Published in final edited form as:

Genet Epidemiol. 2019 April ; 43(3): 330–341. doi:10.1002/gepi.22182.

Application of deep convolutional neural networks in classification of protein subcellular localization with microscopy images

Mengli Xiao¹, Xiaotong Shen², and Wei Pan^{1,*}

¹Division of Biostatistics, University of Minnesota, Minneapolis, MN 55455, USA

²School of Statistics, University of Minnesota, Minneapolis, MN 55455, USA

Abstract

Single cell microscopy images analysis has proved invaluable in protein subcellular localization for inferring gene/protein function. Fluorescent-tagged proteins across cellular compartments are tracked and imaged in response to genetic or environmental perturbations. With a large number of images generated by high-content microscopy while manual labeling is both labor-intensive and error-prone, machine learning offers a viable alternative for automatic labeling of subcellular localizations. On the other hand, in recent years applications of deep learning methods to large datasets in natural images and other domains have become quite successful. An appeal of deep learning methods is that they can learn salient features from complicated data with little data preprocessing. For such purposes, we applied several representative types of deep Convolutional Neural Networks (CNNs) and two popular ensemble methods, random forests and gradient boosting, to predict protein subcellular localization with a moderately large cell image dataset. We show a consistently better predictive performance of CNNs over the two ensemble methods. We also demonstrate the use of CNNs for feature extraction. In the end, we share our computer code and pre-trained models to facilitate CNN's applications in genetics and computational biology.

Keywords

CNNs; Deep learning; Feature extraction; Gradient boosting; Random forests

Introduction

The spatiotemporal variation of a protein location can reflect the dynamics of gene-environment interactions (Chong et al., 2015). One common pipeline to analyze this protein cellular phenotype information is the high-content microscopy image analysis. In the high-content microscopy images, proteins are fluorescently labeled to track their locations within a cell. Cell protein location is also useful to predict the protein functions. In a model organism called *Saccharomyces cerevisiae* budding yeast, proteins of interest are tagged with fluorescence through a genetic technique. Changes of the protein locations in response to environmental and genetic perturbations are recorded via the fluorescent signals under

*Corresponding author. panxx014@umn.edu.

microscopy yeast cell images. Several high-content yeast cell images were collected in existing databases using an automated image analysis system (Koh et al., 2015). A labeled cell image dataset contains cell images with their subcellular localizations as the class labels. Typically, the labels are inferred from the shape of fluorescent signals by human experts. However, the manual labeling process for a large number of yeast cell fluorescent images is error-prone. In addition, since thousands of images could be generated in one day to measure the dynamic cellular process, it is also labor-intensive. Discriminating subtle differences manually among thousands of fluorescent images is challenging. Machine learning methods offer a solution to the above problem if the labels can be accurately predicted by learning from a training sample of labeled images. Chong et al. (2015) utilized an ensemble of 60 support vector machines (SVM) to classify single yeast cell subcellular location. This ensemble of classifiers (ensLOC) achieved high precisions and recall (> 70%). However, the image segmentation and dimension reduction procedure before training the classifier required a significant amount of preprocessing work for new datasets.

On the other hand, Convolutional Neural Networks (CNNs) are shining in their applications in auto-driving cars, game playing, and image and speech recognition (LeCun, Bengio, & Hinton, 2015). Since 2012, CNN has been a top performer in the annual ImageNet competitions (Deng et al., 2009; Krizhevsky, Sutskever, & Hinton, 2012); ImageNet consists of over 14,000,000 real-world object images. In addition to their great predictive performance, CNNs are good at automatic feature extraction (Chen, Jiang, Li, Jia, & Ghamisi, 2016). With CNNs' increasing popularity in computer vision and other applications, efficient implementation systems have been developed to take advantage of highly parallel operations under graphical processing units (GPUs). Importantly, many user-friendly software frameworks have become available, such as Tensorflow and Keras (Abadi et al., 2016; Chollet, 2015), enabling easier implementations for a wide range of applications.

Given CNNs' growing importance, CNNs have been applied and compared with ensembles of SVMs or random forests in predicting protein subcellular localizations (Kraus et al., 2017; Pärnamaa & Parts, 2017). Although the previous authors concluded that CNNs had a higher accuracy than ensembles of SVM and random forests, there is a lack of an extensive comparison between CNNs and other machine learning methods. Except for CNNs, classifiers in their experiments were only applied to vectorized or preprocessed images (Kraus et al., 2017; Pärnamaa & Parts, 2017). Also, in the rapidly-developing field of CNNs, there are emerging new CNN architectures with improved performance; it would be worthwhile to apply them to cell image analysis. An extensive exploration on the impact of CNN architectures would serve as a reference for future researchers to pick a specific CNN architecture.

Here, we implemented various architectures of CNNs and conducted extensive comparisons on their performance and with two popular ensemble machine learning methods in image analysis for protein subcellular localization. Specifically, we discuss the convincing advantage of CNNs in terms of classification accuracy over random forests and gradient boosting. We also demonstrate the use of CNNs as feature extractors to improve other

machine learning methods. Finally, we offer our freely accessible computer code and pre-trained models to serve future endeavors.

