Improving classification accuracy with better feature extraction by CNN-SVM

Bingquan Cai, Chuwei Chen, Jiawei Zhao, Xiaowei Ge and Yuxiang Wan*
Department of Electrical and Computer Engineering, College of Engineering, Boston University, Boston, MA

Abstract

In recent year, machine learning techniques are widely implemented to solve the image classification problems. In this report, we implemented a hybrid model, CNN-SVM, where CNN plays the role of feature extractor which can learn from the data set, and SVM plays the role of a generalized classifier. We showed that our hybrid model improved the classification accuracy compared to each method separately. To benchmark our model performance, we compared the accuracy on classical MNIST data set. And for application, we demonstrated the advanced accuracy of our model on biological samples to fill the need of precise image-based methods for cell identification.

1 Introduction

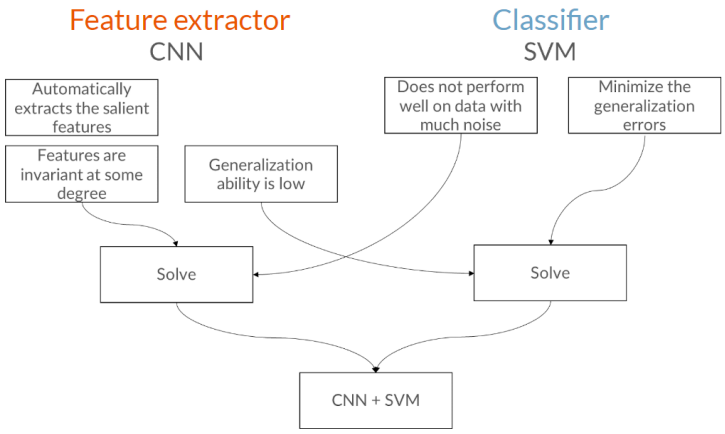

Figure 1: Concept and properties of CNN+SVM

Classification is a central topic in machine learning that has to do with teaching machines how to group together data by particular criteria. Classification is the process where computers group data together based on predetermined characteristics — this is called supervised learning. Classification is an important tool in today's world, where big data is used to make all kinds of decisions in government, economics, medicine, and more. Researchers have access to huge amounts of data, and classification is one tool that helps them to make sense of the data and find patterns. And

*Bingquan, Chuwei, Yuxiang contributed to CNN architecture; Xiaowei, Jiawei contributed to SVM architecture; Chuwei, Jiawei contributed to MNIST data collection; Xiaowei contributed to cell sample preparation; Bingquan and Xiaowei contributed to experiments and performance testing.

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

handwriting digits recognition is one of the most intensively studied challenge. There are numerous research achieved high accuracy using h as K-Nearest-Neighbors (KNNs), Support Vector Machines (SVMs), Neural Networks (NNs), Convolutional Neural Networks (CNNs), and etc. The reason why handwritten digit recognition is still an important area is due to its vast practical applications. One key factor is feature extraction.(1)

A successful classification model requires the extracted features is the most distinguishable, invariant characters in the same class and tolerant to some degree of distortion. In our project, we built a hybrid model CNN-SVM for simple classification problem. Use CNN to extract the features that SVMs can not recognize and replace the output layer of CNN to further improve the accuracy.

2 Related Work

Many researches have been done to explore the performance on the hybrid CNN-SVM model, and most of them achieve an outstanding performance that exceeded the CNN model.

In paper (1), Niu and Suen proposed a structure that used CNN as an automatic feature extractor and it takes SVM to be the output predictor. The efficiency and feasibility of the model were evaluated in the recognition accuracy and the reliability performance. Their experimental results on the MNIST data set achieved an error rate of 0.19 % with no rejections, and 100 % reliability with a rejection rate of 0.56 %.

In paper (2), Darmatasia and Mohamad proposed a similar structure to Niu and Suen. CNN was used as a feature extractor and SVM was the high-end classifier. The proposed method was proven more efficient than modifying the CNN with complex architecture. Based on their experiment result on MNIST SD 192^{nd} edition, the proposed model which combines CNN and linear SVM using L1 loss function and L2 regularization achieved a recognition rate better than only CNN. The recognition rate achieved are 98.85 % on numeral characters, 93.05 % on uppercase characters, 86.21 % on lowercase characters, and 91.37 % on the merger of numeral and uppercase characters. While the original CNN achieves an accuracy rate of 98.30 % on numeral characters, 92.33 % on uppercase characters, 83.54 % on lowercase characters, and 88.32 % on the merger of numeral and uppercase characters.

