Human Protein Atlas Image Classification Using Deep Learning

Mahit Tanikella
Monta Vista High School
21840 McClellan Rd, Cupertino, CA 95014

Acknowledgements

I want to thank my Math teacher Ms. Kathleen Koch for being my mentor and guiding me throughout.

Introduction

Cells are basic structural units of all living organisms. Human body consists of trillions of cells. Each cell contains many proteins organized to maintain its structure and function. These proteins are the building blocks of human bodies. Without them, we would not exist. Almost all of the medications are directed against proteins. As our knowledge about proteins is very little, today's drugs are targeting only very small percentage of the 20,000 different proteins found in the human body. Images visualizing proteins in cells are commonly used for biomedical research, and these cells could hold the key for the next breakthrough in medicine. With advances in high-throughput microscopy, these images are generated at a far greater pace than what can be manually evaluated. Therefore, the need is greater than ever for automating biomedical image analysis to accelerate the understanding of human cells and disease.

In this research paper, a neural network capable of identifying 28 different proteins in cell microscope images is developed and trained with cell microscope images from human protein atlas data [1] [4]. The model has achieved 96.9% accuracy with validation data. This model can be used to build a tool integrated with smart-microscopy system to identify proteins from a high-throughput image.

Materials and Methods

Human protein atlas image data was downloaded from kaggle website [4]. The images are 512x512 PNG files. Each of these images can have one or more of the proteins from Table 1. As shown in Table 2, 48.68% of the images have one protein, 40% have two proteins and 10% have 3 proteins. Data about what proteins are present in what image is provided in a separate metadata file. All image samples are represented by four filters (stored as four individual files), the protein of interest (green) plus three cellular landmarks, microtubules (red), nucleus (blue), and endoplasmic reticulum (yellow). The green filter is to be used to predict the label, and the other filters are references. Each file is for a different color and represents a different filter on the subcellular protein patterns represented by the sample. Figure 1 shows green, red, blue and yellow samples for two different images.

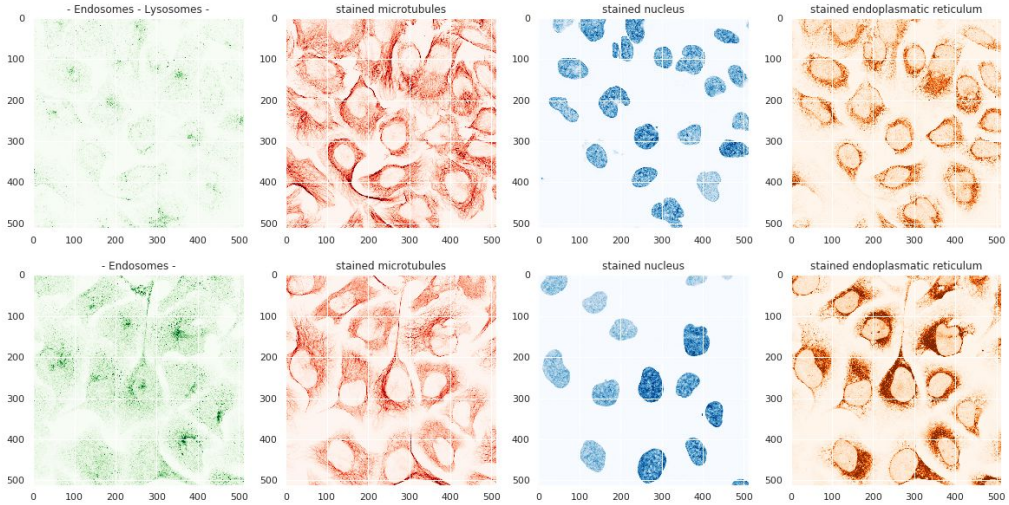

Figure 1: Sample Images

Protein Name	Count
Nucleoplasm	12885
Cytosol	8228
Plasma Membrane	3777
Nucleoli	3621
Mitochondria	2965
Golgi Apparatus	2822
Nuclear bodies	2513
Nuclear speckles	1858
Nucleoli fibrillar center	1561
Centrosome	1482
Nuclear membrane	1254
Intermediate filaments	1093
Microtubules	1066
Endoplasmic reticulum	1008
Microtubule organizing center	902

Protein Name	Count
Cell junctions	802
Actin filaments	688
Focal adhesion sites	537
Cytokinetic bridge	530
Cytoplasmic bodies	328
Aggresome	322
Mitotic spindle	210
Lipid droplets	172
Peroxisomes	53
Endosomes	45
Lysosomes	28
Microtubule ends	21
Rods & rings	11

Table 1: Count of different proteins in microscope images

Number of targets per image	% of data
1	48.68
2	40.18
3	10.17
4	0.96
5	0.01

Table 2: Number of targets per percent of data

Machine learning model was created and trained on Google Cloud platform free tier, which provided $300 free credit. A virtual machine is created on Google Cloud with 1 NVIDIA Tesla K80 GPU, 4 vCPUs, 26 GB memory, and 150 GB hard disk . Compressed images of size 17 GB are downloaded and saved on this virtual machine. After uncompression, total size of the images was around 32 GB, with each image around ~100 KB in size. There are 31072 images in total. Each image is 512x512 pixels.