Methods

Convolutional Neural Networks (CNNs)

What are CNNs—CNNs is a type of Neural Networks (DNNs) tailored for image analysis. A deep NN (DNN) or a deep CNN simply means that there are many (thus deep) layers of neurons extracting from lower- to high-level features. Input data is usually in the format of arrays/tensors such as 2-dimensional images with color channels as the 3rd-dimension. A typical architecture of CNNs is a stack of convolutional and pooling layers followed by one or few fully-connected layers in the end (Figure 2); a deep CNN (or NN) simply many, and thus deep, layers. With intermediate nonlinearity functions such as the Rectifier Linear Units (ReLU) formed as $f(y) = \max(0, y)$ between layers, the stacked architecture allows the CNNs to learn from complicated data with multiple levels of representation (LeCun et al., 2015). The convolution step is illustrated in Figure 3a. The red 2*2 square is a kernel/filter over a 4*4 image (with only one channel). The numbers in the kernel are the weights or the parameters of the kernel, which are shared across the image. The kernel slides through the 4*4 image by computing the dot products between the kernel weights and the pixel values of its covered sub-image. For example, the 2*2 kernel with weights {1,2,3,4} computes the dot product with the first 2*2 grid of pixels {1,2,5,6} in the 4*4 image, resulting in the value $44=1*1+2*2+5*3+6*4$, which is then input to a ReLU function to obtain an output, which consists of a so-called feature map. If the stride (as a hyperparameter) is set to be 1 in the model specification, then the kernel moves through the image horizontally then vertically by a step-size of 1 pixel to compute the dot product with its covered sub-image, e.g., the next 2*2 grid of pixels {2,3,6,7}. Then, by following the same rule, the kernel computes the dot product sequentially with the 2*2 image pixel grid {3,4,7,8}, {5,6,9,10}, ..., {11,12,15,16}. Thus, the resulting feature map has 3*3 pixels, i.e., $(\text{Input size } 4 - \text{kernel size } 2 + \text{padding } 0)/1 + 1 = 3$. The dot product we explained here is a special case of convolution, thus the name. Images often have more than one channel then the same convolution operation with a 3-D kernel applies to both the spatial and the channel dimensions at the same time. The kernel weights are the major part of the unknown parameters that are estimated or learned through model training by minimizing a loss function often using a gradient descent algorithm called back-propagation (Rumelhart, Hinton, & Williams, 1986).

Four key properties contribute to CNNs' success in image data: shared weights, local connections, pooling and depth of layers (LeCun et al., 2015). The number of parameters is reduced by using shared weights and local connections (Angermueller, Pärnamaa, Parts, & Stegle, 2016). In Figure 3a, the kernel weights are shared across all 2*2 grid of pixels in that image channel. Local connection means each neuron in the feature map is only connected to the neuron of the previous layer by kernel's convolution operation. Each pixel in the output feature map is only associated with a certain 2*2 image pixel grid (Figure 3a). In contrast to fully-connected DNNs, those two properties enable CNNs to have a smaller number of parameters and be less likely to overfit. Pooling enables the spatially invariant detection of image features. For example, max-pooling keeps only the most salient features or biggest

values of pixels (Figure 3b). With the use of a non-linear function like ReLU, a deep CNN is able to learn some very complex structure from the data. By taking advantage of large training datasets and speeding-up by GPUs, CNNs have been gaining increasing popularity in imaging recognition; its performance has been shown to be comparable to and then even better than human eyes (He, Zhang, Ren, & Sun, 2015).

Baseline CNN—VGG-19, a 19-layer Visual Geometry Group CNN model (Simonyan & Zisserman, 2014), was trained on ImageNet Large Scale Visual Recognition Challenge dataset (ILSVRC) consisting of natural objects, aircraft, etc. (Deng et al., 2009). It achieved great success as a pioneering deep CNN and has since become popular in practice (Simonyan & Zisserman, 2014). There were several implementations of some 11-layer CNNs similar to VGG-19 for the subcellular localization problem (Kraus et al., 2017; Pärnamaa & Parts, 2017). As a benchmark, we trained an 11-layer VGG-type model from the scratch in Keras (Chollet, 2015). Our model specification followed the DeepYeast model of Pärnamaa & Parts (2017) (Figure 2). The network structure is similar to the first few layers of the pioneering VGG-19 (Table 2); note the unusually large numbers of the parameters (i.e. weights in kernels) to be estimated in the two models. We referred this 11-layer model as VGG-type CNN and treated it as our baseline CNN.

Implementation—Except for the last fully-connected layer, every layer was followed by batch normalization (Ioffe & Szegedy, 2015). The softmax (a multinomial logit) function was used in the last layer to generate an estimated probability in each class for an input. We initialized the weights (i.e. the parameters in kernels) from Glorot-normal distribution (i.e., a truncated normal distribution centered on 0 with

$\sigma^2 = 2/(\# \text{ of input kernels} + \# \text{ of output kernels})$ (Glorot & Bengio, 2010). The Stochastic Gradient Descent (SGD) (Bottou, 2010) with momentum 0.9 was used to minimize the cross-entropy loss. The initial learning rate was 0.1 and was reduced by half after every 16,250 iterations. The batch size was 100. An L2 norm weight decay with $\lambda = 0.0005$ and a dropout rate of 0.5 were applied to reduce the overfitting. The model was trained for 195,000 iterations. We saved and evaluated the performance of the model at an interval of every 32,500 iterations using a separate validation dataset. The best performing model was selected for following evaluations using another separate test dataset. The training took 6 hours on a server with a single NVIDIA GEFORCE GTX 1080 Ti GPU.

