EE5934: Deep Learning

Project # 2
G7

Name: Liu Xingyu **Matric Number: A0116430W**

Name: Soe Moe Kyaw **Matric Number: A0195003N**

Name: Thein Than Khaing **Matric Number: A0213354N**

Table of Contents

Acknowledgement 3

1. Introduction 4

1.1. Background Information 4

1.2. Weakly Supervised Classification 4

1.3. Competition Objective 4

2. Analysis of Training Data 5

2.1. Analysis of train.csv 5

2.2. Visualization of Training Images 6

2.2.1. Visualization of Each Channel (R, B, Y, G) 7

2.2.2. Visualization of 4 Channels in One Single Image 7

2.2.3. Visualization of Each Label 8

3. Training Model and Processes 10

3.1. Training Model 10

3.1.1. EfficientNets 10

3.1.2. Modifying EfficientNet Model to Fit with Data 13

3.2. Training Processes 13

3.2.1. Data Pre – processing 13

3.2.2. Training 14

4. Inference Results and Discussion 18

5. Future Improvements 20

Reference 21

Table of Figures

Figure 1: Image ID and cell labels....5
Figure 2: Image files with color tags....5
Figure 3: Label and associated organelle description....6
Figure 4: Label Distribution....6
Figure 5: First Images of each Channel....7
Figure 6: Blend 1: Yellow, Blue, Red (First 3 images)....7
Figure 7: Blend 2: Red, Green, Blue (First 3 images)....7
Figure 8: One Sample of Label 11....8
Figure 9: Plots of different types of organelle....9
Figure 10: Comparison of EfficientNets with Other Convolutional Neural Networks [2]....10
Figure 11: 5 modules [3]....11
Figure 12: Sub – block 1, Sub – Block 2 and Sub – block 3 [3]....11
Figure 13: Architecture of EfficientNet – B0 [3]....12
Figure 14: Modified output layer....13
Figure 15: Blend 1 Y, B, R....14
Figure 16: Blend 2 R, G, B....14
Figure 17: Training Results using YBR 3 – channel images for 10 epochs....16
Figure 18: Training Results using RGB 3 – channel images for 15 epochs....16
Figure 19: Notebook scores using different weights....18
Figure 20: Final Kaggle Submission....19

Table 1: EfficientNet-B0 baseline network....12
Table 2: Number of Channels for family models [3]....12
Table 3: Efficientnet Models Parameters....13

Acknowledgement

We all three would like to express our gratitude to our **Professor Feng Jiashi** and our **Teaching Assistant, Wang Jiadong** for giving the opportunity to participate in Kaggle Competition. Because of this opportunity, we gained a wide range of experience in deep learning and cell segmentation. We would like to express our gratitude to **Kaggle** as well for providing us the Python notebook environment and free GPU usage.

1. Introduction

1.1. Background Information

This project is based on Kaggle 'Human Protein Atlas – Single Cell Classification' competition. The goal of the competition is to annotate subcellular localizations of proteins in large collection of individual cells images. By achieving high accuracy on automated annotations, it will be beneficial to scientific and medical community when trying to understand how cells functions and how diseases develop.

1.2. Weakly Supervised Classification

This project is a weakly supervised multi-label classification problem. The labels provided for training are image level labels while the model is supposed to predict cell level labels. As the labels given for each image is a collective label for all the cells in the image, it does not necessarily mean that each cell can be applied with the given label. Hence, this imprecise labelling is referred to as 'weakly supervised classification problem'.

1.3. Competition Objective

The main objective is to predict organelle localization labels for each cell in the image. The team task is to train a neural network which can segment cells in an image of proteins under the microscope while using other images as references.

The remaining of this report is organised as followed: **Section 2** is analysis of training data (Competition Data). Training Model and Processes will be covered together with result analysis in **Section 3**. In **Section 4**, inference results and discussion will be performed. **Section 5** will cover conclusion and further improvements.

2. Analysis of Training Data

A large collection of test and training data around 160GB along with training labels are provided for this competition. In this section, analysis of these data is performed as follows: *analysis of train.csv*, *labels distribution*, and *visualization of training images*.

