Human Protein Atlas Image Classification

Amita C. Patil and Rudra S. Bandhu
Department of Computer Science
Stanford University
(amita2, rsbandhu)@stanford.edu

December 17, 2018

Abstract

We investigated models capable of classifying 28 different protein organelle localization labels from 27 different cell structures of highly different morphology in microscope images. We start with models pretrained on ImageNet, fine-tuned only the classifier layer, followed by the entire network. Experimentation was done with different network architectures, loss functions and training methods. DenseNet161 with weighted loss, unique threshold per class and data augmentation achieve best performance with macro F1 score of 0.76 on the validation set.

1 Introduction

Proteins perform important functions in human cells; thereby enabling life. In order to understand the complexity of human cell we need to classify mixed patterns of proteins across a range of cell types. Given the advances in high throughput microscopy, we can accelerate our understanding of human cells and diseases by automating the localization of proteins in cellular substructures by developing models using machine learning techniques [1]. This method holds promise to generalize across different protein types without the knowledge of specific protein type for developing models.

2 Related work

Primary means of annotating protein patterns in microscopic images has been visual inspection. Over the last couple decades various machine learning methods have been used to automate the determination of sub cellular locations in microscopic images [3]-[6]. In these references, numerical features were first extracted from input images; and these features were used as inputs to either a classification tree, support vector machine classifier or a neural network for class prediction. In [7] a DeepBind framework was built using DL approach which outperformed other state-of-the-art methods for predicting the sequence specificities of DNA- and RNA-binding proteins. DeepBind processes five independent sequences in parallel using conv, rectify, pool and NN stages to predict separate scores for each sequence. To address the frequent concern of over-fitting to training data, several regularizers developed in the DL community- including dropout, weight decay and early stopping along with evaluation of 30 calibration settings were incorporated.

3 Dataset and Features

Our dataset is from Kaggle Human Protein Atlas Image Classification website [1] as part of the Human Protein Atlas program [2]). We have 31072 samples, each sample consists of 512 x 512 images from four (RGBY) color filters. The green channel shows the location of the protein obtained by immunoflourescence microscopy after staining using targeted antibodies. The other 3 channels (blue, green and yellow) shows the different regions of the cell. Each samples can have one or more labels from 28 different cell regions corresponding to sub-cellular structure of the protein to be localized. Our dataset has class imbalance that makes the classification problem challenging. For e.g. Nucleoplasm (Class0) is present in 12885 samples, whereas Rods and Rings (Class27) is present in only 11 samples. Lysosomes, Microtubule ends and Mitotic spindle (Classes10, 15 and 17) are not present as unique labels but along with other classes only.

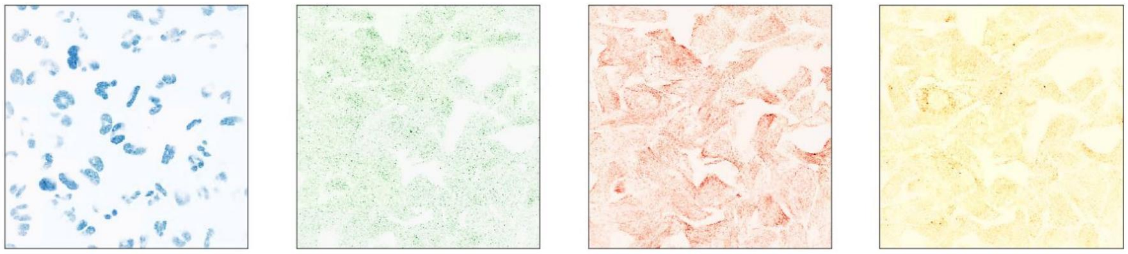


Figure 1: Representative sample image with multiple labels: Nucleoplasm and Microtubule organizing center.

