CAP 5610 Machine Learning Course Project

Human Protein Atlas Image Classification

Yan Xiao

July 28, 2020

Florida International University

Jinglin Lyu

Instructor: Dr. Miguel Alonso

**Table of Contents**

[ABSTRACT 4](#_Toc47053195)

[1. INTRODUCTION 4](#_Toc47053196)

[2. LITERATURE REVIEW 4](#_Toc47053197)

[3. DATA EXPLORATION AND VISUALIZATION 5](#_Toc47053198)

[3.1. Human Protein Atlas Image Classification Challenge Dataset 5](#_Toc47053199)

[3.2. CSV File Exploration and Visualization 6](#_Toc47053200)

[3.3. Image File Exploration and Visualization 8](#_Toc47053201)

[4. METHODOLOGY 9](#_Toc47053202)

[4.1. Data Preprocessing 9](#_Toc47053203)

[4.2. Model Development 10](#_Toc47053204)

[4.3. Model Application and Results Visualization 12](#_Toc47053205)

[5. RESULTS AND DISCUSSION 13](#_Toc47053206)

[6. CONCLUSIONS AND RECOMMENDATIONS 15](#_Toc47053207)

[REFERENCE 15](#_Toc47053208)

**List of Figures**

[Figure 3‑1 Top 5 Rows in Train.csv File 5](#_Toc47053209)

[Figure 3‑2 Example of Binary Target Columns after Conversion 6](#_Toc47053210)

[Figure 3‑3 Class Distribution in Training Dataset 6](#_Toc47053211)

[Figure 3‑4 Distribution of Number of Classes in One Image 7](#_Toc47053212)

[Figure 3‑5 Correlation between Protein Types 7](#_Toc47053213)

[Figure 3‑6 Three Samples of Protein Cells with a Class of Endosomes 8](#_Toc47053214)

[Figure 4‑1 Structure of Study Methodology 9](#_Toc47053215)

[Figure 4‑2 Structure of Basic CNN Model 11](#_Toc47053216)

[Figure 4‑3 Python Code for CNN Model 11](#_Toc47053217)

[Figure 4‑4 Example of Output Metrics Visualization 12](#_Toc47053218)

**List of Tables**

[Table 5‑1 Impacts of Image Augmentation Using Transforms 13](#_Toc47053219)

[Table 5‑2 Impacts of Threshold Selection 14](#_Toc47053220)

[Table 5‑3 Impacts of Modeling Structure 14](#_Toc47053221)

[Table 5‑4 Impacts of Loss Function 15](#_Toc47053222)

# **ABSTRACT**

High-resolution microscope images of protein provide useful information regarding diseases. The Human Protein Atlas hosted an image classification challenge at the Kaggle website. A convolutional neural network model was developed in this project to identify mixed protein pattern in human cells using images provided by the challenge. Different strategies such as image augmentation, learning parameter adjustment, loss function definition, network structures, and threshold configuration have been explored in this project to improve model prediction performance.

# **INTRODUCTION**

Protein pattern in human cell contains important information regarding diseases. Correct classification of protein patterns is very helpful for better understanding diseases and production of medicine. Traditional studies only classify protein into a single pattern or very limited cell types, which may not be able to capture the variation in human cells.

With the availability of high-resolution microscope images, a further examination of protein complex patterns become feasible. The Human Protein Atlas hosted an image classification challenge at the Kaggle website. It would be interesting to develop a machine learning model based on the image data provided by this challenge to identify mixed protein patterns in human cells. Various machine learning techniques have been used in practice for image classification, for example, support vector machine, decision trees, k nearest neighbor, convolutional neural network, and so on (Jain, 2020). Among these methods, convolutional neural network seems to be the most commonly used deep learning method for analyzing images. Therefore, the goal of this project is to develop a convolutional neural network model to classify protein types that exist or coexist in human cell images.

# **LITERATURE REVIEW**

This section provides a brief review of modeling techniques, structures, and performance that have been reported in the literature for protein image classification.

Yang et al. (2019) provided a summary of Human Protein Atlas Image Classification competition, which covered different aspects of competition including participation and performance, and strategies used by the top-ranking solutions. According to the survey conducted by the competition organizer, 44 out of 56 teams used variation of neural network architectures such as Inception, Densenet, and Resnet. 34 out of 56 teams used binary cross entropy as loss function. Other strategies for handling unbalanced classes and improving performance include applying class weights, data augmentation, and adding supplementary images.

Pärnamaa and Parts (2017) trained 11-layer neural network model based on high-throughput microscopy data to classify 12 fluorescent protein subcellular localization patterns in yeast cells. The first 8 were convolutional layers with rectified linear units. The number of units used in the first two convolutional layers was 64, 128 for the layers 3 and 4, and 256 for the remaining 4 convolutional layers. The last 3 layers were fully connected layers with a number of units of 512, 512, and 12. The output class was determined by the softmax function.

Rumetshofer et al. (2019) proposed a new CNN architecture (GapNet-PL) to determine the existence or coexistence of 13 major organelles in Human Protein Atlas project protein images. The GapNet-PL consisted of two-steps. The first step was to use an encoder that was made of several convolutional layers to learn features on different spatial resolutions. The second step was to reduce features to vectors through global average pooling and paas them to a fully connected network with two hidden layers to make final prediction. SELU was used as activation function. The proposed architecture was compared to DenseNet, Multi-scale Convolutional Neural Network (M-CNN), DeepLoc, FCN-Seg, and Convolutional Multiple Instance Learning (Convolutional MIL). The proposed GapNet-PL showed better performance in terms of the metrics of F1 score and AUC.

Ezat et al. (2019) applied convolutional neural network that was pre-trained on CNN CAFFE Image-Net based on large dataset of ILSVRC to classify the PASCAL VOC 2007 image dataset into 4 classes. The performance of the developed model was compared to support vector model and showed better performance.

The Kaggle website posted a leaderboard for Human Protein Atlas Image Classification Challenge (Kaggle, 2020). The performance is quantified by macro F1-score using 29% of the test data. There is a total number of 2160 teams listed on the leaderboard. The highest score is 0.65602 while the lowest score is 0.0.

