Large-scale Protein Atlas Compartmentalization Analysis

Github Repo: `https://github.com/KuangcongLiu/cs229Project.git`
Kuangcong Liu CECILIA4@STANFORD.EDU
Zijian Zhang ZZHANG7@STANFORD.EDU
Wen Zhou ZHOUWEN@STANFORD.EDU

1. Abstract

Large datasets of microscopy images of individual proteins are currently available, but we lack an efficient and accurate approach to systematically classify the compartmentalization of each protein by now. We are addressing this question by combining our biological expertise for feature selection with multiple neural network models. We are able to successfully extract feature scoring matrices for the whole image dataset and achieve decently accurate predictions with our optimized Resnet34 models.

2. Introduction

Proteins are key functional units in the human cell, and correct localization of each protein is critical for their proper functions that together enable life. Hence a precise and systematic map of protein localization is highly desired. Thanks to advances in high-throughput imaging, biologists have generated massive yet continuously growing databases of microscopy images. These images are generated at a far greater pace than what can be manually evaluated, hence the need is greater than ever for automating cellular image analysis to accelerate our understanding of protein function, interactions as well as their roles in human diseases.

In this project, we are focusing on a Kaggle competition: Human Protein Atlas Image Classification [1]. Specifically, the inputs of our algorithm are annotated microscopy images, and we would like to generate a model to accurately predict protein localizations represented as integer labels based on the images. To accomplish our goal, we first use computer vision approaches to extract images feature selected based on biological expertise, and then use multiple convolutional neural networks combined with our image feature scoring matrix to predict the localization of each protein.

[1] `https://www.kaggle.com/c/human-protein-atlas-image-classification/`

3. Related work

Previous attempts to tackle this problem have included numerous methods such as k-NN classifiers, support vector machines (SVM), neural networks and decision trees. More recently, similar questions have been successfully addressed using deep convolutional neural networks (CNNs) for single-localizing proteins in budding yeasts (7). CNNs have also been reported to show high accuracy prediction with human cells, however with relatively few features within single cell types. Especially, the targeted locations included in previous studies were mostly large and distinctive organelle-level localizations, which won't be of enough resolution for most biological studies. However, the severe class imbalance introduced, when considering rare cellular structures makes it harder to create a classifier that is capable of accurately predicting all localizations (4). More importantly, previous studies mostly only focus on single-localizing proteins, which excludes more than half of all known proteins. Disease-related proteins are especially known for its multi-territory characteristic, and therefore are mostly filtered in previous studies. Multiple attempts have been put forward to include multi-localizing proteins in the dataset, which usually dratstically decrease the prediction accuracy (3). Methods called "unmixing" of individual feature patterns have been tested recently, but still lack most of the 28 localizations described in our Human Protein Atlas databases (1).

As concluded from the previous works, convolutional neural networks have been given the most promising outcomes, but are still relatively lower in accuracy when introducing the severe class-imbalance and multi-localizing proteins. Interestingly, most previous studies were using either multi-channel images or extracted features solely as the input for the algorithms (6). We would like to leverage our biological expertise for feature set optimization, and combine the images as well as the feature scoring matrix as our input, to help improve the performance.

4. Dataset and Features

The dataset we are using is provided by Kaggle - Human Protein Atlas Image Classification [2]. In this dataset, each sample is a microscope image of a type of cells that contain a certain type of protein. The image is processed with 4 different filters, resulting in 4 individual images: the protein of interest (green), and three cellular landmarks: nucleus (blue), microtubules (red), and endoplasmic reticulum (yellow). A visualization of our input data is shown in Figure 1. The target is the classes of the green image, which denotes the protein organelle localization and is represented as integers, from 0 to 27. Each sample can belong to more than one class.

There are 31072 samples in the dataset for training, and we split 80% of it as train set and 20% of it as validation set. We also use affine and flip transforms to augment our data. There are 11702 samples in test dataset, and we can submit our predictions on it to Kaggle to get a leaderboard score, which is macro-F1.