3 Methods

3.1 Basic and hybrid models

3.1.1 CNN

Architecture In general, the Convolutional Neural Networks (CNN) are made up of the following layer types: the input layer, the convolutional layer, the pooling layer, the dense layer and the output layer. In our project, the architecture of CNN is referenced from the architecture of CNN according to [paper4]. It starts with the input layer which contains the images, so the width and the height should be the dimensions of the images. Then it is the convolutional layer and the ReLU activation function together with the pooling layer. This pattern will repeats twice and the output of the second pooling layer will be reshaped to be a vector for one single image. Finally, after the dense layer with the SoftMax activation function, it is the output layer which shows the prediction of one single image.

Now, we will discuss about the details of CNN architecture.

Convolutional layer Convolutional layer is the core structure in our convolutional neural network. It contains a set of filters, weights and bias parameters. Filters are normally smaller than the input size, but it will convolve with the input and create a feature map. Each filter slides across the height and the width of the input matrix, and calculates the sum of the dot product between every element of the filter and the input at every spatial position.

Pooling layer Pooling layer is usually incorporated between two successive convolutional layers. The pooling layer reduces the number of parameters and computation by down-sampling the representation. It contains a set of batch size and stride. By selecting a filter and sliding it over the output feature map of the preceding convolutional layer, with stride controlling the step size of the sliding

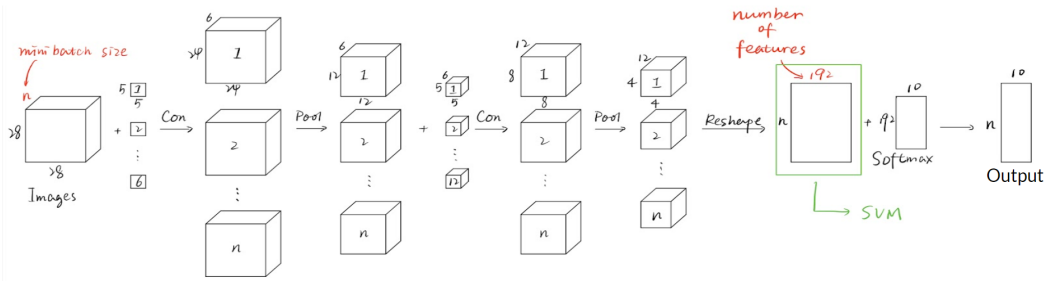

Figure 2: CNN architecture

process, we get a downsampled feature map. There are 2 main types of pooling layers: max pooling and average pooling.

Average Pooling: Calculate the average value for each patch on the feature map.

Max Pooling: Calculate the maximum value for each patch of the feature map

In most cases, max pooling performs better.

Dense layer Dense layer is a simple linear operation in which every input is connected to every output by a weight. Here, the dense layer will transform the reshaped output vector to the output layer which has the same number of neurons corresponding to the number of classes. All neurons will return probabilities of the input image for the respective class. Class with highest probability will be considered as output for that image.

Activation function: ReLU & Softmax

$$\text{ReLu}: RelU(x) = \max(0, x)$$

ReLU is very simple to calculate, as it involves only a comparison between its input and the value 0. It also has a derivative of either 0 or 1, depending on whether its input is respectively negative or not. ReLUs also prevent "vanishing gradient" problem, which is common when using sigmoid functions.

$$\text{Softmax}: \sigma(z_i) = \frac{exp(z_i)}{\sum_{j=1}^{K} exp(z_j)}$$

The softmax function is a generalization of the logistic function to multiple dimensions. It is used in multinomial logistic regression and is often used as the last activation function of a neural network to normalize the output of a network to a probability distribution over predicted output classes.

Backward propagation Backward propagation is the essence of neural network training. It is a method of fine-tuning the weights of a neural network based on the error rate obtained in the previous iteration. It computes the gradient of the loss function for a single weight by the chain rule. Here is the pseudocode which showing in the slide.