Convolutional Neural Networks (CNNs) with deep learning are commonly used for image classification tasks. Transfer learning is a technique where instead of training a large CNN from scratch, training is done on top of pretrained models. Training large neural networks from scratch requires very large datasets, time and compute resources. As all the extracted features i.e. edges, circles, lines etc. from images are similar, features extraction part of a pre-trained model can be reused . Keras [2], deep learning software library, provides multiple pretrained models trained

on ImageNet, a very large dataset of 1.2 million images with 1000 categories available. The state of the art Inception Resnet V2 model [3] is chosen for this work. A custom classification layer is built to replace the final classification layer in the original pretrained model. The final architecture of the network built is as shown in Figure 2. The dense output layer has 28 outputs, one for each of the protein types.

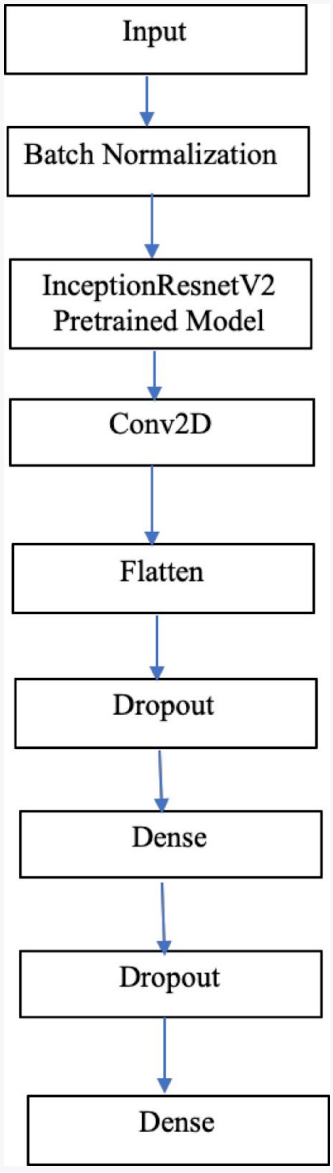

Figure 2: Neural Network Architecture Used

Data is randomly split into two parts, 80% for training and 20% for validation. To use the pretrained model, we have to provide input in 299 x 299 x 3 shape. Red, green and blue images were reshaped to 299 x 299 and combined to form a 3 channel input to neural network for training. Green filter is the most important. Since the pretrained network can only take 3 channel input, different combinations of red, blue and yellow were tried and yellow was found to be least impactful i.e. combination of yellow with any other color is bringing the accuracy down. So, yellow filter is not used. Custom data generator class which combines red, green, and blue images along with image preprocessing is implemented. Images preprocessing included normalization and augmentation. Augmented data was generated by randomly rotating, flipping and changing brightness of the images. Augmentation is used only for training data, not for the validation data. Figure 3 shows samples of images produced by data generator. No yellow is seen in the images as that color is not included.

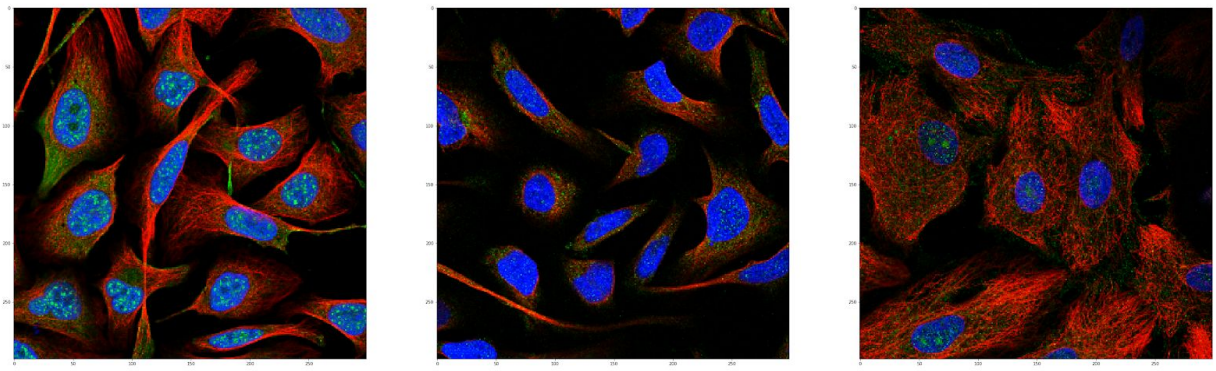

Figure 3: Combined Red, Green and Blue images from Data generator Fed into the neural network