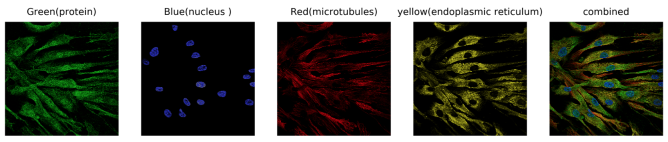


Figure 1. The left 4 images are the input of one sample. Each image has a resolution of 512x512.

4.1. Data Pre-processing

Most of previous classification approaches were purely machine learning-based, hence losing the vast majority of the rich biological information. As an optimization, we utilize our expertise in cell biology and select 10 most crucial features in terms of the relative localization to different organelles:

Overall properties:

1. Relative ratio of green in blue (localization)
2. Structural similarity between green & red
3. Structural similarity between green & yellow
4. Structural similarity between green & blue
5. Total intensity of green / yellow (Intensity)
6. Area size of green above background (Size)

Information about individual protein segments:

7. Averaged compactness of protein segments (shape)
8. Averaged eccentricity of protein segments (shape)
9. Average area size of each protein segments / nucleus (distribution)
10. Average distance of each protein blob to the closest nucleus (distribution)

[2] `https://www.kaggle.com/c/human-protein-atlas-image-classification/data`

4.1.1. Features Extraction Procedure

Feature extraction is implemented with opencv in Python (cv2 module).

1. *Overall properties*

We denoise the protein graph and make a mask for the nucleus component. By laying out the nucleus component on our protein graph we could get great insights about whether this protein is in or out of the nucleus, like shown in figure 2. We calculate structural similarity index (SSIM) to measure similarity between images.

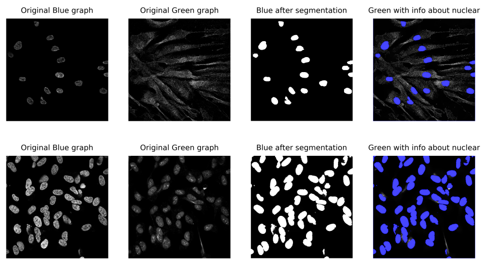


Figure 2. Representative images of how we find relative proportion of protein in nucleus.

2. *Information about individual protein segments*

First, we need to segment the blobs in green image. To do so, we first denoise and Gaussian-blur the image, and computes the connected components. Contours are drawn around the connected component and are fit into ellipses.

The shape features are calculated by: $compactness = \frac{perimeter^2}{4*\pi*area}$

$eccentricity = \frac{\text{major axis}}{\text{minor axis}}$.

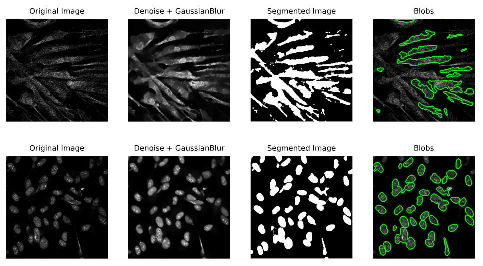


Figure 3. Representative images of how we segment proteins.

5. Methods

We use Resnet34 based on Residual Network for image recognition (2) as our baseline, and then add the calculated features to Resnet34 as our own model.

Because this is a multi-label classification problem, we decide whether an image belongs to each class independently, and we use binary cross entropy as our objective function for each of the 28 classes:

$$BCE(y_i, \hat{y}_i) = -(y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i))$$

$$i = 1, 2, 3, ..., 28$$

5.1. Residual Network

Residual Network (2) is built on residual block which adds an identity mapping to outputs of original stacked layers. When the optimal solution is closer to identity mapping than to zero mapping, Residual Network could efficiently find optimal solution, since it doesn't need to learn the identity function as a new one.