# **DATA EXPLORATION AND VISUALIZATION**

## **Human Protein Atlas Image Classification Challenge Dataset**

The dataset used in this project was downloaded from the Kaggle website for Human Protein Atlas Image Classification Challenge. It consists of two csv files (tain.csv and sample\_submission.csv) and two image folders (one for training and another one for testing). Figure 3-1 shows the first 5 rows of train.csv. As shown in this figure, there are two columns. The first column lists the image id, which can be used to find the corresponding images under the train image folder. The second target column lists the types of protein found in that image. A total number of 28 protein types have been included in the challenge dataset. In order to use those target protein types in the development of multi-class classification model, the target column was converted into multiple binary columns that correspond to 28 protein types in this study, as shown in Figure 3-2. A total number of 62,144 samples is included in the training dataset and 23,404 samples for the testing dataset.



**Figure 3‑1 Top 5 Rows in Train.csv File**

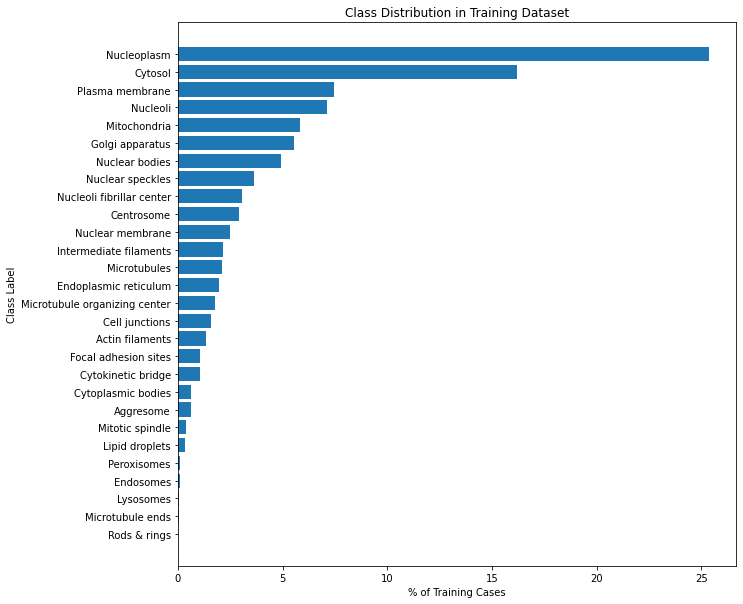


**Figure 3‑2 Example of Binary Target Columns after Conversion**

## **CSV File Exploration and Visualization**

* **Examination of Class Distribution in Training Dataset**

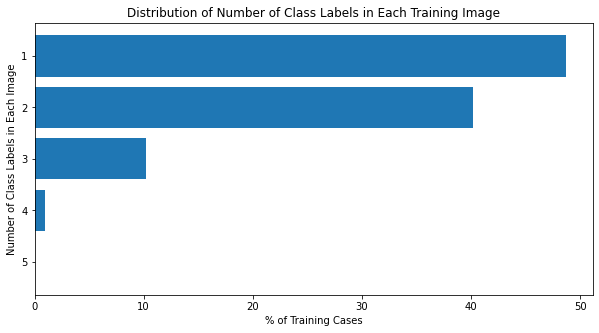
As the first step of data exploration and visualization, the occurrence of each class in the training data were counted. Note that one sample image may have multiple classes. The corresponding percentage of each class were calculated and shown below. It is seen from this figure that the most common protein is Nucleoplasm (Class 0) having a percentage of 25.4%, followed by Cytosol (Class 25) with a percentage of 16.2%, and 7.43% of Plasma Membrane (Class 21).



**Figure 3‑3 Class Distribution in Training Dataset**

* **Number of Coexisting Classes in Training Images**

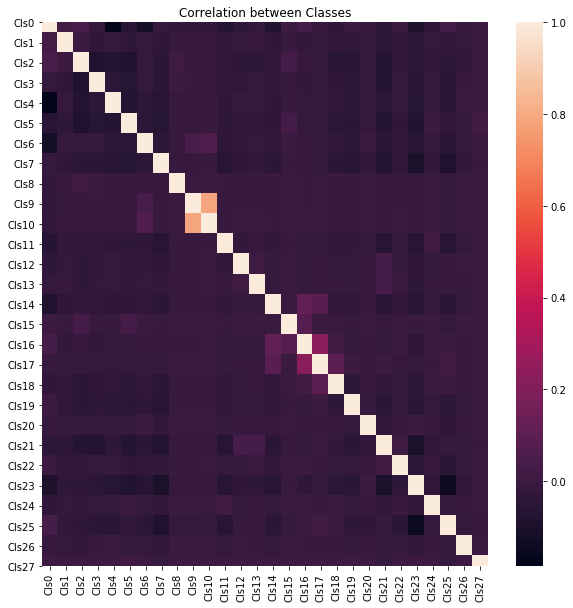
The number of classes that exist or coexist in a training image were counted and the corresponding percentage distribution is shown in Figure 3-4. This figure shows that about 50% of images only have one type of protein. About 40% of training images have two class labels. The images having 3 or more class labels are relatively rare.



**Figure 3‑4 Distribution of Number of Classes in One Image**

* **Correlation between Class Labels**

The purpose of this exploration is to examine which pair of classes more likely coexist in a Human cell image. The correlation between two class columns were found and visualized in the heat map below (Figure 3-5). The correlation matrix shows that Classes 9 and 10 (that is, Endosomes and Lysosomes) have a higher correlation, which means that they are usually located in the same image. Classes 16 and 17 (i.e., Cytokinetic bridge and Mitotic spindle) have certain correlation.