2.1. Analysis of train.csv

In '*train.csv*' file, training file names and image level labels are provided. Each file name in the '*train.csv*' is associated with four image files in the training folder. The images are generated with multiple colors by using different filter on the protein patterns. Colors are red for microtubules, blue for nucleus, yellow for Endoplasmic Reticulum (ER), and green for protein of interest. The green filter is the main filter which should be used for test label prediction, while others can serve as reference.

Figure 1 shows the image data from the first 5 lines of train.csv file. In this train.csv file, there are 21806 samples and each sample consist of 4 files. The labels from this file are all image – level labels. So, there are altogether $21806 \times 4 = 87224$ image files. Each of these 4 files represents a different filter (red, blue, yellow, green) on the subcellular protein patterns represented by the sample. **Figure 2** represents the naming format of image files.

	ID	Label
0	5c27f04c-bb99-11e8-b2b9-ac1f6b6435d0	8|5|0
1	5fb643ee-bb99-11e8-b2b9-ac1f6b6435d0	14|0
2	60b57878-bb99-11e8-b2b9-ac1f6b6435d0	6|1
3	5c1a898e-bb99-11e8-b2b9-ac1f6b6435d0	16|10
4	5b931256-bb99-11e8-b2b9-ac1f6b6435d0	14|0

Figure 1: Image ID and cell labels

```
5e3a2e6a-bb9c-11e8-b2b9-ac1f6b6435d0_red.png
315a9edc-bbc6-11e8-b2bc-ac1f6b6435d0_yellow.png
437fa1ce-bb9f-11e8-b2b9-ac1f6b6435d0_yellow.png
8a51782e-bb9b-11e8-b2b9-ac1f6b6435d0_green.png
0df0c3aa-bbca-11e8-b2bc-ac1f6b6435d0_blue.png
bf0b3946-bbba-11e8-b2ba-ac1f6b6435d0_red.png
641a0682-bbb7-11e8-b2ba-ac1f6b6435d0_green.png
05d32f36-bba3-11e8-b2b9-ac1f6b6435d0_red.png
defc6fbe-bba0-11e8-b2b9-ac1f6b6435d0_blue.png
```

Figure 2: Image files with color tags

The complete list and detail explanation of labels representation can be found at https://www.kaggle.com/c/hpa-single-cell-image-classification/data. A summary of label and organelle representation is shown in **Figure 3**.

```
['0: nucleoplasm',
 '1: nuclear_membrane',
 '2: nucleoli',
 '3: nucleoli_fibrillar_center',
 '4: nuclear_speckles',
 '5: nuclear_bodies',
 '6: endoplasmic_reticulum',
 '7: golgi_apparatus',
 '8: intermediate_filaments',
 '9: actin_filaments',
 '10: microtubules',
 '11: mitotic Spindle',
 '12: centrosome',
 '13: plasma_membrane',
 '14: mitochondria',
 '15: aggresome',
 '16: cytosol',
 '17: vesicles',
 '18: negative']
```

Figure 3: Label and associated organelle description

Some useful statistics can be extracted from 'train.csv'. From the 'label' column, total 19 unique labels: 0 to 18 can be found. We can also learn distribution among different label categories. In this chart below, blue represents the number of times a unique label represents an image in the training data and orange represents overall images of each labels.