Another method we combined with basic Resnet model is to make the learning rate decay periodically. This cyclical learning rate schedule method is proposed by *Leslie N. Smith* (5). The learning rate is calculated as:

$$cycle = floor(\frac{1 + \text{epochs}}{\text{cycle length}})$$

$$lr = 1 - |\frac{2 * \text{epochs}}{\text{cycle length}} - 2 * cycle + 1|$$

When the learning rate is very low, it takes a long time for the algorithm to converge. But when the learning rate is too high, the algorithm can also be hard to converge because it may miss the minimum point with large step. Therefore, we make the learning rate increase and decrease in some specific period as shown in Figure 4.

Therefore, Residual Network could help us learn more features about the image, especially when the image has complicated features that are hard to recognize even for human eyes, like the noisy protein images in our dataset.

5.2. Network Architecture with Features

1. We treat the last fully connected layer output from Residual Network as a graph representation of each image sample.
2. After calculate our own data features, we normalize all the features so that each of them is between 0 and 1. Then we assign different weights to those 10 features according to their biological importance related to this problem, by [2, 4, 1, 1, 2, 1, 1, 0.5, 0.5, 2] respectively.

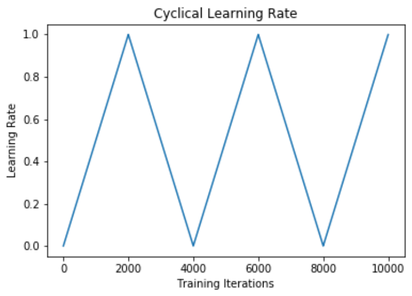


Figure 4. Periodical change of learning rate with respect to number of epochs

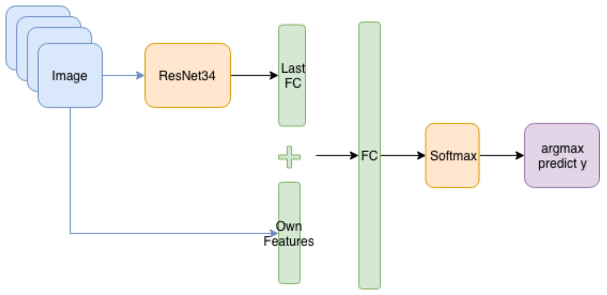


Figure 5. Network structure of combining Resnet34 with our own designed features.

3. Our network architecture is shown in figure 5. By combining this graph representation with our own designed features for each image sample as mentioned above, we could add more expertise-based protein classification features on locations and morphology to our state-of-the-art machine learning techniques.
4. We use another fully connected layer after the combination so that we could discover more non-linear relations between resnet representations and our image features. Since Resnet34 is a deep network, to solve the problem of overfitting and add more regularization, we use dropout after this FC layer to randomly remove some features and spread out the weight of each feature.
5. The last layer is a softmax layer to select the class labels with highest probabilities.

6. Experiments

We implemented our models in Python, and trained them on shared Stanford-wide GPU machines via Sherlock cluster.

6.1. Metrics: Macro-F1

Since this is a multi-label classification task and each sample can belong to more than one class, we decided to use the macro-average of F1-score as our primary metrics to evaluate our models. This metrics is in accordance with the score on Kaggle leader board.

For each class, let tp / fp be the number of true / false positive result, and tn / fn be the number of true / false negative result, then we have:

$$\text{precision}: p = \frac{tp}{tp + fp}$$

$$\text{recall}: r = \frac{tp}{tp + fn}$$

$$\text{F1-score}: F = 2 \cdot \frac{p \cdot r}{p + r}$$

Then the macro-average of F-scores of all classes $(\frac{\sum(F)}{\#classes})$ represents the performance of our model. This evaluation score ranges from 0 to 1, where 1 means the best prediction performance and 0 means the worst.

6.2. Residual Network

We first experimented on pure residual network without our features in order to find the best network model.