Residual Neural Networks

By observing difficulties in training and performance degradation of very deep CNNs, He, Zhang, Ren, and Sun (2016a) proposed a new architecture, called Residual Neural Network (ResNet), with a shortcut connection between some layers and replacing fully-connected layers with convolution layer blocks to ease the optimization (Figure 5), making it possible to train very deep CNNs such as a 1001-layered CNN (He, Zhang, Ren, & Sun, 2016b). The convolutional block design in ResNets also reduces the number of parameters as compared to a VGG-type CNN (Table 3). We therefore analyzed the subcellular image data with the state-of-the-art ResNet. In addition, since there have been some further extensions to ResNet with changing depth and width, we also extensively studied ResNet with different depth and width to better illustrate CNN's performance with possibly deeper and wider CNNs.

Implementation—We implemented the ResNets and their corresponding VGG-type CNNs for comparison (colored rows in Table 4). All the training parameters including learning rate, the training iteration, regularization, weight initialization, and the optimizer follow the same protocol as before. Furthermore, since the identity shortcut connection was found to have the lowest test error among all types of shortcut connections (He et al., 2016b), we implemented a 18- and a 50- layered models (Table 3) with only the identity shortcut, in contrast to the ResNet 18 and ResNet 50 employing the projection shortcut (1*1 convolutions to match the dimensions). To avoid the dimension mismatch (different number of kernels/filters) between two connected layers, we made the numbers of kernels/filters the same for all convolutional layers and named it the straightened ResNets. In addition, to examine the influence of augmented dataset and training optimizer, we trained the straightened ResNet with augmented dataset (in which images were augmented by random rotation, shifting, and reflection) and a newer and better SGD algorithm called Adam optimizer (Kingma & Ba, 2014).

Since the Wide Residual Network (Zagoruyko & Komodakis, 2016) shows a promising performance over the original or thin ResNet by increasing the width of the convolutional layer (the number of kernels/filters per layer), we implemented the Wide Residual Network with a widening factor 2 and depth 40 (W40–2) that was computationally feasible on our server with only one GPU.

We compared the ResNet model structures to the baseline 11-layer CNN model (DeepYeast) and our straightened versions of ResNet in Table 3.

Other state-of-the-art CNNs

Despite the success of both VGG-type and Residual CNNs, achieving an optimal performance can be tricky. A higher performance is most likely associated with an increasing network size in both depth and width at the expense of dramatically increasing computational cost and larger training data. Thus, a new structure called Inception was proposed to save computing cost with increasing depth and width in CNNs through using 1*1 convolutions (Szegedy et al., 2015). The most recent Inception model called Inception-ResNet V2 combines the advantages of both the ResNet and the Inception (Szegedy, Ioffe, Vanhoucke, & Alemi, 2017), which is illustrated in Figure S1.

On the other hand, although CNNs detect spatially invariant features across the whole image, it generally requires a large number of labeled images and more balanced labels/classes for training, which are crucial for CNNs to learn full variations of inputs (Jiménez-Sánchez, Albarqouni, & Mateus, 2018). Sabour, Frosst, and Hinton (2017) introduced a new architecture called Capsule Network (CapsNet) to recognize the different viewpoints or variations of the same input object. Instead of yielding a scalar before the ReLU activation function for each subimage, CapsNet groups a set of kernels together to output a vector called capsule, learning more comprehensive properties than the brute-force scalar (Figure S2). Each element of the vector represents one particular type of object information such as relative position in the image.

Since our cell images dataset contains a moderate number of training samples and is unbalanced, we implemented the Inception-ResNet V2 and CapsNet on our cell image dataset to provide an extensive evaluation on the state-of-the-art CNNs in addition to ResNets. We followed the implementation details from Szegedy et al. (2017) and Sabour et al. (2017) except for several architecture parameters adapted to our image dimension and dataset size. For example, as a compromise between network complexity and computation power in our machine, we changed the number of kernels in the Inception-ResNets to be 64 if it is larger than 64.

Traditional machine learning methods

To compare with CNNs, we implemented two tree ensemble methods known for very good classification performance in practice: random forests (Breiman, 2001) and extreme gradient boosting (XGBoost) (Chen & Guestrin, 2016). For further comparisons, we also implemented several standard classifiers: linear discriminant analysis, K-nearest neighbor, linear support vector machine and lasso logistic regression (Cover & Hart, 1967; Hearst, Dumais, Osuna, Platt, & Scholkopf, 1998; Lachenbruch & Goldstein, 1979; Tibshirani, 1996). Tuning parameters, such as the numbers of trees in random forest or boosting, were chosen based on the validation dataset, instead of cross-validation, because the DeepYeast validation dataset is large enough (12,500 images). All image data needs be vectorized before use. All classifiers except lasso logistic regression random were implemented using Python Scikit-learn (Pedregosa et al., 2011) while lasso logistic regression was in R glmnet package (Friedman, Hastie, & Tibshirani, 2009).