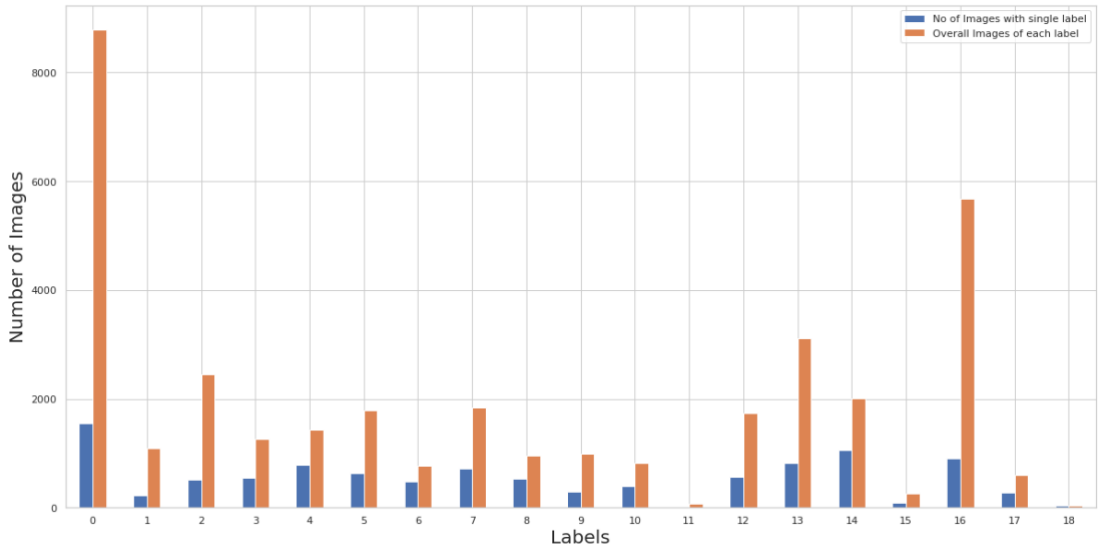


Figure 4: Label Distribution

As shown in **Figure 4**, number of images with one label and overall images of each label for label 11, label 15 and label 18 are too small compared to other images. As the competition is based on multilabel classification, these three labels will be difficult to predict.

2.2. Visualization of Training Images

In this subsection, visualization of each channel, visualization of 4 channels in one single image by blending and visualization of each label are performed.

2.2.1. Visualization of Each Channel (R, B, Y, G)

Images of each channel (Red: Microtubules, Blue: Nucleus, Yellow: Endoplasmic Reticulum (ER), Green: Protein of interest) are visualized, and the result is shown in **Figure 5**.

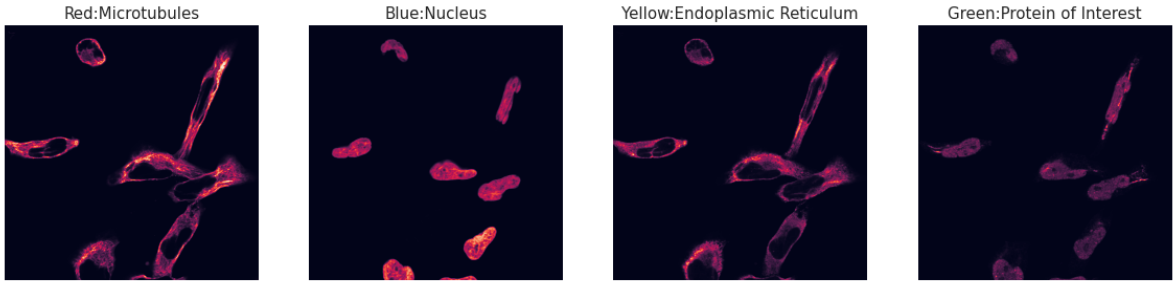


Figure 5: First Images of each Channel

2.2.2. Visualization of 4 Channels in One Single Image

To display 4 channels (R, B, Y, G) in one single image, 3 channels are blended. In this report, Yellow, Blue and Red blend (Blend1) and Red, Green and Blue blend (Blend2) are used for both training processes. In **Figure 6**, first 3 images of Blend 1 are shown, and **Figure 7** represents first 3 images of Blend 2.

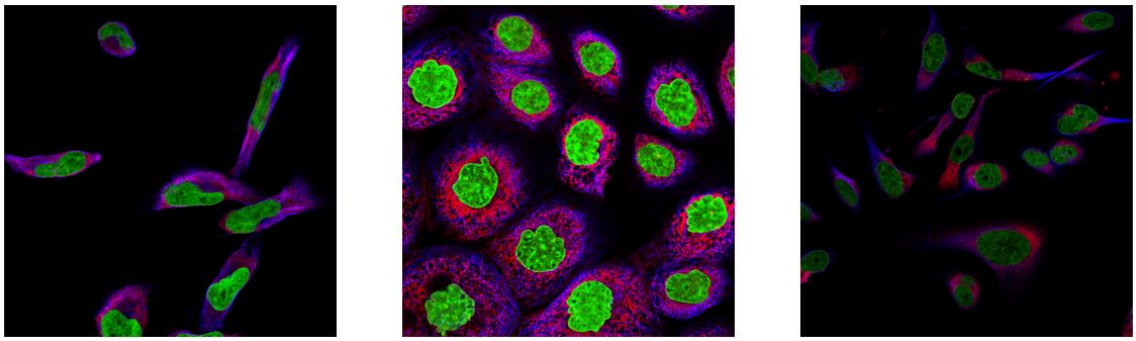


Figure 6: Blend 1: Yellow, Blue, Red (First 3 images)