We adopted ImageNet pre-trained Resnet as our baseline model. Since our data has 4 color channels (RGBY) instead of 3 (RGB), we modified the basic Resnet model by initializing the first convolution layer to take in 4 channels instead of 3, keeping corresponding pre-trained weights for RGB channels, and initializing new weights for the 4th channel. We re-sized the image into 224x224 due to computing power limit.

Since we want to use the parallel processing power of our GPUs effectively, we chose a slightly large batch size, 64, which is a commonly used batch size for Resnet. We used 0.01 for our learning rate. This value was found by a learning rate finder implemented by FastAI, and it works by increasing the learning rate after each batch and picking the learning rate at which the loss starts to increase. After this point, the learning rate starts to get too big that a step will get pass the loss minimum.

6.2.1. Results & Discussion

Model	Epoch	Train Loss	Val Loss	**Kaggle LB**
zero initial weight for 4th layer (baseline)	20	0.0716	0.0808	0.366
zero initial weight for 4th layer + threshold	40	0.0502	0.0795	0.387
pre-trained weight for 4th layer + threshold	40	0.0499	0.0795	0.403
pre-trained weight for 4th layer + threshold + additional dropout	40	0.0700	0.0790	0.394
pre-trained weight for 4th layer + threshold + TTA	40	0.0509	0.0799	0.401

Table 1. Performance of different models, evaluated by Kaggle leaderboard(LB) score, which is the macro-F1 on test set. Loss is evaluated by binary cross-entropy loss.

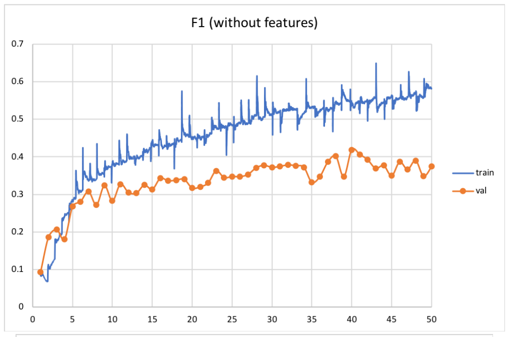
1. *Resnet34* & *Resnet50*
   We first trained both Resnet50 and Resnet34 for 20 epochs, but we found that Resnet50 would be badly overfitting as its validation set loss went as high as above 100 while its train set loss was continuously decreasing. Hence we continued with Resnet34. During this first experiment, we found that 20 epochs are not enough for the model to converge so we switched to 40 epochs later.
2. *Threshold*
   We explored the distributions of train and test datasets provided by Kaggle and found they are very different. So we decided to use different threshold probabilities for each class when identify labels for the test dataset images, according to its statistical information.
3. *Weight for Yellow Channel (4th layer)*
   Two different ways to initialize the weight for the 4th channel in the first Conv layer:
   (a) initialize it with all zeros
   (b) initialize it by copying the ImageNet pre-trained weights for the third channel.
   With the second initialization, our model reached a higher F1 score. This is because we are examine cells and pre-trained first conv layer from ImageNet are tuned to extract simple shapes, which fits our task.
4. *Additional Dropout*
   We tried to resolve overfitting issue by adding an additional 50% dropout layer before the last fully connected layer. This did successfully resolve the validation-train loss gap at the 40th epoch, but we found that this new model did not completely converge at the 40th epoch, and thus the Kaggle score is slightly lower. Unfortunately, we did not have time to train it with more epochs.

5. *Test Time Augmentation (TTA)*
We tried TTA to improve our F1-score on test set which does not change the model itself. However, we found that TTA was not very helpful on the test set for this task.

6.3. Concatenate Our Features

In the meantime when we experimented on Resnet, we were also extracting our own selected features and comparing two Resnet models (with different test threshold and additional dropout) without/with our own features concatenated.

During our biological analysis on features, we also found that the endoplasmic reticulum (yellow channel) is not very important in localizing proteins and it has many overlapping information with the red channel. So we discarded this channel to exchange for more memory and computing power such that we can pass in the original 512x512 images instead of resizing them into 224x224 and to use additional dropout layer for controlling overfitting and use an additional FC layer for feature concatenation.