Feature Extraction and Transfer Learning

Feature extraction—A possible advantage of CNN is feature representation. A trained CNN is also used as a feature extractor or as an initializer for other classifiers to obtain better performance efficiently (Sharif Razavian, Azizpour, Sullivan, & Carlsson, 2014). We hypothesized that its feature extraction could benefit the analysis of other statistical methods. Leveraging CNN's feature extraction, we replaced the last fully-connected layer of the DeepYeast with random forests or XGBoost. Then we compared the performance against applying those two classifiers from scratch.

Although it is in general difficult to interpret specific features extracted in each layer of a CNN, efforts have been made with some progress, such as applying gradient ascent to generate input that activates a kernel, visualizing activation heatmaps after occluding some objects (Yosinski, Clune, Nguyen, Fuchs, & Lipson, 2015; Zeiler & Fergus, 2014). Here, for illustration, we took the simple (and common) approach to visualize the activation in the first and last convolution layers of a CNN.

Transfer learning with pre-trained VGG-19 on ImageNet—We also chose VGG-19 as the feature extractor as its convolutional layer alignment is similar to the DeepYeast model (Table 2). As VGG-19 was trained on ImageNet, the re-use of it in our data is transfer learning (Pan & Yang, 2010; Simonyan & Zisserman, 2014). We obtained the pre-trained VGG-19 model (available from the Keras library and trained on ImageNet) to extract image features from DeepYeast training data. Extracted features by VGG-19's convolutional parts

were fed into subsequent classifiers including fully connected layers, random forests and gradient boosting. For fully connected classifiers, we trained for 65,000 iterations and use validation dataset to select model iteration with the best validation accuracy.

Transfer learning to different cell images—To show a higher efficiency of applying CNN in a new dataset, we deployed the code from Kraus et al. (2017) and applied a pre-trained 11-layered CNN to another dataset from Yofe et al. (2016), in which the protein of interest was dyed in red fluorescence, each cell image was not much preprocessed, and there were only 11 localization classes, as detailed in the next subsection. To eliminate the influence of imbalanced classes on the assessment of transfer learning performance, the network was evaluated with an increasing equal number of samples per class (1, 3, 5, 10, 25, 50, 100, 150, 200, 250, and 500). For each number of samples, we bootstrapped 15 times from the original dataset and took the average of the performance in the final evaluation. We fine-tuned the weights from the DeepLoc layers (Kraus et al., 2017) while training the final fully-connected (softmax) layer from the scratch. The experiment was conducted in Tensorflow and the procedure followed the methods in Kraus et al. (2017).

Data sources

The data consists of segmented high-throughput microscopy image data from Pärnamaa and Parts (2017). Their data before segmentation or cropping comes from Chong et al. (2015), which is stored in the CYCLOPs database (Koh et al., 2015). They cropped 64*64 pixel patches centered on a single cell from microscopy pictures, thus there might be some other surrounding cells with the same fluorescent patterns. Specific fluorescent patterns of cells inform the subcellular locations (Figure 1a). Red and green channels in an image mark the cell body as background and track the protein location respectively. The dataset consists of 65,000 training, 12,500 validation and 12,500 test single cell microscopy images with imbalanced counts from 12 localization classes (cell periphery, cytoplasm, endosome, endoplasmic reticulum, Golgi, mitochondrion, nuclear periphery, nucleolus, nucleus, peroxisome, spindle pole, and vacuole) (Table 1). To verify the performance of VGG-type CNNs, we reproduced the CNN on a similar dataset but each microscopy image was not segmented into one cell from Kraus et al. (2017), who also processed their images from Chong et al. (2015). This similar dataset includes two more quality control classes, divides class “spindle pole” into two classes and separates class “vacuole membrane” into “vacuole” and “vacuole membrane”, resulting in 19 localization classes. The training, validation and test datasets include 21,882, 4,916 and 4,224 cell images respectively. Finally, CNN’s transfer learning efficiency was demonstrated in a substantially different cell image dataset (Figure 1b). The dataset was provided by Kraus et al. (2017) and taken from Yofe et al. (2016), in which each protein of interest was dyed in red fluorescence, instead of green in the other dataset. Further, the dataset was not most suitable for automated analysis with some clustered and overlapping cells in many images (Kraus et al., 2017). The new dataset had only 11 localization classes (ER, Bud, Bud Neck, Cell Periphery, Cytosol, Mitochondria, Nuclear Periphery, Nucleus, Punctate, Vacuole, Vacuole Membrane).

Code availability

To facilitate the applications of CNNs in genetics and computational biology, we offer our computer code and fitted models for public download at <https://github.com/menglix/CNNsCellImages>.