3.1 Data Preprocessing

5% of both unique and multilabel samples along with at least 5 samples from each class were held out for validation. Validation set includes 1576 samples. All images were downsampled from 512 x 512 to 224 x 224 so we could use pretrained networks such as ResNet18 and DenseNet161 for finetuning. We performed two levels of data augmentation on training samples

1) 3 rotations: 90°, 180°, 270°and 2 mirrorings: horizontal and vertical flips
2) 7 rotations: 45°through 315°in 45°steps and the 2 mirrorings.

4 Methods

4.1 Loss Formulation

For our multi-class classification problem we used Binary Cross Entropy with logits loss. The loss function is described below:

$$Loss = -\frac{1}{N}\sum_{i=1}^{N} w_i * [p_i * y_{true_i} * log(y_{pred_i}) + (1 - y_{true_i}) * log(1 - y_{pred_i})]$$

where N=28 (# of classes), w_i = weights for class i and p_i = weights for positive examples of class i.

We used different weighting methods in the loss function to address: 1) Large number of negative labels; i.e. 24 to 27 out of 28 labels are negative in each sample, and 2) Large imbalance of class sample count

Our weighting experiments are summarized below:

1) $w_i \propto \frac{1}{\log(N_i)}$ We scaled this to class 27 (class with fewest occurrences). Weights w_i ensure we penalize more heavily for mispredicting classes that have fewer counts.

Weights p_i are needed for positive classes to account for imbalance between positive and negative labels. We have experimented with two choices for p_i:

2) $p_i = \frac{N_{tot}-N_i}{N_i}$
3) $p_i = \sqrt{\frac{N_{tot}-N_i}{N_i}}$

Here N_{tot} is the total number of training samples and N_i is number of times class i appears in it.

4.2 Models

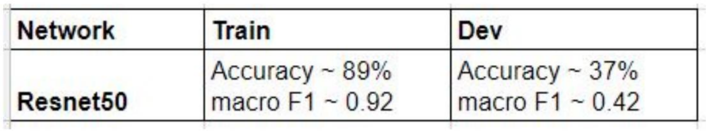
The training framework is developed in PyTorch [8]. Starting from simple to complex we trained following architectures:

- Tuning only classifier layers of VGG19, Resnet18, Densenet161 pretrained on ImageNet.
- Finetuning entire Resnet18 and Densenet161 networks
- Fully train a simpler Densenet architecture (we call it Densenet73) from scratch. This architecture consists of 3 dense blocks with 6,12,16 layers respectively and initial feature size of 64 and growth rate of 12.

5 Experiments/Results/Discussion

5.1 Tuning Classifier Network

When tuning classifier layer best performance was achieved using Resnet50. Performance of the training and dev set is shown in Table 1. All 3 networks were over-fitting to training data.



Network	Train	Dev
Resnet50	Accuracy ~ 89% macro F1 ~ 0.92	Accuracy ~ 37% macro F1 ~ 0.42

Table 1: Performance summary Resnet50 Classifier only

5.2 Fine Tuning pretrained Network

Macro F1-score [10] on the validation set is chosen as the performance metric. We want to maximize the macro F1-score of the validation set. Threshold value is optimized for each class. Learning rate=0.0001. Models were trained for 20 epochs or until the loss plateaued using Adam optimizer. Mini-batch size of 32 and 128 were attempted to stay within the memory budget (16GB) of our AWS instances.

Performance on both train and validation sets improved when we fine-tuned the entire Densenet161 (see table 2 for comparison summary).

Without any data augmentation Densenet161 achieved higher performance using Loss weight type 3 for positive examples (compare columns 2 and 3 in Table 3). Macro F1 score on training and validation set increased to 0.98 and 0.64 respectively. This macro metrics suggests we are still over fitting the training dataset.