**Figure 3‑5 Correlation between Protein Types**

## **Image File Exploration and Visualization**

* **Check If Each Image ID Have 4 Images**

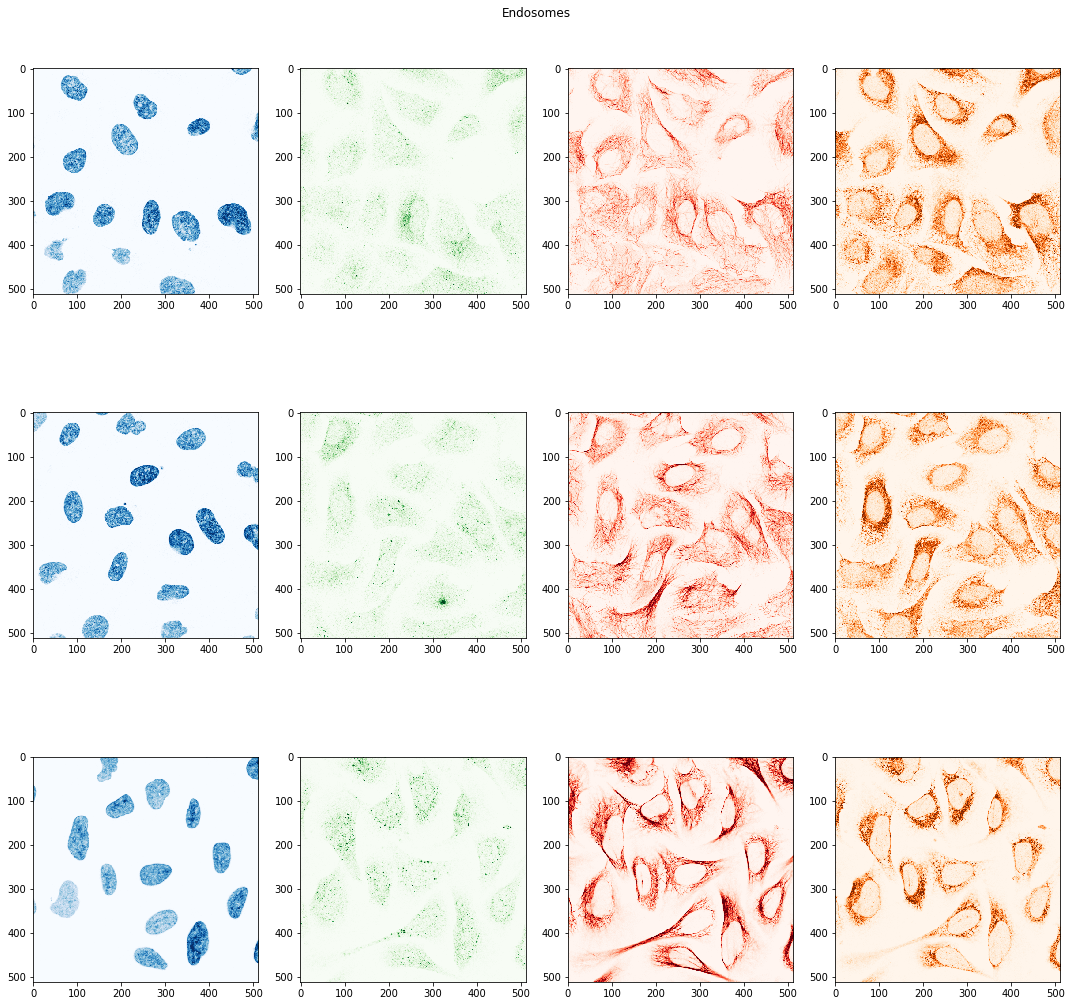
Each image id in Human Atlas Image dataset include 4 images that are corresponding to 4 color channels, red, green, blue, and yellow, respectively. A total number of 62,144 image ids is listed in train.csv file and there are 124,288 (that is, 62,144x4) images in the training image folder. It means that no images are missing. Similarly, it is found that all the 23,404 image ids listed in the submission csv file have 4 corresponding single-channel images.

* **Check If Any Image File is Corrupted**

Codes were written to open each image file and check if any image file is corrupted. The check results indicate that every image file is good.

* **Image Visualization**

Example of images were loaded to program and visualized. Three samples of Class Endosomes are visualized in Figure 3-6. As shown in this figure, these three samples have some similar patterns but some differences can also be observed



**Figure 3‑6 Three Samples of Protein Cells with a Class of Endosomes**

# **METHODOLOGY**

Human protein image classification is a multi-label classification problem. This study used deep learning algorithms, specifically basic convolutional neural network (CNN) and its enhancement, to identify mixed protein types. The deep learning model were implemented using PyTorch packages. Figure 4-1 illustrates the steps that have been taken in this project.

Model Application

Model Development

Image Conversion

Data Sampling

Image Transformation and Loading

CNN Model Creation

Performance Metrics

Model Train and Prediction

Results Visualization

Data Preprocessing

**Figure 4‑1 Structure of Study Methodology**

## **Data Preprocessing**

The Challenge dataset downloaded from the Kaggle website is a very large data set, including more than 85,000 images. A number of steps were taken in this project to reduce dataset size and prepare for model development.

* **Image Conversion**

The images provided by the Challenge are obtained using advanced microscopy system. These images provide spatial distribution of proteins with a human cell. Each image of cell is split into four single-channel images using green, red, blue, and yellow filters. Note that only single-channel images are provided in the raw dataset. There are two options to load these images. The first one is to load four single-channel images separately in the model, while the second option is to combine these single-channel image to be one RGBY image and only load the combined image. Considering the speed of running model, it was decided in this study to combine images to RGBY image first.

* **Data Sampling**

As mentioned above, the Challenge image dataset is extremely large. It takes several hours to run one case. Sampling images becomes very important in helping reduce running time especially in case of testing loss convergence on different models. But as the input is so imbalanced, we face the danger of losing all positive samples with some rear labels when decrease the input size. So, for the class label with very low occurrence, for example, less than 100 images, the images with those labels were kept in the training and testing dataset, while for those class labels with more frequent occurrence, only 10% of them were randomly selected. Instead of using k-fold cross validation that requires more running time, this study used the test-train-split method to generate train and test data sets. The split percentage is about 1/3. For smallest samples, the total number of training cases was reduced to 2,156 and the validation data set has a size of 1,081. We also tried 5000 images samples as train sets for better accuracy, and finally input all images for selected model to examine them.

* **Image Transformation and Loading**

The original image has a size of 512 x 512. To further reduce the size of training dataset, the Challenge images were resized to lower sizes (for example, 64, 128 and 256). These images were also normalized by a number of 255. Since the input images are four-channel images with multiple labels, the commonly used PyTorch ImageFolder command cannot be used. A customized Dataset class have to be defined. Training and validation images and class labels were then loaded through PyTorch dataloader.