Algorithm 1 Backward Propagation

Input: training example (**x**,y)
Set $\mathbf{a}^{(0)} = \mathbf{x}$, and the Forward Propagation algorithm to compute $\mathbf{a}^{(l)}$ and $\mathbf{z}^{(l)}$ for $l = 1, 2, ..., H$
Compute $\tilde{\mathbf{a}}^{(H)} = l'(y, \mathbf{a}^{(H)})$
for $l = H - 1$ down to 0 **do**
  Compute $\tilde{z}_i^{(l+1)} = g'(z_i^{(l+1)}) \cdot \tilde{a}_i^{(l+1)}$ for $i = 1, 2, ..., k^{(l+1)}$
  Compute $\tilde{W}^{(l+1)} = \tilde{\mathbf{z}}^{(l+1)} \cdot \mathbf{a}^{(l)^T}$
  Compute $\tilde{\mathbf{a}}^{(l)} = W^{(l)^T} \tilde{\mathbf{z}}^{(l+1)}$
end for
Output $\tilde{W}^{(l)}$ for $l = 0, 1, ..., H - 1$

3.1.2 SVM

Soft margin SVM The objective of the support vector machine(SVM) algorithm is to find a hyperplane which has maximum margin. In non-separable cases, constraint is relaxed to yeild soft-margin SVM. Here, we replace constraint as the hinge loss, and set λ controls trade off between slack penalies and size of margin.

$$\arg\min_{\mathbf{w}\in\mathbb{R}^d} f_{GD}(\mathbf{w}) := \frac{\lambda}{2}||\mathbf{w}||_2^2 + \frac{1}{|S|}\sum_{(\mathbf{x},y)\in S}[1 - y < \mathbf{w}, \mathbf{x} >]_+$$

Implementation with gradient descent Gradient descent (GD) is an iterative optimisation algorithm used to find a local minimum of a given function. The cost function of soft margin SVM is a convex function and hence we compute gradient of it in order to update weights of function.

$$\begin{aligned}\mathbf{w}_{t+1} &:= \mathbf{w}_t - \eta_t(\lambda wt + \frac{1}{|S|}\sum_{(\mathbf{x},y)\in S: y<\mathbf{w}_t, x>\leq 1} y\mathbf{x}) \\ &= (1-\lambda\eta_t)\mathbf{w}_t + \eta_t\frac{1}{|S|}\sum_{(\mathbf{x},y)\in S: y<\mathbf{w}_t, x>\leq 1} y\mathbf{x}\end{aligned}$$

Multiclass SVM In most cases, SVM supports binary classification and splits samples into two classes. For multiclass classification, the same algorithm is applied after breaking down the multi-classification problem into multiple binary classification problems. The idea is to create classifier between each two classes. In our project, we used linear multiclass approach as predictor and utilized generalized hinge loss in our objective function.

$$\arg\min_{W\in\mathbb{R}^{d\times|\mathcal{Y}|}} f_{multi}(\mathbf{W}) := \frac{\lambda}{2}\sum_{y\in\mathcal{Y}}||\mathbf{w}_y||^2 + \frac{1}{m}\sum_{i=1}^{m} l^{MH}(\mathbf{W}, \mathbf{x}_t, y_t)$$

$$l^{MH}(\mathbf{W}, \mathbf{x}_t, y_t) = \max_{y'\in\mathcal{Y}}(\Delta(y', y_t) + < \mathbf{w}_{y'} - \mathbf{w}_y, \mathbf{x}_t >)$$

Multiclass SVM implemented with SGD Stochastic gradient descent (SGD) computes gradient using a single training sample, which could help less computationally expensive and reaches convengence much faster than batch gradient descent as shown before.

$$\arg\min_{W\in\mathbb{R}^{d\times|\mathcal{Y}|}} f_{multi-SGD_t}(\mathbf{W}) := \frac{\lambda}{2}\sum_{y\in\mathcal{Y}}||\mathbf{w}_y||^2 + l^{MH}(\mathbf{W}, \mathbf{x}_t, y_t)$$

We firstly applied multiclass SVM with SGD method for our project. Pseudo code of algorithm (Algorithm 2) shown as follow:

For each iteration, we shuffled all training samples. Firstly, generalized hinge loss for each example and the max prediction are calculated, which is used to compute stochastic subgradient of cost function. Then, we updated weights by using the subgradient. Finally, the loop stops until finishing all training samples.

Mini-batch implementation In SGD method, we used one sample at a time and did not apply vectorized implementation, which could slow down computations (Fig. 3). To solve this problem, we tried mini-batch GD method to update weights frequently.