Results

CNNs consistently outperformed other classifiers on raw image data

In the presence of imbalanced class sizes in the data (Table 1), all VGG-type CNNs along with variations of ResNet achieved a consistently higher predictive accuracy (correct prediction rates ranging from 0.819 to 0.891 in Table 4) than other machine learning methods (0.228–0.679 in Table 4) on raw image data (only subtracting the training image pixel values by their sample mean). The classification accuracy was measured on the test data consisting of 12,500 images. The VGG-type CNN, with a typical CNN structure, achieved a higher classification accuracy of 0.851 than all other traditional classifiers, among which the top performance at accuracy 0.596 and 0.679 were obtained from random forests and XGBoost respectively (Table 4).

To assess the robustness of CNNs, we perturbed the training data by randomly assigning (incorrect) class labels to 5% or 10% of training images in each class category. The VGG-type CNNs still achieved high test accuracy at 0.828 and 0.801 respectively.

It was confirmed that training CNNs could be time-consuming even with a GPU server: it took 6 hours for 11-layered VGG-type CNNs, compared to 2 or 10 h on a CPU server for the two ensemble methods.

ResNets boosted both training speed and accuracy

Except for few ResNet structures with a higher depth (Res50) or width (Wide ResNet), the majority of ResNets were faster to train than both the random forest and XGBoost (Table 4), but this should be interpreted with caution because the former were trained on a server with a GPU while the latter (in Python Scikit-learn) on a standard CPU server. Furthermore, those ResNets not only improved the accuracy over the VGG-type CNN counterparts regardless of the depth (rows with the same color are the ResNet and its counterparts differing only in shortcuts in Table 4), but also achieved a higher accuracy than all other machine learning methods (0.843–0.891 vs. 0.679) with typically shorter training time (e.g., 1.75 h vs. 2 h).

CNNs were robust with their hyperparameters

The high performance of CNNs (with accuracy > 0.800 and training time < 6 h) was maintained with their structure/hyperparameter variations. From Table 4, none of the CNNs had a predictive accuracy rate below 0.800. In terms of training time, most CNNs had less than 6 h training time to accomplish an accuracy of greater than 0.800. CNN's structure variations still yielded higher performance over the ensemble methods (all other machine learning methods in our experiment and ensembles of SVM in Chong et al. (2015)), even though there were structural/hyperparameter variations such as different building blocks in ResNets (Table 3). We also noticed that a better optimizer could improve the classification

accuracy of CNNs from 0.876 (Res18 - SGD optimizer) to 0.891 (Res18 - Adam optimizer) for our problem (Table 4). On the other hand, little improvement on the test accuracy of our Res18 with identity shortcut was observed through the off-the-shelf image data augmentation in Keras such as image rotation and reflection. In contrast, DeepYeast model was slightly improved with the same data augmentation (0.874 vs. 0.851 in Table 4). As a side note, a more sophisticated data augmentation procedure by using random cropping and normalizing the image pixels to be in the range [0,1] (Kraus et al., 2017) improved the DeepYeast CNN's test accuracy to about 0.9.

Did wider and deeper CNNs perform better?

It is widely reported that CNNs' performance improved through a deeper and wider network (He et al., 2016a; Zagoruyko & Komodakis, 2016). In our single cell image analysis problem, we noticed that wider and deeper CNNs did not necessarily outperform shallower and thinner CNNs, perhaps due to the only moderately large sample size with imbalanced categories. According to Table 4, performance was worse with an increased depth of layers in the VGG-type CNNs. The test accuracy of VGG-type CNN dropped from 0.845 to 0.819 when the depth of the layers increased from 18 to 50. Moreover, the Wide ResNet with widening factor 2, with the same design of convolutional building block as in Res18 (except the number of filters is larger hence is wider) (Table 3), achieved a lower accuracy (0.853) than the thinner Res18 (0.871). However, the ResNet exhibited an improved accuracy (from 0.853 to 0.886) as the network went deeper from 18 to 50 layers. This is consistent to He et al. (2016), which found that deeper layered ResNets achieved better results than the shallower ones. Interestingly, the shortcut structure showed more benefits with a 50-layer model as the increase of accuracy from the VGG-type CNN (0.67=0.886–0.819; red rows in Table 4) was higher in Res50 than that of the 18-layer model (0.08=0.853–0.845; blue rows in Table 4).

In addition, two state-of-the-art CNN architectures, Inception-ResNet V2 and CapsNet with the former being large in both the depth and width while the other large in only the width, were found to perform slightly worse than both VGG-type and ResNet (0.815 and 0.826 vs. > 0.85 in Table 4). It might suggest that the numbers of parameters in the two models were too large to strike an effective balance between the model complexity and the training sample size.

Random forests and XGBoost gained from CNNs' feature extraction

Compared to applying the two machine learning classifier alone, i.e., random forest and XGBoost from the scratch, the test accuracy was improved by connecting them to both the DeepYeast and VGG-19 feature extractors. After extracting features from the image input by DeepYeast, the test accuracy increased from 0.596 to 0.850. Besides the gain in accuracy, the training was also faster with only 512 extracted features out of the original 12288 (=64*64*3) features. On the other hand, although ImageNet dataset is very different from the cell microscopy images, both training speed and accuracy were enhanced by VGG-19 network's feature extraction. It achieved descent test accuracy of 0.660 in random forest and 0.730 in XGBoost (Table 4). Preceding the fully-connected classifier, the VGG-19 feature extractor helped it take only 3 minutes to train while achieving an accuracy of greater than

0.7; in contrast, training a VGG-type CNN from the scratch required at least 1.58 hours on our server with a GPU (Table 4). To illustrate the features extracted by CNNs, we visualized the activations in the first and last convolution layers of DeepYeast in Figures S3 and S4 respectively.