To address over-fitting of training data following approaches were experimented with:

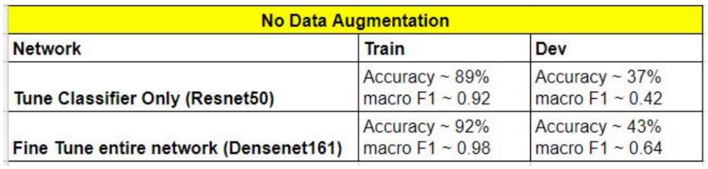
- 2 levels of data augmentation (see section 3.1)
- Drop out in dense layers with a drop rate of 0.2, 0.3
- Use simpler network e.g. resnet18 network (fewer parameters)

With level 2 data augmentation (see section 3.1 for details) macro accuracy increased to 0.56 and macro F1 score to 0.76 on the validation data. This result suggest that the network is able to generalize to the dev set but over-fitting still remains an issue. We believe the task of predicting some infrequently occurring classes could be particularly challenging. Specifically for Class27 we have only 6 unique labels in train set and 5 labels of this class to predict in the validation set. Further, the macro metrics do not take class imbalance into account.

Thresholding per class improves macro F1 score on validation set by 5 to 6%. A plot of macro F1-score vs. threshold for Classes 0 through 6 is shown in Figure 2:

A summary of our experiments is shown in Table 3. All models are trained using RGB images for 20 epochs using learning rate of 0.0001. Densenet73 was trained from scratch while for all other models we fine-tuned the entire network. The validation metrics are reported using optimum threshold for each class.

When we used RGY images instead of RGB images the performance was poor on both train and validation sets. This result makes sense since blue image filter represents the cell nucleus and our dataset contains many labels representing the cell nucleus e.g. Nucleoplasm, Nuclear membrane, Nucleoli, Nucleoli fibrillar center, Nuclear speckles and Nuclear bodies.



No Data Augmentation		
Network	Train	Dev
Tune Classifier Only (Resnet50)	Accuracy ~ 89% macro F1 ~ 0.92	Accuracy ~ 37% macro F1 ~ 0.42
Fine Tune entire network (Densenet161)	Accuracy ~ 92% macro F1 ~ 0.98	Accuracy ~ 43% macro F1 ~ 0.64

Table 2: Comparison of tuning Classifier only vs fine tuning entire network with no data augmentation

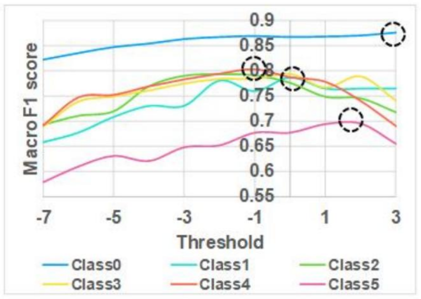


Figure 2: Picking optimum threshold per class for maximizing macro F1-score on validation set.

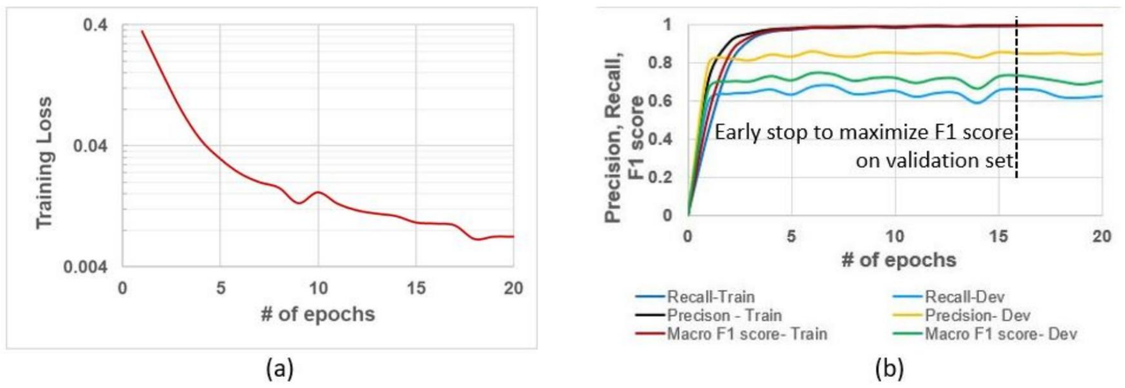


Figure 3: (a) Training Loss vs. # of epochs and (b) Macro Accuracy, Precision, Recall and F1 score for train/validation set vs. # of epochs