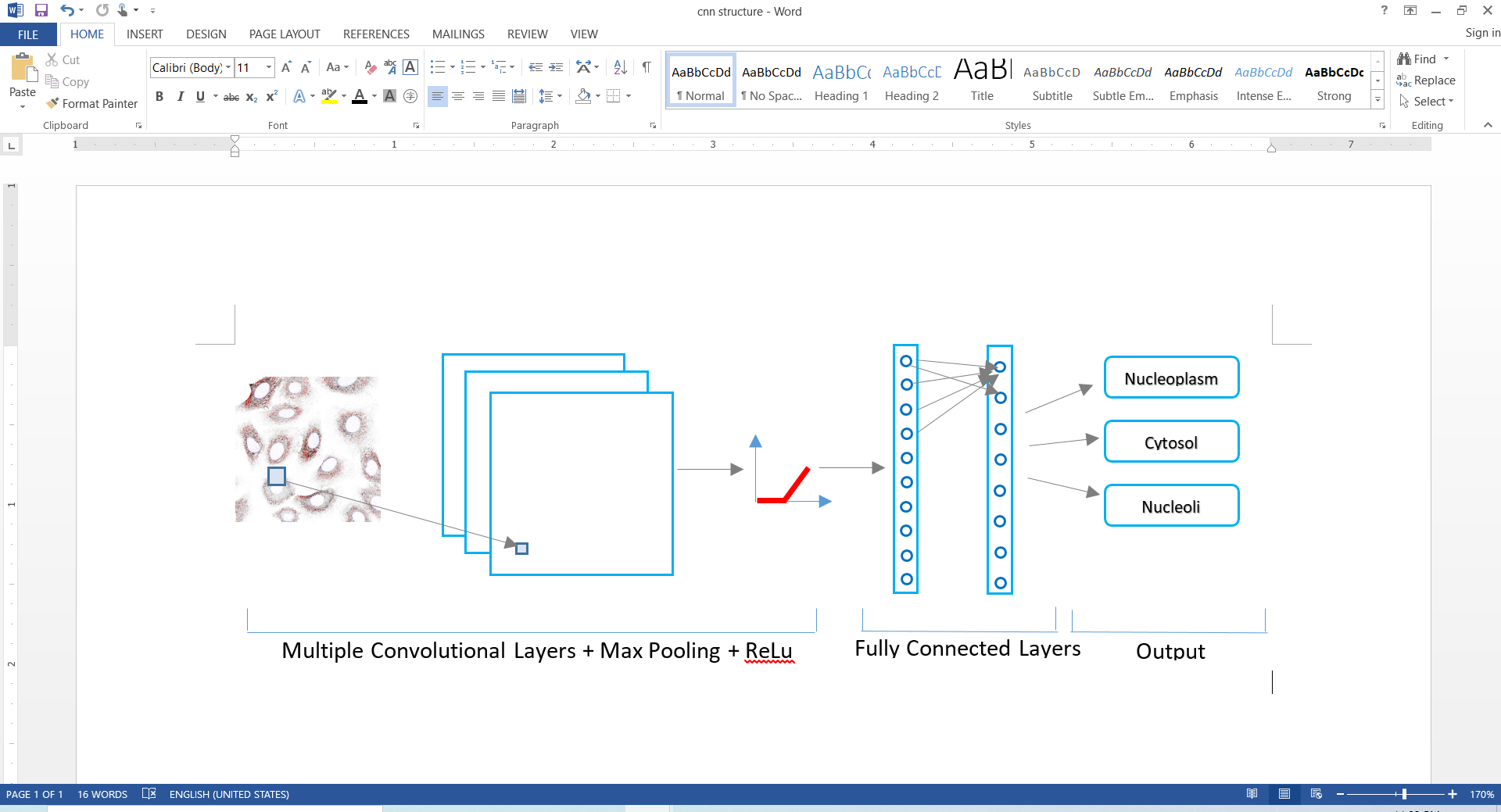
## **Model Development**

In this study, a basic convolutional neural network was developed as a base case. Transfer learning by using pretrained network such as AlexNet was also tested. Below are the descriptions of these models.

* Basic CNN Model

Figure 4-2 shows one example of CNN models used in this study. As shown in this figure, the input of CNN model are images in small batches. These images pass through convolutional layers where patches of images are extracted and applied to a same transformation using kernel. The output of convolutional layer is an output feature map, showing high-level features such as edge. The spatial size of the output of convolutional layer can be calculated using the following equation.

where W is the input image size, K is kernel size, P is padding at the border, and S is striding that controls the steps to move the filters.



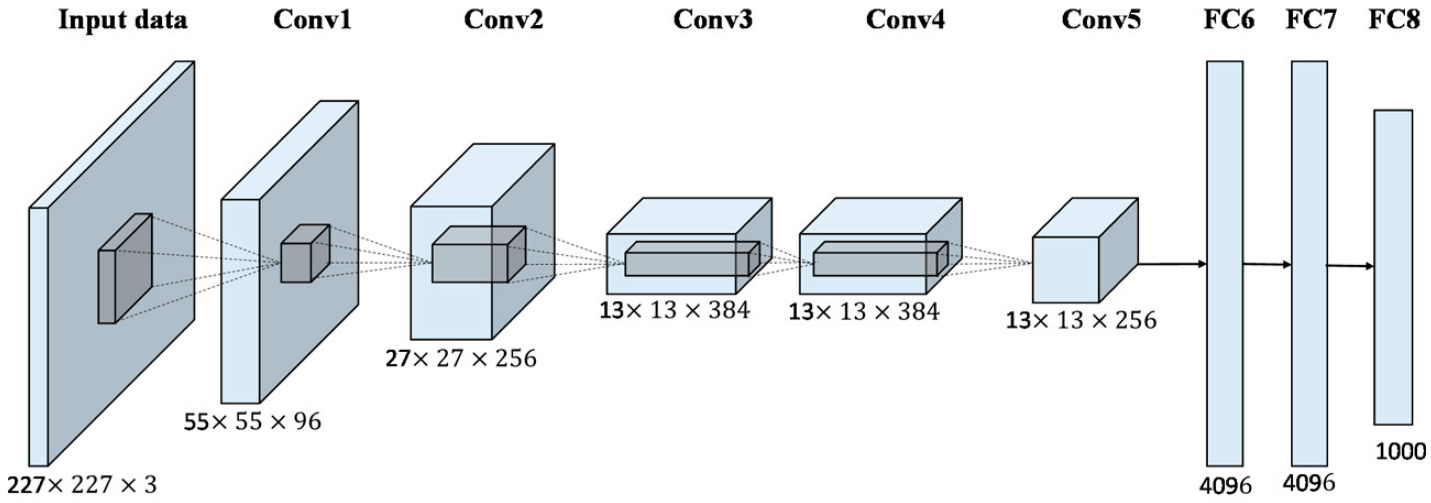
**Figure 4‑2 Structure of Basic CNN Model**

The convolutional layer is then followed by a maximum pooling layer and a Rectified Linear Unit (ReLu) layer. The pooling layer divides feature map into sub-region. The maximum values of each sub-region are output. The ReLu layer applies ReLu activation function. The output of convolutional layers are then flattened and connected two fully connected layers. The final class labels are determined by sigmoid function. Figure 4-3 shows the corresponding python code for CNN structure.