CNN was generalizable to different cell images

We demonstrate that a pre-trained CNN could be fine-tuned and transferred to cell images to achieve a descent classification accuracy of 0.744 on different images with 100 labeled samples per class (average of the diagonal elements in Figure 5b). The images were not segmented with overlapping cell bodies, which were not designed for automated image analysis. In addition, red fluorescence was used to mark the protein of interest, instead of green fluorescence used in our main dataset (Figure 1b). There is a clear advantage of transfer learning with respect to the test accuracy when the number of labeled training samples per class was small (Figure 5a). According to the confusion matrix, when the sample size per class was only 100, we found that for 7 out of 11 localization classes the test accuracy exceeded 0.7 by transfer learning (Figure 5b), while random forests obtained test accuracy of 0.169 on this dataset (probably due to that the images were not as well preprocessed for analysis as the main dataset).

Conclusion

In the context of predicting protein subcellular localization categories on cell images, we presented extensive comparisons of classification performance between several types of CNNs and other machine learning methods. We demonstrated that CNNs, with little feature preprocessing on input images, always outperformed other methods on moderately large cell image dataset with imbalanced categories (0.819–0.891 vs. 0.228–0.679). Among all the CNNs, we showed that ResNets not only obtained the highest predictive power but also took the least amount of training time among all the methods tested by leveraging a GPU. With changes of structures or hyperparameters, CNNs were capable of maintaining good performance (classification accuracy > 0.800). In addition, our experiment suggests that, rather than relying on domain expertise or special software for feature preprocessing, other machine learning methods could benefit from CNN's feature extraction. Finally, since CNNs show promising performance in complex data with strong stationary, local and rich features, such as in biomedical images and DNA sequences, we expect more applications of CNNs to these types of data in the future. We share our computer code and pre-trained CNN models to facilitate such applications.

Supplementary Material

Refer to Web version on PubMed Central for supplementary material.

Acknowledgement

We thank the reviewers for their helpful comments and suggestions. This research was supported by NIH grants R21AG057038, R01HL116720, R01GM113250 and R01HL105397 and R01GM126002, and NSF grants.

References

- Abadi M, Barham P, Chen J, Chen Z, Davis A, Dean J, Isard M. (2016) Tensorflow: A system for large-scale machine learning. OSDI. pp. 265–283.
- Angermueller C, Pärnamaa T, Parts L, & Stegle O (2016) Deep learning for computational biology. *Molecular Systems Biology*, 12, 878. [PubMed: 27474269]
- Bottou L (2010) Large-scale machine learning with stochastic gradient descent. *Proceedings of compstat'2010 Springer*, pp. 177–186.
- Breiman L (2001) Random forests. *Machine Learning*, 45, 5–32.
- Chen T, & Guestrin C (2016) Xgboost: A scalable tree boosting system. *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining ACM*, pp. 785–794.
- Chen Y, Jiang H, Li C, Jia X, & Ghamisi P (2016) Deep feature extraction and classification of hyperspectral images based on convolutional neural networks. *IEEE Transactions on Geoscience and Remote Sensing*, 54, 6232–6251.
- Chollet F (2015) Keras.
- Chong YT, Koh JL, Friesen H, Duffy SK, Cox MJ, Moses A, Andrews B. (2015) Yeast proteome dynamics from single cell imaging and automated analysis. *Cell*, 161, 1413–1424. [PubMed: 26046442]
- Cover T, & Hart P (1967) Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13, 21–27.
- Deng J, Dong W, Socher R, Li L-J, Li K, & Fei-Fei L (2009) Imagenet: A large-scale hierarchical image database. *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on Ieee*, pp. 248–255.
- Friedman J, Hastie T, & Tibshirani R (2009) Glmnet: Lasso and elastic-net regularized generalized linear models. R package version, 1.
- Glorot X, & Bengio Y (2010) Understanding the difficulty of training deep feedforward neural networks. *Proceedings of the thirteenth international conference on artificial intelligence and statistics* pp. 249–256.
- He K, Zhang X, Ren S, & Sun J (2015) Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *Proceedings of the IEEE international conference on computer vision* pp. 1026–1034.
- He K, Zhang X, Ren S, & Sun J (2016a) Deep residual learning for image recognition. *Proceedings of the IEEE conference on computer vision and pattern recognition* pp. 770–778.
- He K, Zhang X, Ren S, & Sun J (2016b) Identity mappings in deep residual networks. *European conference on computer vision Springer*, pp. 630–645.
- Hearst MA, Dumais ST, Osuna E, Platt J, & Scholkopf B (1998) Support vector machines. *IEEE Intelligent Systems and Their Applications*, 13, 18–28.
- Ioffe S, & Szegedy C (2015) Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*.
- Jiménez-Sánchez A, Albarqouni S, & Mateus D (2018) Capsule networks against medical imaging data challenges. *arXiv preprint arXiv:1807.07559*.
- Kingma DP, & Ba J (2014) Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.
- Koh JL, Chong YT, Friesen H, Moses A, Boone C, Andrews BJ, & Moffat J (2015) Cyclops: A comprehensive database constructed from automated analysis of protein abundance and subcellular localization patterns in *saccharomyces cerevisiae*. *G3: Genes, Genomes, Genetics*, g3 115.017830.
- Kraus OZ, Grys BT, Ba J, Chong Y, Frey BJ, Boone C, & Andrews BJ (2017) Automated analysis of high-content microscopy data with deep learning. *Molecular Systems Biology*, 13, 924. [PubMed: 28420678]
- Krizhevsky A, Sutskever I, & Hinton GE (2012) Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*. pp. 1097–1105.
- Lachenbruch PA, & Goldstein M (1979) Discriminant analysis. *Biometrics*, 69–85.
- LeCun Y, Bengio Y, & Hinton G (2015) Deep learning. *Nature*, 521, 436. [PubMed: 26017442]