Figure 3(a) shows training loss vs. # of epochs for our best model. In 20 epochs our training loss has plateaued to reasonably low value. Figure 3(b) shows macro metrics on train and validation sets. Accuracy, precision, recall and F1 score for training data reaches to 1.0 within 10 epochs; where as on validation set we have macro F1 score of 0.76 after 20 epochs. We could use early stopping to improve macro F1 score on validation data by another 1%.

	Network	Densenet73	Densenet161	Densenet161	Densenet161	Densenet161 + drop rate in dense layers = 0.3	Densenet161	Resnet18
	Data Augmentation	None	None	None	Level1	Level1	Level2	Level1
Metrics (Macro)	Loss Weight Type	1	2	3	3	3	3	3
Accuracy	Train	0.57	0.24	0.92	0.97	0.84	0.98	0.94
	Dev	0.38	0.19	0.43	0.49	0.46	0.56	0.47
Precision	Train	0.82	0.58	0.98	0.99	0.975	0.996	0.99
	Dev	0.62	0.53	0.68	0.78	0.73	0.83	0.76
Recall	Train	0.57	0.23	0.97	0.99	0.96	0.995	0.99
	Dev	0.38	0.5	0.62	0.7	0.73	0.73	0.69
F1 score	Train	0.65	0.31	0.98	0.99	0.967	0.995	0.99
	Dev	0.43	0.48	0.64	0.73	0.7	0.76	0.71

Table 3: Performance summary of our models.

5.3 Tuning Densenet73 from scratch

To address over-fitting issue with Densenet161, we trained a densenet architecture of 73 layers (3 dense blocks with 6, 12, 16 dense layers respectively), initial feature size 64 and growth rate 12. The entire network was tuned from scratch with no data augmentation. Performance of this network is shown in Table 3 column 1. However, we didn't have enough time to tune the hyper-parameters. We believe that performance on this network can be improved by carefully tuning the hyper-parameters and using data augmentation.

6 Visualization

In order to better understand the classification power of the network, we plotted the class activation map (CAM) samples that were predicted correctly by the model. CAM plots from 2 samples overlaid on each of the 4 channels for both samples are shown in Fig 4 and 5. In figure 4 one can see from the blue channel that highlighted regions are outside the nucleus correctly indicating that the protein is localized somewhere in the cytosol. Since mitochondria is normally distributed throughout cytosol, CAM plot shows that both red and and yellow channels are highlighted. Fig 5 shows the CAM plot where the protein is delocalized in the entire cytosol region. In this case one can see the almost all cytosol regions except the top right corner are highlighted.

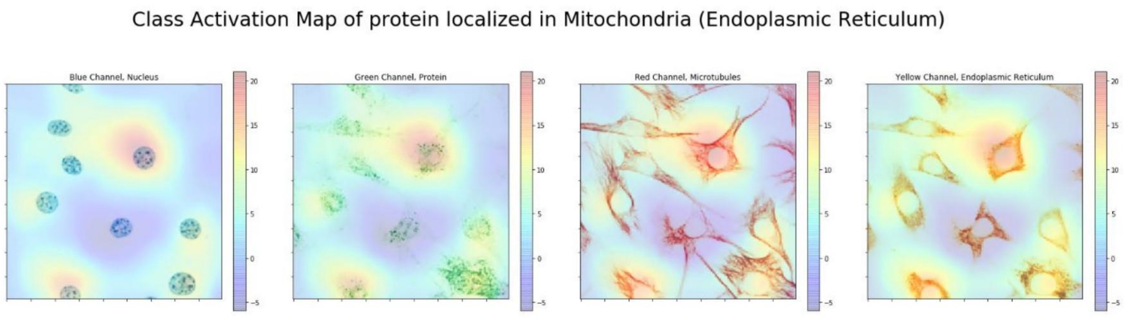

Class Activation Map of protein localized in Mitochondria (Endoplasmic Reticulum)
Blue Channel, Nucleus
Green Channel, Protein
Red Channel, Microtubules
Yellow Channel, Endoplasmic Reticulum