Pseudo code of mini-batch SGD-SVM (Algorithm 3) shown as follow. In this method, we divided training samples into mini-batchs, and set the batch size as 4.

We also compared the performance between SGD and mini-batch GD implementation by using Mnits dataset (784 features). We can find from Table 1, especially from line 2 and line 5, SGD method takes almost the same time with mini-batch GD, and these two methods have familiar performance on accuracy also. For this reason, we decided not to implement mini-batch in our project.

Algorithm 2 Multiclass SVM with Stochastic Gradient Descent (SVM-SGD)

for $i = 1\,to\,T$ **do**
  if Finish one run on data set **then**
    Shuffle the data set
  end if
  $l_{mat}^{MH} \leftarrow (\Delta(y, y_t) + < \mathbf{w}_y - \mathbf{w}_{y_t}, \mathbf{x}_t >)\ \forall y \in \mathcal{Y}$ ▷ Calculate generalized hinge loss
  $y' \leftarrow \arg\max_{y' \in \mathcal{Y}} l_{mat}^{MH}$
  $W \leftarrow (1 - \eta\lambda)W$
  $\mathbf{w}_{y'} \leftarrow \mathbf{w}_{y'} + \eta\mathbf{x}_t$
  $\mathbf{w}_{y_t} \leftarrow \mathbf{w}_{y_t} - \eta\mathbf{x}_t$
end for

Table 1: Performance by Using mini-batch SGD or Not

Feature number	Method	Validation iteration	Training iteration	Accuracy
MNIST raw 784 features	SGD-SVM	1e5/λ ∼1e-2	1e6/22.84s	92.15%
			1.5e5/3.96s	91.45%
			1e4/0.98s	87.07%
	SGD-SVM-miniBatch (size 4)	1e5/λ ∼1e-2	1e6/219s	91.12%
			1e4/3.90s	91.14%

3.1.3 CNN-SVM

The hybrid CNN-SVM model is using the CNN as an automatic feature extractor and its last layer is replaced by the SVM as a classifier. The output probability of the CNN model is calculated by the softmax activation function. The input of the activation function is the linear combination of the outputs form the previous hidden layer with weights and a bias term. Here, the output values is meaningless except the CNN model itself. However, these values can be considered as meaningful features for other classifier like the SVM. Therefore, in the hybrid CNN-SVM model, the output values are treated as the input of the SVM. The number of features of the input to the SVM is equivalent to the length of the reshaped vector after the second pooling layer.

3.2 Evaluation of Performance on MNIST data set

3.2.1 Data preparation

For further comparison and analysis, we use down sampling to generate fewer features for MNIST data set, showing in Fig.4. The origin image size is 28-by-28, which contains 784 features for one

Algorithm 3 SGD-SVM with mini-batch

for $i = 1\,to\,T$ **do**
  if Finish one run on data set **then**
    Shuffle the data set
  end if
  Divide data set into mini-batches
  for Each mini-batch **do**
    for Each data in mini-batch **do**
      $l_{mat}^{MH} \leftarrow (\Delta(y, y_t) + < \mathbf{w}_y - \mathbf{w}_{y_t}, \mathbf{x}_t >)\ \forall y \in \mathcal{Y}$
      $y' \leftarrow \arg\max_{y' \in \mathcal{Y}} l_{mat}^{MH}$
      $\delta\mathbf{w}_{y'} \leftarrow \mathbf{x}_t$
      $\delta\mathbf{w}_{y_t} \leftarrow -\mathbf{x}_t$
    end for
    $\overline{\delta W} = \frac{1}{mini-batch\ size} \sum_{y', y_t in\ mini-batch} \delta\mathbf{w}_y$
    $W \leftarrow (1 - \eta\lambda)W + \eta\overline{\delta W}$
  end for
end for

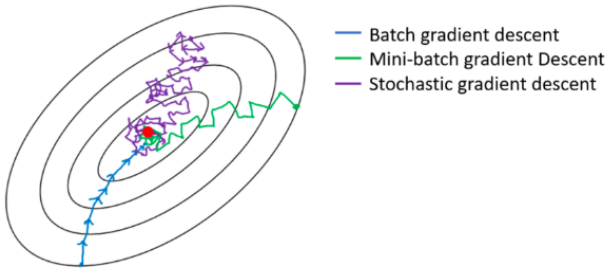

Figure 3: Convergence with different types of gradient descent

image. Here, we use down sampling to generate image size from 20-by-20 to 2-by-2 which contains fewer features than the origin image. The function we use for the down sampling in MATLAB is called imresize. The default method for down sampling in imresize function is bicubic interpolation. It means that the output pixel value is a weighted average of pixels in the nearest 4-by-4 neighborhood.