- Pan SJ, & Yang Q (2010) A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22, 1345–1359.
- Pärnamäa T, & Parts L (2017) Accurate classification of protein subcellular localization from high-throughput microscopy images using deep learning. *G3: Genes, Genomes, Genetics*, 7, 1385–1392. [PubMed: 28391243]
- Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Dubourg V. (2011) Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12, 2825–2830.
- Rumelhart DE, Hinton GE, & Williams RJ (1986) Learning representations by back-propagating errors. *Nature*, 323, 533.
- Sabour S, Frosst N, & Hinton GE (2017) Dynamic routing between capsules. *Advances in Neural Information Processing Systems*. pp. 3856–3866.
- Sharif Razavian A, Azizpour H, Sullivan J, & Carlsson S (2014) Cnn features off-the-shelf: An astounding baseline for recognition. *Proceedings of the IEEE conference on computer vision and pattern recognition workshops* pp. 806–813.
- Simonyan K, & Zisserman A (2014) Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.
- Szegedy C, Ioffe S, Vanhoucke V, & Alemi AA (2017) Inception-v4, inception-resnet and the impact of residual connections on learning. *AAAI*. pp. 12.
- Szegedy C, Liu W, Jia Y, Sermanet P, Reed S, Anguelov D, Rabinovich A (2015) Going deeper with convolutions. *Proceedings of the IEEE conference on computer vision and pattern recognition* pp. 1–9.
- Tibshirani R (1996) Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, 267–288.
- Yofe I, Weill U, Meurer M, Chuartzman S, Zalckvar E, Goldman O, Knop M. (2016) One library to make them all: Streamlining the creation of yeast libraries via a swap-tag strategy. *Nature Methods*, 13, 371. [PubMed: 26928762]
- Yosinski J, Clune J, Nguyen A, Fuchs T, & Lipson H (2015) Understanding neural networks through deep visualization. *arXiv preprint arXiv:1506.06579*.
- Zagoruyko S, & Komodakis N (2016) Wide residual networks. *arXiv preprint arXiv:1605.07146*.
- Zeiler MD, & Fergus R (2014) Visualizing and understanding convolutional networks. *European conference on computer vision Springer*, pp. 818–833.

Figure 1:
Example images from the DeepYeast (a) and transfer learning (b) datasets provided by Pärnamaa & Parts (2017) and Kraus et al. (2017) respectively.

Figure 2:
An 11-layer baseline CNN architecture.

a. Discrete convolution in each image channel **b. Maxpooling**

Figure 3:
Convolution with a 2*2 kernel by stride 1 on a 4*4 image (with only one channel).

Figure 4: Convolution layer learns the “residual” features from the previous layer by an identity shortcut connection.

Figure 5:

Performance of CNN's transfer learning a. Transfer learning accuracy with increasing sample sizes (averaged over ten bootstrapped experiments). Bars show ± 1 standard error of the mean accuracy. b. Confusion matrix for each localization class with the numbers indicating the proportions of classifications.

Table 1:

Sample sizes in the DeepYeast data.

Subcellular categories	Training	Validation	Test
Cell periphery	6924	961	1569
Cytoplasm	6935	1223	1276
Endosome	2692	697	689
ER	6195	1393	1755
Golgi	2770	208	382
Mitochondria	6547	1560	1243
Nuclear Periphery	6661	1252	1164
Nucleolus	7014	1147	1263
Nuclei	6440	1312	1627
Peroxisome	1683	297	164
Spindle	4713	1517	781
Vacuole	6426	936	587
Total	65000	12500	12500

Table 2:

Comparison of the VGG-19 and an 11-layered baseline CNN architectures.