Figure 4: CAM plots of protein localized in mitochondria. Regions of interest are clearly outside the nucleus

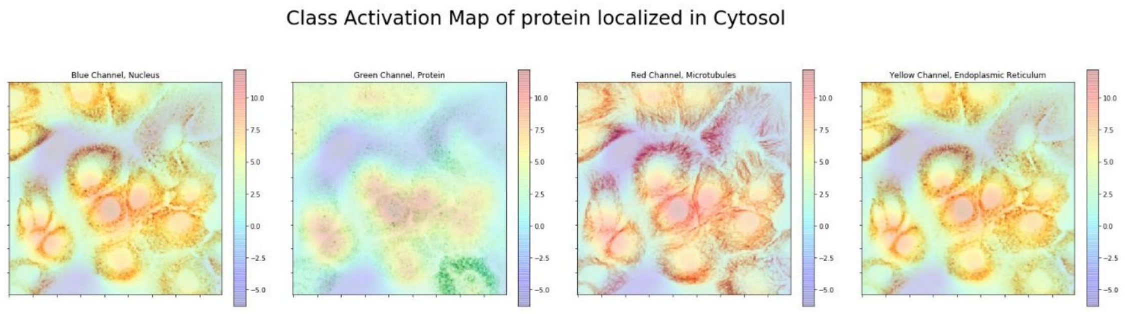

Class Activation Map of protein localized in Cytosol
Blue Channel, Nucleus
Green Channel, Protein
Red Channel, Microtubules
Yellow Channel, Endoplasmic Reticulum

Figure 5: CAM of protein delocalized in entire Cytosol region

7 Future Work

To address over-fitting issue and improve generalization capability of the model we suggest the following options for future work:

- Use full resolution images 512 X 512 and level 2 data augmentation as input features to the network. This will enable the network to learn small sub-cellular structures in the images.
- Train a light weight architecture e.g. densenet73 with careful optimization of the hyper-parameters using the dataset as above.
- Train separate networks to learn structures inside the nucleus and inside cytosol (outside the nucleus) separately and combine the results of both networks.

8 Contributions

Amita and Rudra contributed equally in all parts of the project including code development, analysis and preparation of reports. Our github repository is located here: $https://github.com/amita2/CS230\_Project$

References

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

[2] https://www.proteinatlas.org/humanproteome/cell

[3] Boland, Michael V., Mia K. Markey, and Robert F. Murphy. "Automated recognition of patterns characteristic of subcellular structures in fluorescence microscopy images." Cytometry: The Journal of the International Society for Analytical Cytology 33.3 (1998): 366-375.

[4] Newberg, Justin, and Robert F. Murphy. "A framework for the automated analysis of subcellular patterns in human protein atlas images." Journal of proteome research 7.6 (2008): 2300-2308.

[5] Glory, Estelle, and Robert F. Murphy. "Automated subcellular location determination and high-throughput microscopy." Developmental cell 12.1 (2007): 7-16.

[6] Shariff, Aabid, et al. "Automated image analysis for high-content screening and analysis." Journal of biomolecular screening 15.7 (2010): 726-734.

[7] Alipanahi, Babak, et al. "Predicting the sequence specificities of DNA-and RNA-binding proteins by deep learning." Nature biotechnology 33.8 (2015): 831.

[8] https://pytorch.org/docs/stable/index.html

[9] https://pytorch.org/docs/stable/torchvision/models.html

[10] https://scikit-learn.org/stable/modules/classes.html