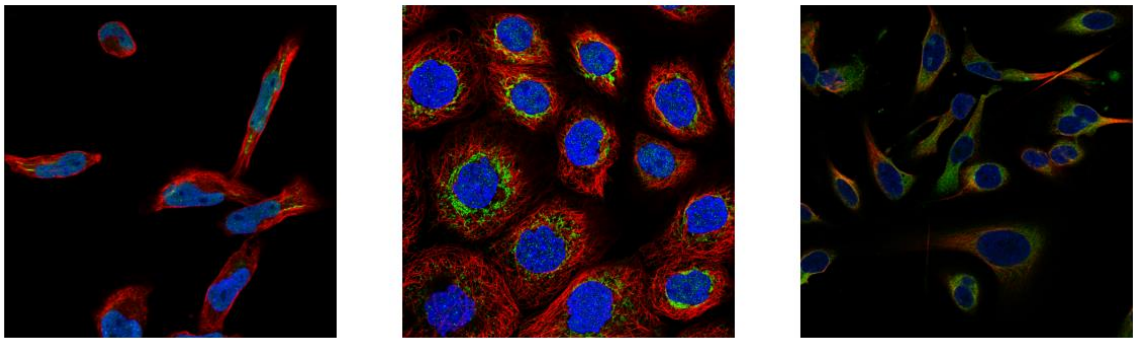


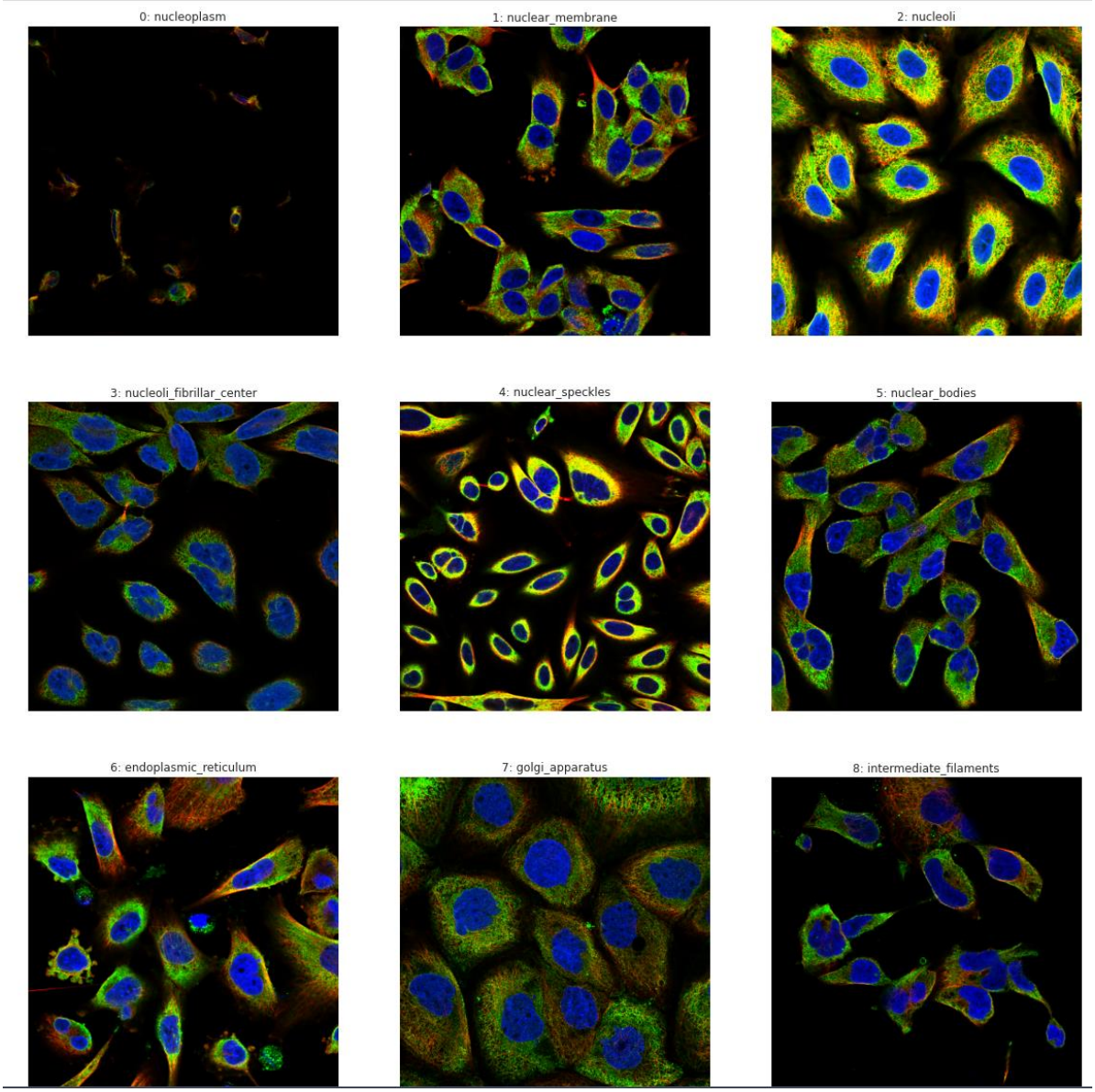
Figure 7: Blend 2: Red, Green, Blue (First 3 images)