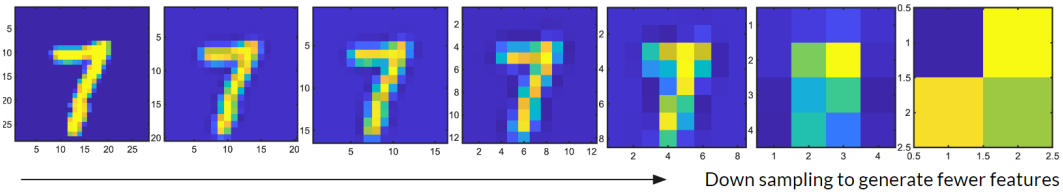

Figure 4: MNIST data down sampling

3.2.2 Training and results

As showing in the Fig.5 A, which is the performance by using the SGD-SVM method on MNIST, the accuracy is still at a high-level when the image size is down sampling to 20-by-20 and 16-by-16. Then, the accuracy begins to decrease when the number of features is lower than 12-by-12 and followed by a huge decreasing.

Fig.5 B shows the performance comparison on MNIST among SVM, CNN, and the hybrid CNN-SVM model. It shows that the performance of the hybrid CNN-SVM model is better than the performance by using single CNN model or single SVM model.

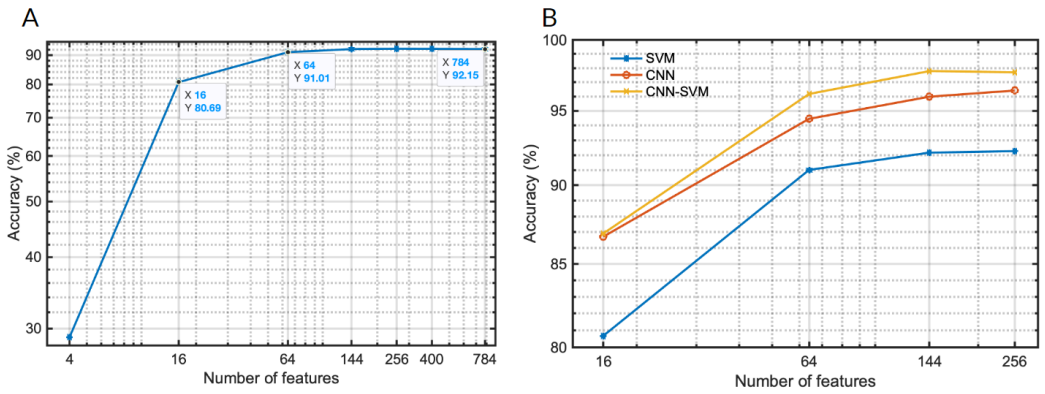

Figure 5: Performance comparison on MNIST dataset

Also, from the confusion matrix showing in the Fig.6, we can see that the performance of the hybrid CNN-SVM model, which has an accuracy of 0.977, is still better than the performance of the full

feature single SVM model, which has an accuracy of 0.9215, even if we use the down sampling features for the hybrid CNN-SVM model.

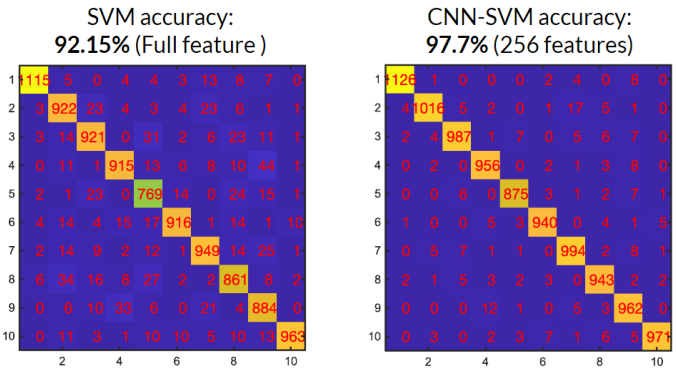

Figure 6: Confusion matrix of SVM and CNN-SVM on MNIST dataset

4 Application on biosamples

4.1 Data preparation

Image acquisition and cleaning To initial our application on biosamples, we collected some optical microscopy images of the cell sample to construct our data set for cell identity classification. The microscopy technique used here is stimulated Raman scattering microscopy, which maps the chemical distribution inside the organism. It avoids the interference from the transmission property of the biosamples. In this project, we used three kinds of bacterial cells, *B. producta* (BP), *B.thetaiotaomicron* (BT), *C.scindens* (CS) as our objects. With the purified culture of each species, the images collected from each sample are with know identities.