For training, binary cross entropy [6] loss function was used with adam optimizer [5]. The learning rate used was 0.001. The model was trained for 150 epochs with batch size of 16. The training accuracy, and validation accuracy are saved for each epoch as training progressed. The model was trained with early stop configured, which will stop training if the validation loss does not improve. It stopped after 150 epochs.

Results

Figure 4 shows how both training and validation loss decreased as training progressed with increase in number of epochs. This shows that the learning rate chosen was good. Also, model is not overfitting because both training and validation loss are going down.

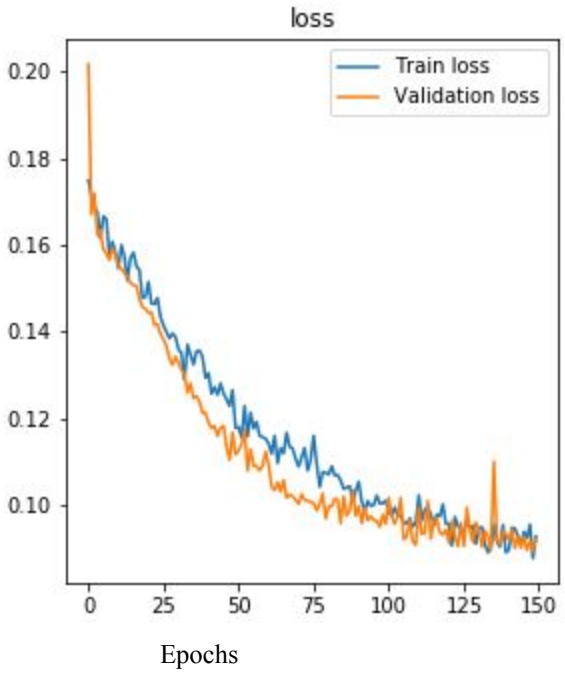

Figure 4: Training and Validation Loss vs. Epochs

Figure 5 shows how training and validation accuracy improved with increase in number of epochs. Accuracy of 96.9% was achieved with validation data when the training was stopped as there was no improvement in validation loss.

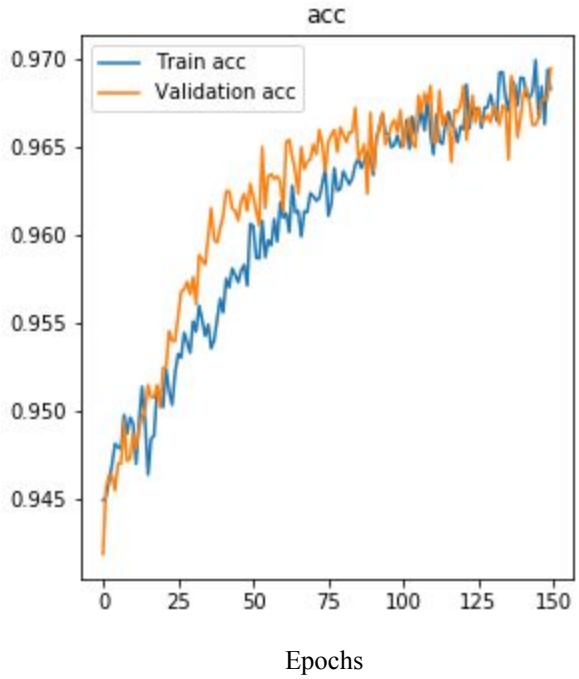

Figure 5: Training and Validation Accuracy vs. Epochs

Conclusions

The neural network model developed has achieved accuracy of 96.9% with validation data. Further improvement in accuracy can be possibly achieved by tuning hyper parameters and adding more layers to the network. This model can be used in biomedical research for automated identification of proteins in cell microscopic images, enabling us to better diagnose and create better drugs for diseases. This will also help us understand the biology of human beings in more basic sense.

References

[1] The human protein atlas. https://www.proteinatlas.org/

[2] Keras: The Python Deep Learning library. https://keras.io/

[3] Szegedy, C., Ioffe, S., & Vanhoucke, V. (2017). Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning. *AAAI*.

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

[5] Kingma, D.P., & Ba, J. (2014). Adam: A Method for Stochastic Optimization. *CoRR, abs/1412.6980*

[6] Cross Entropy. https://en.wikipedia.org/wiki/Cross_entropy