**Figure 4‑3 Python Code for CNN Model**

* Pretrained AlexNet

AlexNet did a great success in 2012, and the improvement it did was a huge leap in deep learning. But its structure seems to be quite concise compared with larger and deeper natural networks which appeared later. It contains eight layers; the first five were convolutional layers, some of them followed by max-pooling layers, and the last three were fully connected layers. It used the non-saturating ReLU activation function, which showed improved training performance over tanh and sigmoid. In fact, we built our basic model patterning AlexNet but simplified it and add sigmoid function on the output.

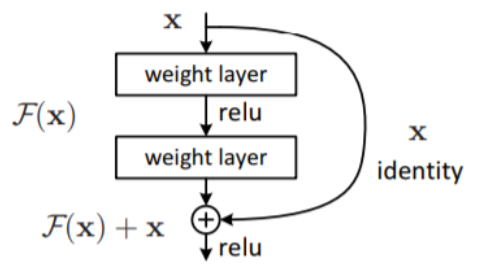
**Figure 4‑4 Structure of AlexNet Model**

**(**<https://www.mdpi.com/2072-4292/9/8/848/htm>**)**

We choose AlexNet as a test case because it removed sigmoid or softmax function on the output side and we want to know whether this accelerate the converge of loss function. In our test we use pretrained AlexNet model in pytorch, vision0.6.0 and modified the first and last layer to make it runnable for our data.

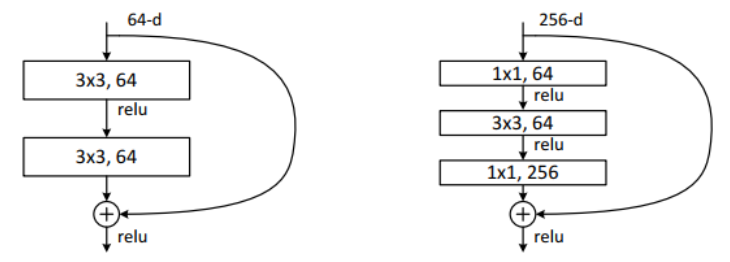
* Pretrained Resnet

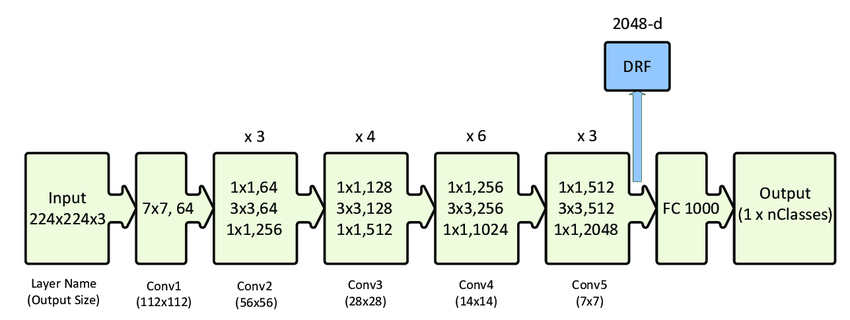
Residual neural network (ResNet) was introduced on motivation of skipping over layers is to avoid the problem of vanishing gradients. When the depth of the neural network is continuously deepened, there will be a degradation problem, that is, the accuracy will first rise and then reach saturation, after which increasing the depth will cause decrease of accuracy. ResNet reuses activations from a previous layer until the adjacent layer learns its weights, in this way it ‘skips’ between layers, reducing the impact of vanishing gradients.

Residual Block is the vital components in ResNet. Residual means the difference between the actual observation value and the estimated value (fitted value). Set the input of a certain residual block is x, and the fitted output is H(x). If we directly pass the input x to the output as the observation result, then the residual we need to learn is F(x)=H(x)−x. We calculate the parameters as follows:

,

It is easier to optimize with residuals: the mapping after introducing residuals is more sensitive to changes in output. ResNet has many bypass branches that can directly connect the input to the following layers, so that the latter layers can directly learn the residuals, simplifying the difficulty of learning, and can protect the integrity of the information.

In ResNet50, the two 3x3 convolutional layers are replaced by 1x1 + 3x3 + 1x1 convolution for calculation optimization. The middle 3x3 convolutional layer in the structure first reduces the calculation under a dimensionality reduction 1x1 convolutional layer, and then restores it under another 1x1 convolutional layer, which not only maintains the accuracy but also reduces the amount of calculation. This structure is called the bottleneck module. ResNet50 contains 16 bottlnecks in total as the follow picture shows.



**Figure 4‑4 Structure of ResNet50 Model**

(<https://www.researchgate.net/figure/ResNet-50-architecture-26-shown-with-the-residual-units-the-size-of-the-filters-and_fig1_338603223>)

We apply the pretrained ResNet50 model in pytorch and change the cannel number in first conv2d layer from 3 to 4 while keeping the weights, suggested by pongba in Kaggle(<https://www.kaggle.com/zhugds/resnet34-with-rgby-fast-ai-fork>).

The CNN model weights are optimized by minimizing loss function. For a multi-label problem, Binary Cross Entropy(BCE) is one of the commonly used loss function. The equation below shows the equation to calculate binary cross entropy.

where m is sample size. The stochastic gradient descent optimizer with momentum and L2 penalty in PyTorch was used in this study to find optimal weights.

Focal Loss for Dense Object Detection is the Best student paper of ICCV2017. The idea of ​​the article is very simple but very pioneering. The introduction of Focal Loss is mainly to solve the imbalance in the number of difficult and easy samples based on the concept of Cross-Entropy(CE).

In order to solve the problem of imbalance of positive and negative samples, we usually add a parameter in front of the cross-entropy loss :

Although the positive and negative samples are balanced, it does not help the imbalance of difficult and easy samples. The author of this article believes that easy-to-divide samples (ie, samples with high confidence) have very little effect on the model improvement, and the model should mainly focus on those difficult-to-divide samples. Thus, they introduced another parameter to reduce the weight of easy-to-divide samples in loss calculations:

Their experiments show that the best performance occurs when =2 and =0.25.