2.2.3. Visualization of Each Label

For visualization of each label, Red channel, Blue channel, and Yellow channel are combined into one image. As for visualization purpose, only images with single label are used here. However, only single label of each channel of first location are stacked because as shown in **Figure 4**, single label image of label 11 is only one image. Sample of that one image can be seen as follows:

	ID	Label
0	b6a469d8-bbad-11e8-b2ba-ac1f6b6435d0	11

Figure 8: One Sample of Label 11



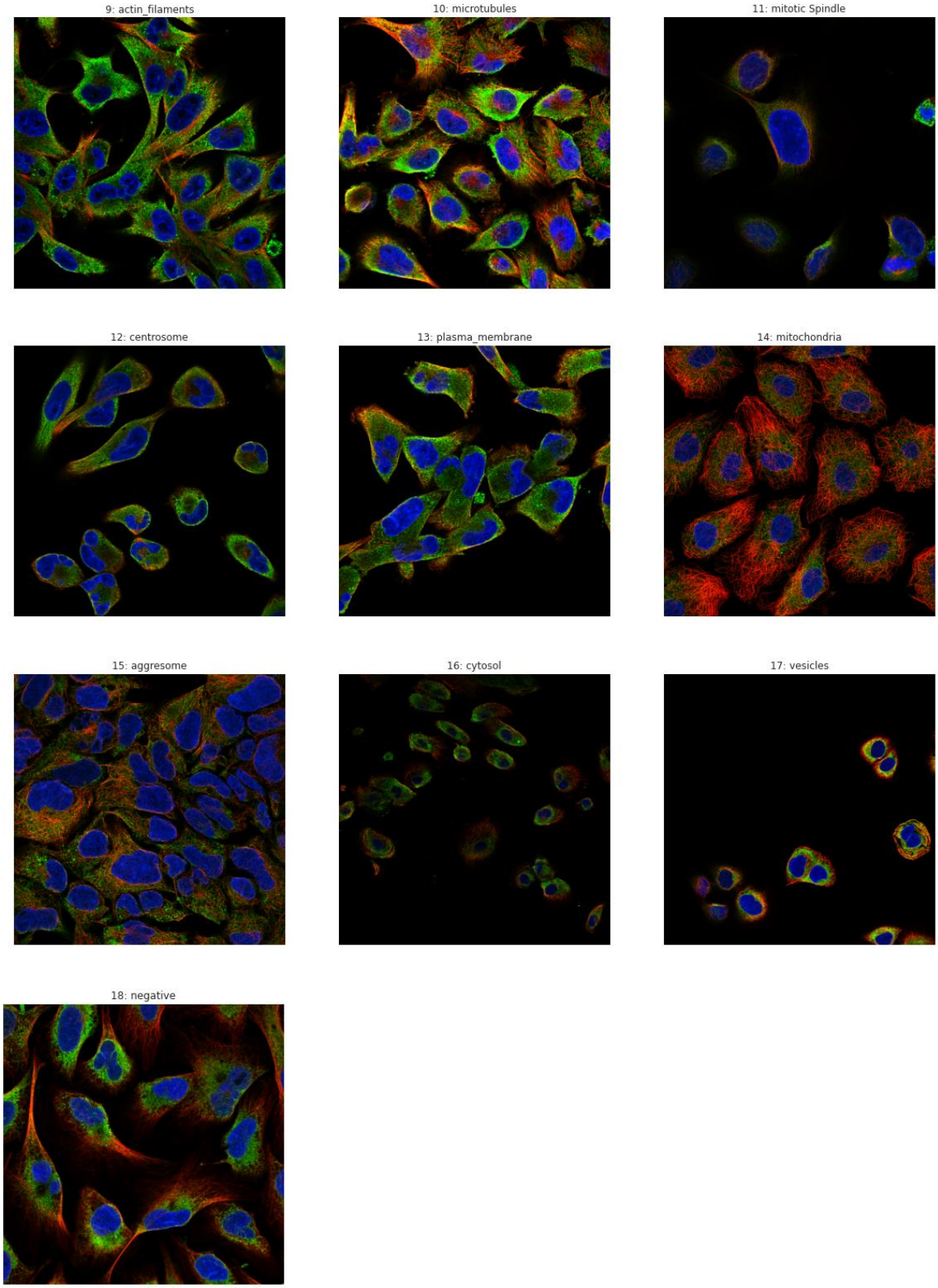


Figure 9: Plots of different types of organelle

3. Training Model and Processes

In this section, training model and training processes will be explained.

3.1. Training Model

As a training model, pretrained EfficientNet-B0 from EfficientNets [1] is used. This EfficientNets implementation in Pytorch is the same with the one in Tensorflow. Before using EfficientNet, data of 10k images of three channels combination were trained using ResNet101 and ResNet50, it took 23 hrs for ResNet101 to train for 5 epochs and 21 hrs for ResNet50 by using mmdetection. For that model, weights are from coco dataset and accuracy is around 78%. Training data size of competition is too large if three channels are combined even though image sizes are reduced. For these reasons, EfficientNet is decided to use.

