of regularising CNN. By setting the patience value, model will run on this many examples regardless. And by setting how much long we would like to wait when a new best is found and how much improvement we considered is significant, model will update value of patience within each iteration and break when finding the best result. Learning rate used in this experiment is 0.00001.

Table 6 and Diagram 3 show the effect of using different patience value. Seen from Diagram 3, the Pearson correlation increases with the growing of patience value and stay same after a specific value, in contrast, RMSE decreased with the growing of patience. And for some patience value, model runs at same iteration time, the reason for this is because patience value can be updated within learning process which means it may end up with same result by defining different initial patience value.

What we can get form this experiment is that defining proper patience value is important since the Pearson correlation is too low if our patience value is too small. Large patience value makes the model run same time because they have same iteration times.

Table 6 Comparison of RMSE and Pearson correlation based on different patience value

Patience	iteration	Train Set		Testing Set		
		RMSE	Pearson	RMSE	Pearson	Train Time(Secs)
10	215	1.0294	-0.4247	1.0018	-0.4970	14.36
50	340	1.0090	0.1578	0.9813	0.1150	19.19
100	340	1.0090	0.1578	0.9813	0.1150	19.19
500	500	0.9956	0.6093	0.9656	0.6158	29.61
1000	4000	0.4526	0.8983	0.4495	0.8971	180.9
5000	4000	0.4526	0.8983	0.4495	0.8971	180.9

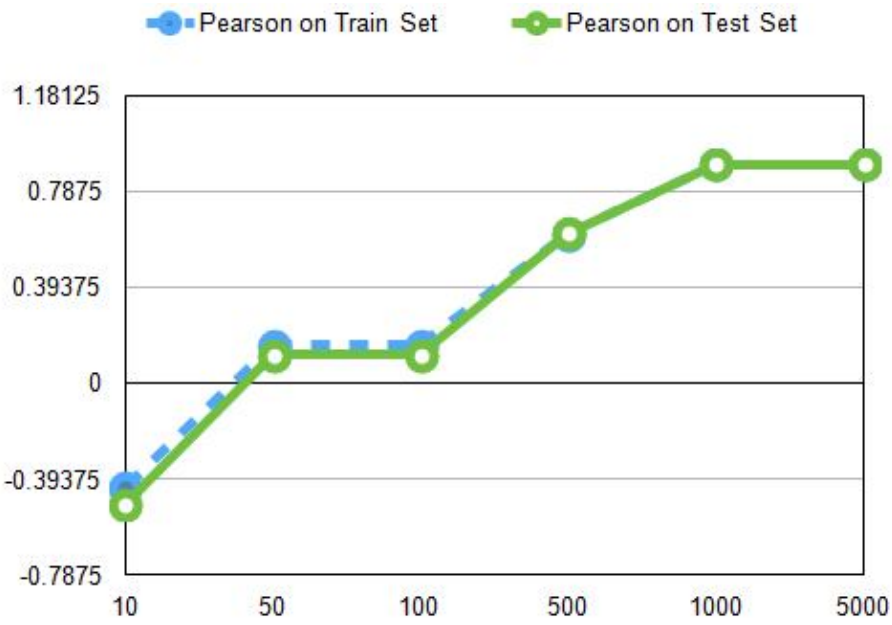

Diagram 3 Comparison of Pearson Correlation of train set and test set on different patience

5.4.2 Change Learning Rate

With same patience set as 5000, we now observe the effect of setting various learning rate. Learning rate is used when doing optimisation by SGD. By setting learning rate, it decide how far the model can go as per step during learning process. Learning rate can not be good when set too large or to small.

Seen from Table 7, for this specific case, learning rate set as 0.00001 or 0.0001 are the best choice. The result performs best at learn rate = 0.0001 but may exist some problem of overfitting. When learning rate grows to 0.0005 or 0.001, the model cannot be trained at all. The reason is with this learning rate and patience value set, SGD optimisation cannot find the best parameters within iteration. This situation is been depicted clearly in Diagram 4.

Table 7 Comparison of RMSE and Pearson correlation based on different learning rate

Learning Rate	Train Set		Testing Set		
	RMSE	Pearson	RMSE	Pearson	Train Time(secs)
0.000001	1.0033	0.4195	0.9753	0.4053	180.52
0.000005	0.6676	0.8661	0.6213	0.8695	176.61
0.00001	0.4526	0.8983	0.4495	0.8971	188.14
0.0001	0.2410	0.9721	0.4089	0.9132	195.61
0.0005	1.0094	NAN	0.9822	-0.0346	189.37
0.001	1.0099	-0.04	0.9829	0.0257	179.04