Some preprocessing was applied to improve the quality of the generated dataset and reduce the Bayes error of the dataset. Due to the property of the optical microscopy, the illumination on the image collected is not even. So the illumination background was generated by fitting the minimum intensity in every block of 30 by 30 pixels block using quadratic function. Then the raw data was divided by illumination to correct the intensity of the image. After illumination correction, the edge of the image was enhanced by calculating the gradient of the image. Then these edges were subtracted to help enhance the gap and increase the separation between the cells for better single cell segmentation.

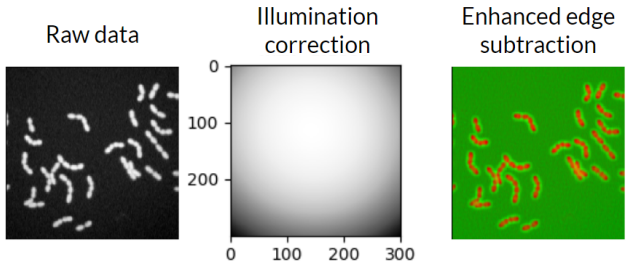

Figure 7: Biosample data preprocessing

Single cell segmentation For the single cell segmentation, we import the figure after processing to an open source software, *CellProfiler*, to generate the single cell segmentation masks. Then we applied these binary masks on the normalized gray scale images to generate single cell images with the size of 40 by 40 pixels.

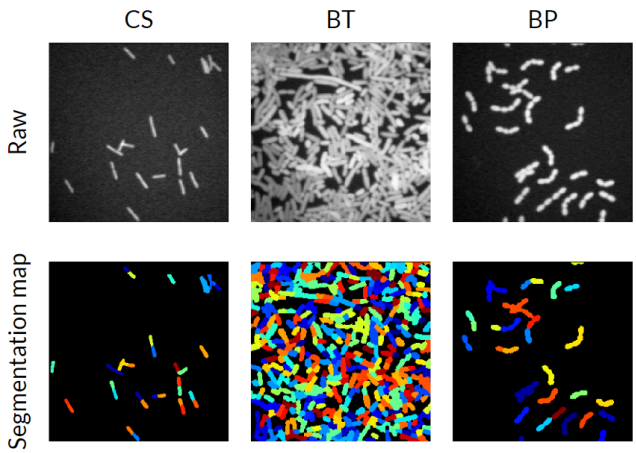

Figure 8: Single cell segmentation of three types of samples

Data augmentation Due to the imbalance in the number for the three classes (CS 200, BT 4000, BP 200), we did a simple data augmentation by rotating CS and BP cells with different degrees. And eventually, we generated a dataset with 12,761 samples, which was then separated into training set (5/7), validation set (1/7) and test set (1/7).

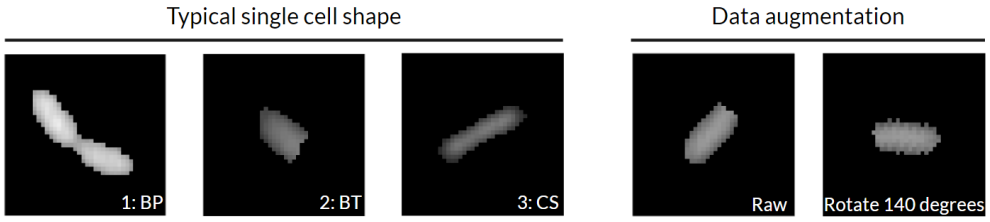

Figure 9: Single cell data as the input to machine learning model

4.2 Performance evaluation

To get a baseline and rough feeling of the features essential for classification, we still start testing using SVM with raw data and down sampling features. The performance of SGD-SVM is shown in Fig. 10A. After fine-tuning the regularizer on the validation set, we achieved the highest accuracy with SGD-SVM on the biosample is 82.61%. And the accuracy kept decreasing with the down sampling feature, and the accuracy showed a drastic drop from 256 features to 144 features, which means after the down sampling, the information for classification is insufficient.