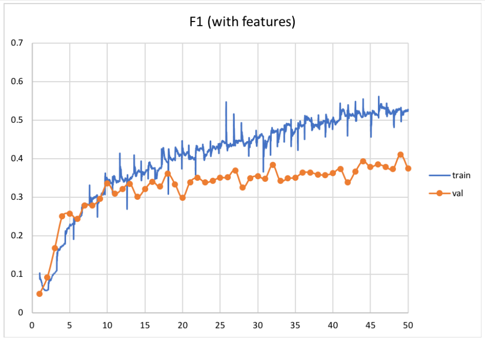


Figure 6. Macro-F1 on train and validation datasets. The top figure is for the Resnet34 model without our own features, and the highest validation-F1 it reached is 0.4189 at 40th epoch. The bottom figure is for the Resnet34 model with our own features concatenated, and the highest validation-F1 it reached is 0.4110 at 49th epoch.

6.3.1. Results & Discussion

We have trained both of the two models for 50 epochs, the batch size and learning rate are the same ones described in section 6.2. The two models' macro-F1 scores on train/validation dataset are visualized in Figure 6.

After examining Figure 6, we identified the following two problems with our model, and we also think about several solutions:

1. The model still has overfitting as there is a gap between train f1 and val f1. In the future, we will try to resolve this issue by:
   (a) Add some regularization such as L2 regularization or weight decay regularization in Adam
   (b) Add more external training image data from other sources
2. Our feature scoring matrix did not significantly boost the prediction performance.
   (a) A reason to this problem could be that the small scale of our own features, which is now mostly within range of (0, 1), may prevent the gradient being effective in backpropagation. So in the future, we will try to scale our own features by higher factors to make back propagation capture more information about it.
   (b) We can also develop more related features from the RGBY channels that could help us explore more about the biological features.

7. Conclusion

We have developed several different deep learning models to solve the task of human protein compartmentalization. We first experimented on pure ResNet models and found that using different thresholds for test and train datasets and initializing weights for additional input channel with ImageNet pre-trained weights can significantly boost the prediction performance. The highest macro-F1 score we have achieved on test dataset is 0.403. Additional dropout layer can help mitigate overfitting but whether it can improve the final performance needs further experiment in the future.

The second part of our study is to extract our own features according to biological knowledge and concatenate them into ResNet. Our feature scoring matrix however, needs more fine tuning in order to boost the effect, and future work is discussed in section 6.3.1.

8. Contributions

Kuangcong Liu: pre-process data features #1-#6, and design models for combining the resnet representation and our own features. Implement resnet34 model with our own features. Analysis on models.

Zijian Zhang: design models for the pre-processing process, select the proper features for the scoring matrix, and implement the training process of the Resnet neural network and analysis.

Wen Zhou: segment individual proteins and extract features #7 - #10. Implement the 4-channel Resnet models without our own features as well as the train/test procedure. Analyze on different models' performance.

References

[1] L.P. Coelho. Quantifying the distribution of probes between subcellular locations using unsupervised pattern unmixing. *Bioinformatics*, 2010.

[2] Shaoqing Ren Jian Sun Kaiming He, Xiangyu Zhang. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[3] O.Z. Kraus. Classifying and segmenting microscopy images with deep multiple instance learning. *Intell. Data Anal.*, 2002.

[4] Japkowicz N. The class imbalance problem: A systematic study. *Intelligent data analysis*, 2002.

[5] Leslie N. Smith. Cyclical learning rates for training neural networks. *Applications of Computer Vision (WACV), 2017 IEEE Winter Conference.*, 2017.

[6] Winsnes C. Åkesson L. et al. Sullivan, D. Deep learning is combined with massive-scale citizen science to improve large-scale image classification. *Nature biotechnology*, 2018.

[7] LeCun Y. Deep learning. *Nature*, 2015.