VGG-19	Baseline CNN/VGG-type CNN/DeepYeast
19 weight layers	11 weight layers
Input: 224×224×3	Input: 64×64×3
conv3-64	conv3-64
conv3-64	conv3-64
maxpool 2×2	maxpool 2×2
conv3-128	conv3-128
conv3-128	conv3-128
maxpool 2×2	maxpool 2×2
conv3-256	conv3-256
conv3-256	conv3-256
conv3-256	conv3-256
conv3-256	conv3-256
maxpool 2×2	maxpool 2×2
conv3-512	Fully-connected layer-512
conv3-512	Dropout-0.5
conv3-512	Fully-connected layer-512
conv3-512	Dropout-0.5
maxpool 2×2	Fully-connected layer-12 (softmax)
conv3-512	(BN added except for the last FC layer)
conv3-512	
conv3-512	
conv3-512	
maxpool 2×2	
Fully-connected layer-4096	
Dropout-0.5	
Fully-connected layer-4096	
Dropout-0.5	
Fully-connected layer-1000 (softmax)	
# of parameters is 144,000,000	# of parameters is 3,128,908

¹. Each convolution layer is followed by a ReLU nonlinearity. Adding a stack of convolution layers helps to introduce nonlinearity and reduce the parameters.

². Batch normalization is not used in the original VGG model.

³. conv3-64 means the convolutional layer has 64 kernels with size 3*3. Similarly, convA-B means the convolutional layer has B kernels with size A*A.

Table 3

: Various CNN/ResNet architectures.

Block name	DeepYeast	Res18 (ours)	ResNet 50	Res50 (ours)	W40-4	W40-2
conv1_x	$[3 \times 3, 64] \times 2$	$[7 \times 7, 64]$	$[7 \times 7, 64]$	$[7 \times 7, 64]$	$[3 \times 3, 16]$	$[3 \times 3, 16]$
conv2_x	$[3 \times 3, 128] \times 2$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 3 \times 3, 16 \times 4 \\ 3 \times 3, 16 \times 4 \end{bmatrix} \times 6$	$\begin{bmatrix} 3 \times 3, 16 \times 2 \\ 3 \times 3, 16 \times 2 \end{bmatrix} \times 6$
conv3_x	$[3 \times 3, 256] \times 2$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 32 \times 4 \\ 3 \times 3, 32 \times 4 \end{bmatrix} \times 6$	$\begin{bmatrix} 3 \times 3, 32 \times 2 \\ 3 \times 3, 32 \times 2 \end{bmatrix} \times 6$
conv4_x		$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 64 \times 4 \\ 3 \times 3, 64 \times 4 \end{bmatrix} \times 6$	$\begin{bmatrix} 3 \times 3, 64 \times 2 \\ 3 \times 3, 64 \times 2 \end{bmatrix} \times 6$
conv5_x		$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 64 \end{bmatrix} \times 3$		
	max pooling [512-d fc] $\times 2$ 12-d fc, softmax				average pooling, 12-d fc, softmax	
# of parameters	3,128,908	605,452	23,596,940	775,116	8,966,348	2,252,140

¹. All ResNets' designs had shortcut connections between the convolutional building blocks (shown inside of the bracket)

². The VGG-type CNN counterpart of a particular ResNet follows almost same network design except for not having shortcut connections between the convolutional building blocks.

Table 4:

Comparison of prediction accuracy (on a test dataset) among different methods.

Network	Training Time	Test accuracy
11-layer VGG-type model (DeepYeast model)	6 h	0.851
11-layer VGG-type model with data augmentation (DeepYeast model)	6 h	0.874
Res18 (He et al., 2016)	2.45 h	0.853
Res50 (He et al., 2016)	12.75 h	0.886
Random Forest (Direct feature vectorization, 1000 trees)	2 h	0.596
XGBoost (Direct feature vectorization, 1000 trees)	10 h	0.679
Linear Discriminant Analysis	16 min	0.289
K-Nearest Neighbor (K=50 selected)	18 h	0.478
Support Vector Machine (C=8 selected)	18.3 h	0.228
Lasso Logistic Regression ($\lambda = 0.000796$ selected)	13 h	0.441
Feature extraction by DeepYeast:		
Random Forest	10 min	0.850
XGBoost	1 h	0.840
Feature extraction by VGG-19 (transfer learning):		
Random Forest	12 min	0.660
XGBoost	14 h	0.722
FC layers	3 min	0.73
18-layer VGG-type model A (VGG-type counterpart of Res18)	1.58 h	0.845
18-layer VGG-type model B (VGG-type counterpart of Res18 with only identity shortcut)	1.75 h	0.843
Res18 with identity shortcut	1.75 h	0.871
Res18 with identity shortcut (augmented data, SGD optimizer)	1.75 h	0.876
Res18 with identity shortcut (augmented data, Adam optimizer)	1.75 h	0.891
50-layer VGG-type model A (VGG-type counterpart of Res50)	13 h	0.819
50-layer VGG-type model B (VGG-type counterpart of Res50 with only identity shortcut)	5.9 h	0.839
Res50 with identity shortcut	5.4 h	0.854
Wide residual network (Wide ResNet) with widening factor 2	46 h	0.853
Capsule Network	4 h	0.815
Inception-ResNet V2	36 h	0.826

Note. Rows with the same shading indicates the one-to-one relationship between a VGG-type CNN and ResNet CNN