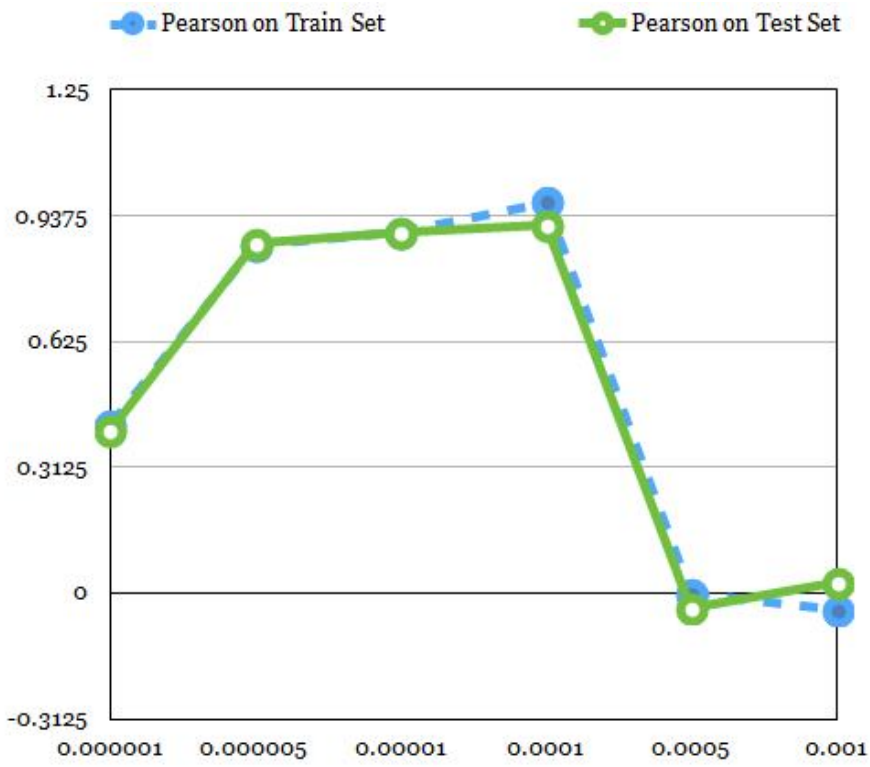

Diagram 4 Comparison of Pearson Correlation of train set and test set on different learning rate

6. Conclusion and Future work

In this project, we have proposed a complete convolutional neural network to solve the problem of cell counting. We train this model on a dataset we generated from limited amount of original data, set new label based on initial ground truth. Size of training set increases from hundreds to thousands after working. Then we evaluate our CNNs model with different network configurations by changing construction of network or changing the parameters used in the model. Relative experiments are done and results are present in paper. We then present some ideas about how to improve the model based on the results we have got.

CNNs is proved to have significant affect on cell counting problem. By using CNNs, the amount of features contained in image can be decreased to limited but important ones. Local information between pixels are reserved perfectly. Together with fully-connected to deal with global information of image, and linear regression to get final output by supervised learning, CNNs makes great contribute in problem processing large features dataset.

Challenging things about CNNs is that they have numerous factors about network that can be changed to make improvement, the construction of the model, the value of the parameters, the setting of some input number can all affect the performance of CNNs. How to get the best combination without causing the problem of overfitting deserves more work. Future work could be done in getting more reasonable and better understanding of the factors' affect.

References

1. Jaswal, D., V, S. and Soman, K.P. (2014) 'Image classification using Convolutional neural networks', International Journal of Scientific and Engineering Research, 5(6), pp. 1661–1668. doi: 10.14299/ijser.2014.06.002.

2. Xie, W., Noble, J.A. and Zisserman, A. (2016) 'Microscopy cell counting and detection with fully convolutional regression networks', Computer Methods in Biomechanics and Biomedical Engineering: Imaging & Visualization, , pp. 1–10. doi: 10.1080/21681163.2016.1149104.

3. Shelhamer, E., Long, J. and Darrell, T. (2016) 'Fully Convolutional networks for semantic segmentation', IEEE Transactions on Pattern Analysis and Machine Intelligence, , pp. 1–1. doi: 10.1109/tpami.2016.2572683.

4. A., M. and A., Y. (2013) 'Automated edge detection using Convolutional neural network', International Journal of Advanced Computer Science and Applications, 4(10). doi: 10.14569/ijacsa.2013.041003.

5. Lecun, Y., Bottou, L., Bengio, Y. and Haffner, P. (1998) 'Gradient-based learning applied to document recognition', Proceedings of the IEEE, 86(11), pp. 2278–2324. doi: 10.1109/5.726791.

6. 2010, - (2008) Convolutional neural networks (LeNet) — DeepLearning 0.1 documentation. Available at: http://deeplearning.net/tutorial/lenet.html (Accessed: 2 June 2016).

7. Wu, H.. and Dikman, S. (2010) 'Segmentation and thickness measurement of glomerular basement membranes from electron microscopy images', Journal of Electron Microscopy, 59(5), pp. 409–418. doi: 10.1093/jmicro/dfq060.

8. Jurrus, E., Paiva, A.R.C., Watanabe, S., Anderson, J.R., Jones, B.W., Whitaker, R.T., Jorgensen, E.M., Marc, R.E. and Tasdizen, T. (2010) 'Detection of neuron membranes in electron microscopy images using a serial neural network architecture', Medical Image Analysis, 14(6), pp. 770–783. doi: 10.1016/j.media.2010.06.002.

9. Li, W., Jia, F. and Hu, Q. (2015) 'Automatic segmentation of liver tumor in CT images with deep Convolutional neural networks', Journal of Computer and Communications, 03(11), pp. 146–151. doi: 10.4236/jcc.2015.311023.

10. Leena Silvoster, M. and Govindan, V.K. (2012) 'Enhanced CNN based electron microscopy image segmentation', Cybernetics and Information Technologies, 12(2). doi: 10.2478/cait-2012-0014.

11. Tygert, M., Bruna, J., Chintala, S., LeCun, Y., Piantino, S. and Szlam, A. (2016) 'A mathematical motivation for complex-valued Convolutional networks', Neural Computation, 28(5), pp. 815–825. doi: 10.1162/neco_a_00824.

12. Kinjyo, I., Qin, J., Tan, S.-Y., Wellard, C.J., Mrass, P., Ritchie, W., Doi, A., Cavanagh, L.L., Tomura, M., Sakaue-Sawano, A., Kanagawa, O., Miyawaki, A., Hodgkin, P.D. and Weninger, W. (2015) 'Real-time tracking of cell cycle progression during CD8+ effector and memory t-cell differentiation', Nature Communications, 6, p. 6301. doi: 10.1038/ncomms7301.

13. Y. Buyoung, C. Kim, and S. Yang (2007) "Recognition of handwritten digit with transformed invariance distance", Project Final Report, EECS 545 Machi ne