3.1.1. EfficientNets

EfficientNets are a family of image classification models which can provide faster and better accuracy than other previous Convolutional neural networks. By doing a new scaling method that uniformly scales all dimensions of depth/width/resolution using a simple highly effective compound coefficient, EfficientNet is obtained. To get a family of models, a new baseline network is scaled up. [2] To develop a mobile-size baseline network, AutoML Mobile Framework is used, and it is named as EfficientNet-B0. To get its family models, that baseline network is scaled up using compound scaling method. They are altogether 7 models, EfficientNet – B0 to EfficientNet – B7. [1] The comparison of EfficientNets with other convolutional networks is shown in **Figure 10** below.

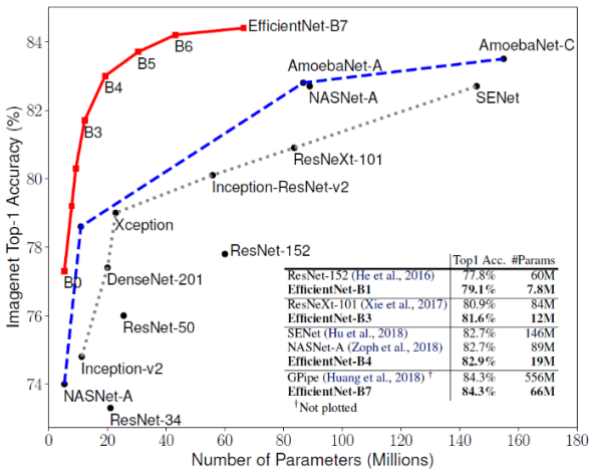


Figure 10: Comparison of EfficientNets with Other Convolutional Neural Networks [2]

Total number of layers in EfficientNet – B0 is 237 and EfficientNet – B7 is 813. All of these layers are made from 5 modules. [3]. These 5 modules are shown in the following **Figure 11**.