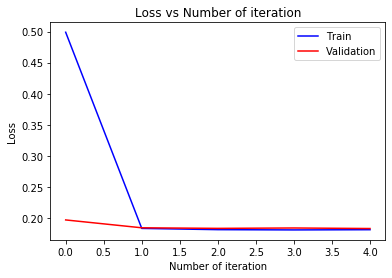
The performance of model is mainly quantified by macro F1-score as required by the Kaggle Challenge. Macro F1-score is the average of class-wise F1-scores. The expression for macro F1-score is shown below.

Where i represents the ith class. In addition to macro F1-score, the accuracy of the developed model as well as sample-based F1-score (that is, calculate F1-score for each sample and then find the average) were also calculated in this study.

Since the number of input images are large, it is more preferable to run a smaller number of epochs and check the results. So, codes are written to save intermediate model. When needed, the saved model can be reloaded, and the training process can be resumed.

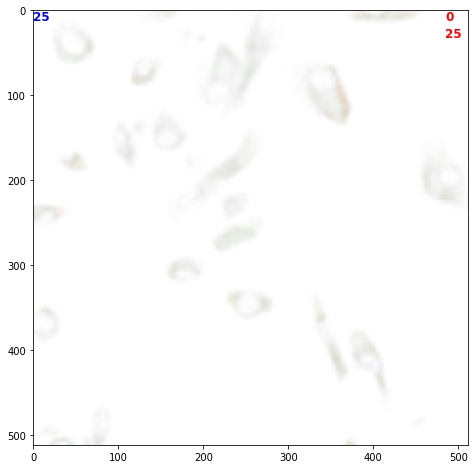
## **Model Application and Results Visualization**

The developed model in the previous section were applied to train input images and compare with validation dataset to quantify the model performance. To help check the values of metrics, the variation of loss, accuracy, sample-based F1-socre, macro F1 score were plotted. Figure 4-4 shows an example of metric visualization.

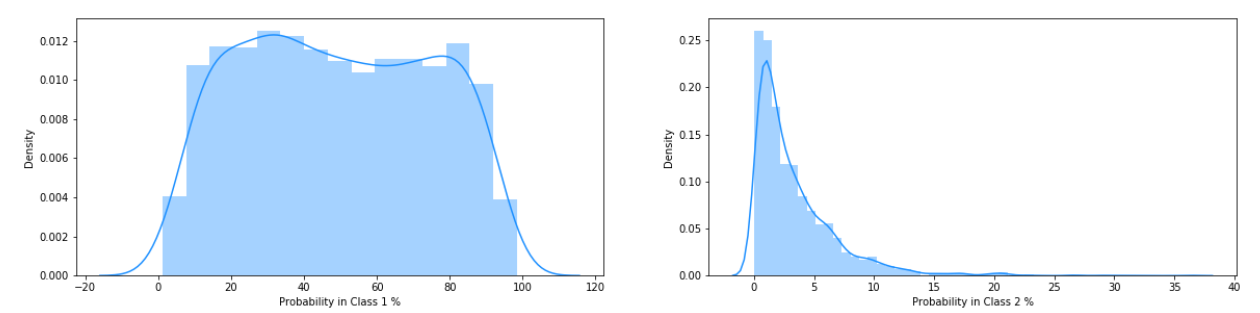


**Figure 4‑4 Example of Output Metrics Visualization**

Figure 5-2 shows an example of visualization of specific image for its class labels. The number at the left top corner shows the actual class label for the image, while the numbers at the right top corner are the predicted class labels.



The density of output probabilities for specific class is also important as the input images are so imbalanced. If it is a bimodal distribution that means our model already learned something to classify positive or negative results. But unfortunately, we got a lot of Unimodal distribution from trained models with simple architectures or smaller input size, which means it is nearly hopeless for these models to predict.



# **RESULTS AND DISCUSSION**

This section lists and analyzes the results that have been obtained in the study. A basic CNN model with two convolutional layers was used as base case in this study. An initial run of this base case resulted in a macro F1-score of 0. A number of strategies have been tried in this study to improve the model classification performance. This section summarizes the strategies and parameters that have been used and their corresponding results.

* **Impacts of Image Augmentation**

The default case only used transformation of resize(128) for images. Additional image augmentation methods were applied to images for testing purpose, including changing size, adding brightness and contrast, rotating images, and randomly horizontal flip. Table 5-1 shows the sensitivity analysis results for validation dataset. It is seen from Table 5-1 that image augmentation has very limited impacts on model performance.

**Table 5‑1 Impacts of Image Augmentation Using Transforms**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Strategy** | **Parameter** | **Accuracy** | **Sample-Based F1-Score** | **Macro F1-Score** |
| **Resize image** | Resize(64) | 0.91651 | 0.33346 | 0.03450 |
| Resize(128) | 0.91651 | 0.33346 | 0.03450 |
| Resize(256) | 0.91651 | 0.33346 | 0.03450 |
| **Add brightness and contrast** | Brightness=0  Contrast=0  Hue=0  Saturation=0 | 0.91651 | 0.33346 | 0.03450 |
| Brightness=0.1  Contrast=0.1  Hue=0.1  Saturation=0.1 | 0.91651 | 0.33346 | 0.03450 |
| Brightness=0.25  Contrast=0.25  Hue=0.1  Saturation=0.1 | 0.91654 | 0.33346 | 0.03451 |
| Additional image transform | Brightness=0.25  Contrast=0.25  Hue=0.1  Saturation=0.1  Rotation=30  Random horizontal flip | 0.91654 | 0.33346 | 0.03451 |

* **Impacts of Threshold Setting**

Usually, the probability of 0.5 is selected as a threshold for model output to determine if a sample belongs to a class. However, for very imbalanced classes, its probability may never reach to 0.5. To overcome such imbalance cases, some studies participating the Kaggle Challenge have used 0.15, 0.2 to 0.3 as thresholds. Instead of using the same threshold for all the class labels. This study also tried to use the occurrence percentage of each class label in training dataset as thresholds. Table 5-2 lists the analysis results for threshold selection. Compared to image augmentation, the selection of threshold has a greater impact on model performance.

**Table 5‑2 Impacts of Threshold Selection**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Strategy** | **Parameter** | **Accuracy** | **Sample-Based F1-Score** | **Macro F1-Score** |
| **Threshold Selection** | 0.5 | 0 | 0 | 0 |
| 0.3 | 0.91750 | 0.32525 | 0.03380 |
| 0.2 | 0.91651 | 0.33346 | 0.03450 |
| 0.15 | 0.89362 | 0.31321 | 0.04146 |
| Threshold varies for each class label based on their occurrence percentage in training dataset | 0.3671 | 0.13987 | 0.08708 |

* **Impacts of Modeling Structure**

Different numbers of convolutional layers as well as pretrained network structure including AlexNet and Resnet were tested in the study. The table below shows their impacts.