After adjusting the mini-batch size in CNN, we tested CNN precision with the different architectures (second to last layer neuron number: 144 and 256). The output from that layer also generated output for SGD-SVM to test our hybrid model precision. The performance was compared in Fig. 10B and Fig. 11. Although there is a gap in the SVM performance using the down sampling features, CNN can still efficiently extract the features for accurate classification. And with hybrid model, the precision was further improved to 87%. The accuracy could be further improved by fine tuning the CNN architecture to adapted to the biosample data. Besides, segmentation which provides lower Bayes error may also be an impact factor.

5 Conclusions and Discussions

Our project applies a hybrid CNN-SVM model to the MNIST data set and the bio-sample data set. In the hybrid CNN-SVM model, the CNN is considered as a automatic feature extractor and the last

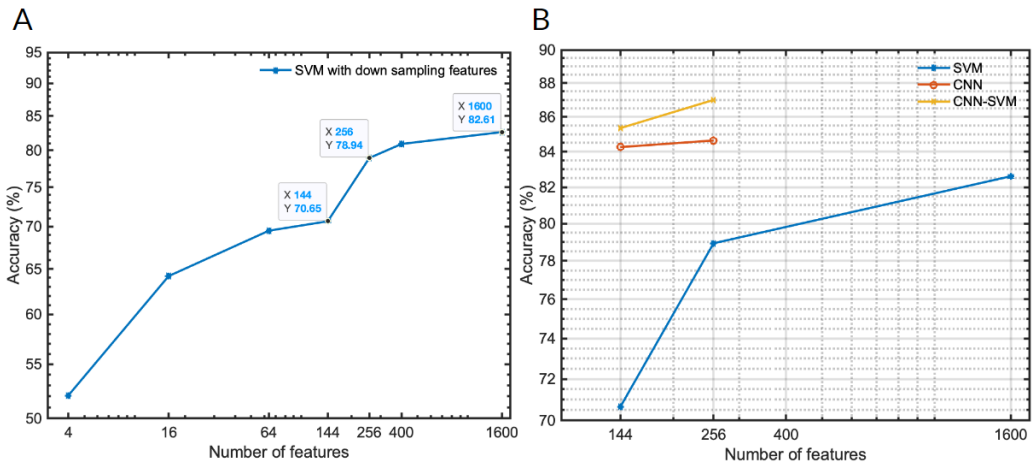

Figure 10: Performance comparison on biosample

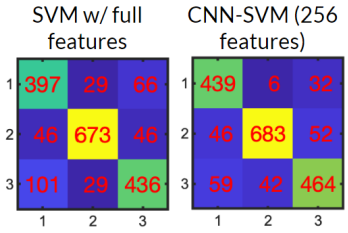

Figure 11: Confusion matrix of SVM and CNN-SVM on biosample

layer of the CNN is replaced by the SVM as a classifier. The CNN and the SVM are both famous and successful method in the classification, and the hybrid CNN-SVM model takes the advantages of them to have a better performance.

Our results show that the hybrid CNN-SVM model has a better performance over the single CNN model or the single SVM model. The hybrid CNN-SVM model can achieve a higher accuracy than the single CNN or SVM model. Also, the hybrid CNN-SVM model can still have a higher accuracy using a proper number of down sampling features comparing to the full-feature CNN or SVM model.

Our next step is trying to further improve the accuracy while having a relatively low increasing on the complexity. For example, the CNN part in the hybrid CNN-SVM model could be better designed by choosing a proper size of the input layer or number of layers. Also, more powerful kernels could be applied to the SVM part to achieve a better accuracy.

References

[1] X.-X. Niu and C. Y. Suen, "A novel hybrid cnn–svm classifier for recognizing handwritten digits," *Pattern Recognition*, vol. 45, no. 4, pp. 1318–1325, 2012.

[2] Darmatasia and M. I. Fanany, "Handwriting recognition on form document using convolutional neural network and support vector machines (cnn-svm)," in *2017 5th International Conference on Information and Communication Technology (ICoIC7)*, pp. 1–6, 2017.