**Table 5‑3 Impacts of Modeling Structure**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Strategy** | **Parameter** | **Accuracy** | **Sample-Based F1-Score** | **Macro F1-Score** |
| **Network Structure**  **(with BCE loss)** | 2-layer Convolutional Layer | 0.94049 | 0 | 0 |
| 3-layer Convolutional Layer | 0.94034 | 0.00046 | 0.00017 |
| Pretrained AlexNet | 0.94202 | 0.00185 | 0.00219 |
| Pretrained Resnet50 | 0.94627 | 0.22379 | 0.03286 |

We decide to train some simple models by small input size (around 2000 in train set and 1000 in test set)to see whether these structures could work. Our 2-conv-layer or 3-conv-layerbase model as well as AlexNet do not performed well as their output was seriously ‘converge’ to one value ,almost with no distribution.

The follow figure is the probability distribution by validation output in base model testing. Most of the predicted results on one label by different input are almost the same: In class 1 ‘Nucleoplasm’, the minimum is 0.38 and max is 0.39.

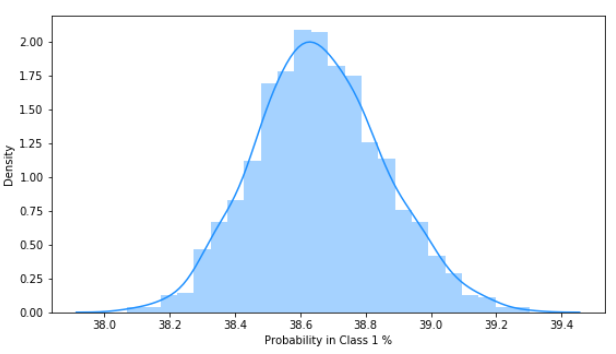


Figure: Output ‘converge’ to one value

We found that the final converged value is almost the same to the percentage of different labels:

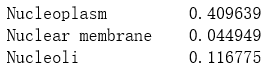
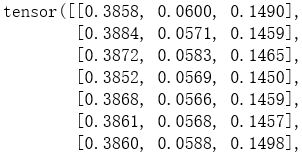
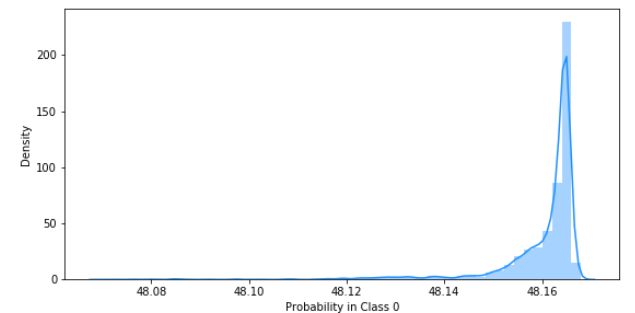


Figure: Left: output for first 3 labels. Right: occurrence of first 3 labels

AlexNet do no perform any better. Because the output of AlexNet is not between 0 and 1 we use BCE-with-LogistsLoss() function to calculate BCE loss. A learning rate scheduler was applied to reduce learning rate on latter epochs. Out of our expectation the convergency of output data is even worse:



The loss of these two models seems not to change after some iterations:

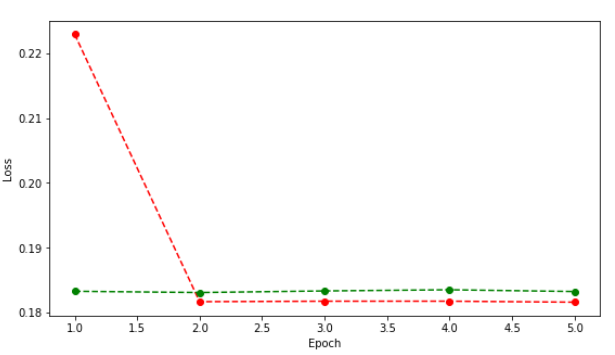


Figure: Base model Loss in each epoch

Basically, we get the conclusion that such simple models are not able to process such challenging problem.

ResNet is much more complex then the formers and it take more time running. The performance of ResNet seems to be much better: The accuracy is larger than 0.9403, the accuracy with all zeros output – which means it really begin to classify something. Though the Macro F1 score is still less than 0.1 but is already much better.

We could see that the density distribution of output is much better, although there are much noises, the value varies from 0 to 1 for class 1 ‘Nucleoplasm’:

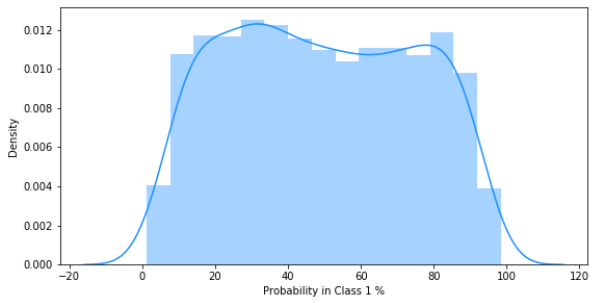


Figure: density distribution of predictions of class 1 in Resnet

Though convergence still occur in some other labels, the total output now is much more make sense:

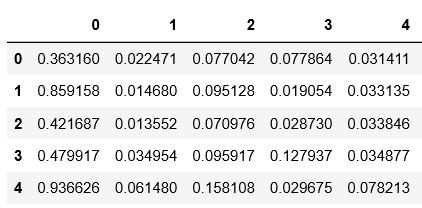


Figure: Examples of output matrix of Resnet

Thus, we believe that only ResNet is capable to work in classifying these trunks of Proteins. We will mainly use ResNet in latter performance tests.

* **Impacts of Loss Function**

Two types of loss functions have been used in the study, one is binary cross entropy and another one is focal loss. Table 5-4 listed the corresponding results.

**Table 5‑4 Impacts of Loss Function**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Strategy** | **Loss Function** | **Accuracy** | **Sample-Based F1-Score** | **Macro F1-Score** |
| **Base Model** | Binary Cross Entropy | 0.94034 | 0.0 | 0.0 |
| Focal Loss Function | 0.94194 | 0.00309 | 0.00076 |
| **ResNet50** | Binary Cross Entropy | 0.94627 | 0.22379 | 0.03285 |
| Focal Loss Function | 0.94801 | 0.28371 | 0.04592 |

This time we enlarge the sample size to about 5000 to test performance of BCE loss and Focal Loss. We expected that Focal loss could reweight the imbalanced samples and give better results, but for simple base models Focal Loss do not work well, either.

The result of base model with BCE loss is quite random and possibly be zero. With Focal loss function, the total performance seems to be unchanged, however, we found that Focal loss effectively reduce the converge of output value. The data now varies much more.

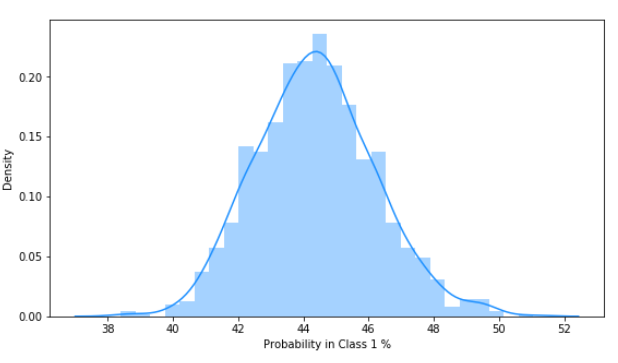


Figure: output for predicted label 1 in base model with Focal Loss

This is more like normal distribution, though. We guess that Focal loss expanded the loss when update the gradient so much noise is added in the result, and this somewhat reduce the converge.

In Resnet, Focal loss become to make good changes: accuracy and F1 scores are improved. The density distributions of output begin to perform ‘two-side’, not converged or averaged.

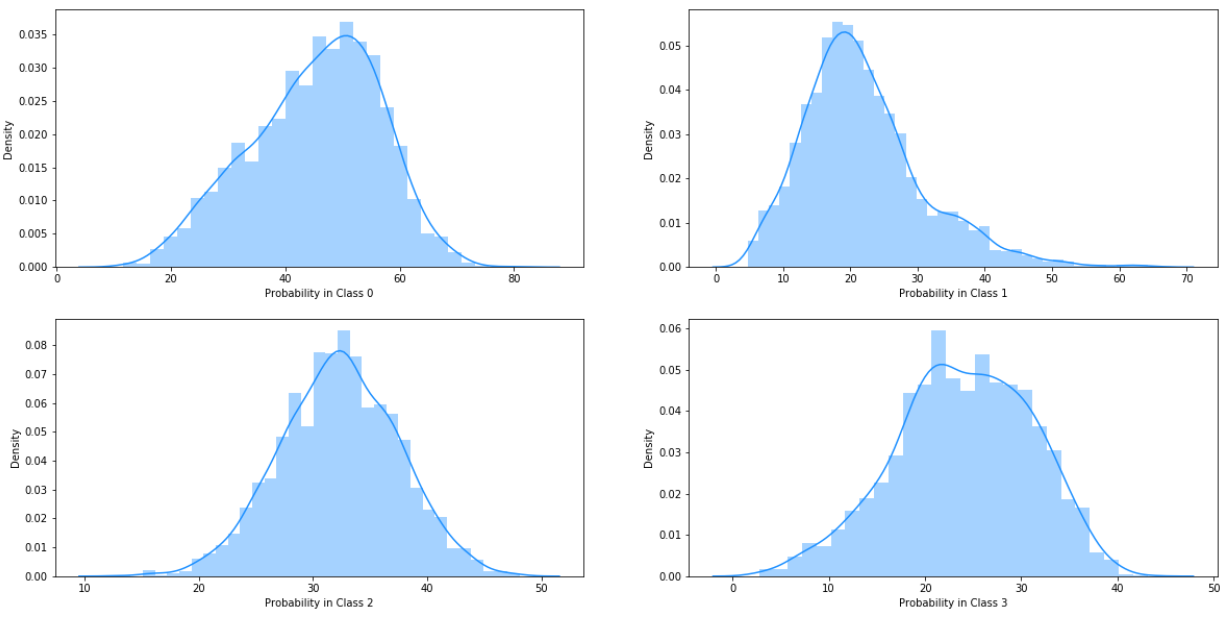


Figure: Some of the density distribution of output by ResNet with Focal loss

# **CONCLUSIONS AND RECOMMENDATIONS**

The Human Protein Atlas hosted an image classification challenge at the Kaggle website. The Human Protein Atlas hosted an image classification challenge at the Kaggle website. A convolutional neural network model was developed in this project to identify mixed protein pattern in human cells using images provided by the challenge. Different strategies such as image augmentation, learning parameter adjustment, loss function definition, network structures, and threshold configuration have been explored in this project to improve model prediction performance.

# **REFERENCE**

Ezat, W. A., M. M. Dessonuky, and N.A. Ismail. (2020). “Multi-class Image Classification Using Deep Learning Algorithm.” Journal of Physics: Conference Series. pp. 1447.

Jain, T. (2020). “Basics of Image Classification Techniques in Machine Learning.” <https://iq.opengenus.org/basics-of-machine-learning-image-classification-techniques/>. Accessed on July 28, 2020.

Kaggle. (2020). “Human Protein Atlas Image Classification.” <https://www.kaggle.com/c/human-protein-atlas-image-classification/overview>. Accessed on July 28, 2020.

Kaggle. (2020). <https://www.kaggle.com/c/human-protein-atlas-image-classification/discussion/73246>. Accessed on July 28, 2020.

Kaggle. (2020). <https://www.kaggle.com/c/human-protein-atlas-image-classification/discussion/77269>. Accessed on July 28, 2020.

Pärnamaa, T. and L. Parts (2017). “Accurate Classification of Protein SSubcellular Localization from High-Throughput Microscopy Images Using Deep Learning.” *G3 Genes | Genomes | Genetics*. Volume 7, pp. 1385-1392.

Rumetshofer, E., M. Hofmarcher, C. Rohrl, and S. Hochreiter. (2019). “Human-Level Protein Location with Convolutional Neural Networks.” Conference Paper at ICLR 2019.

Yang, W., C. F. Winsnes, M. Hjelmare, and et al. (2019). “Analysis of the Human Protein Atlas Image Classification Competition.” *Nature Methods.* Vol. 6, pp. 1254-1261.

Focal Loss <https://arxiv.org/abs/1708.02002>

ResNet <https://arxiv.org/abs/1512.03385>