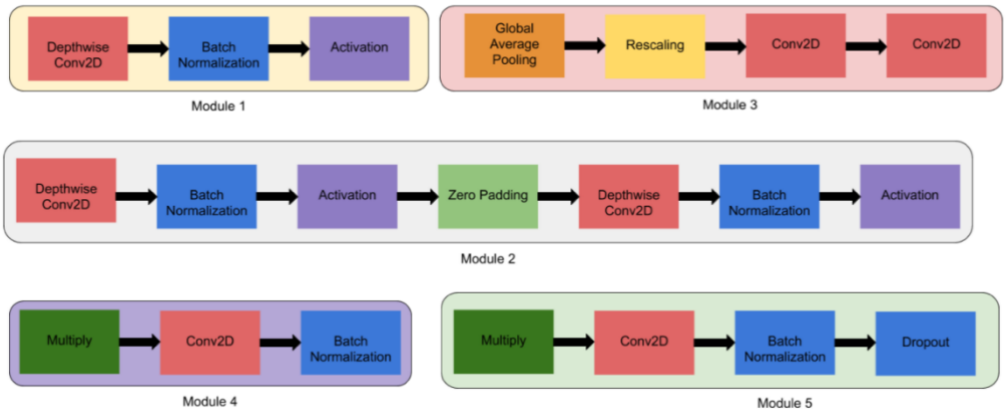


Figure 11: 5 modules [3]

Module 1 is used as starting point for the sub – blocks, and the first sub – block of all the 7 main blocks except first one uses Module 2 as a starting point. Module 3 is for a skip connection to all the sub – blocks, and to combine skip connection in the first sub – blocks is done by Module 4. Skip connections between each sub – block and its previous sub – blocks are combined using Module 5 [3].

To form sub – blocks, these modules are all combined and how they will be used are shown in **Figure 12**. Sub – block 1 is used as the first sub – block in the first block, and for the first sub – block in all the other blocks, sub – block 2 is used. Sub – block 3 is used for any sub – block except the first one in all the blocks [3].

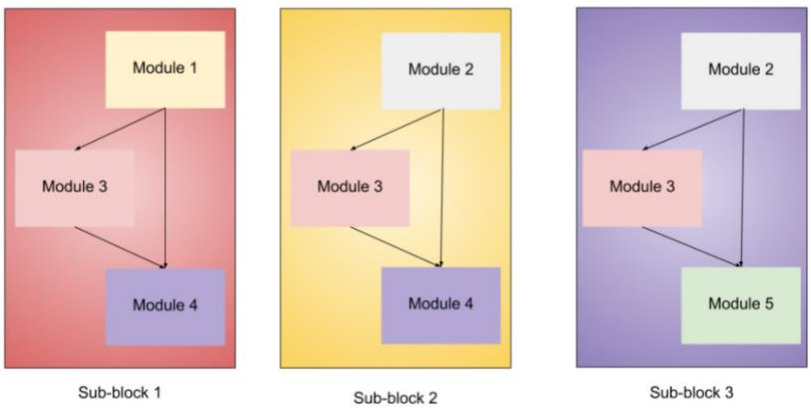


Figure 12: Sub – block 1, Sub – Block 2 and Sub – block 3 [3]

Above modules are all combined to create EfficientNet models. In **Figure 13** architecture of EfficientNet – B0 is shown. In the figure, x2 means the modules inside the brackets are repeated twice.

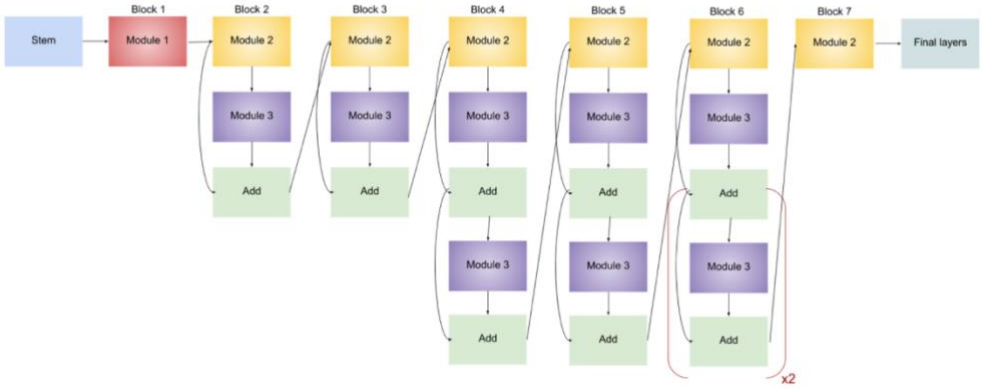


Figure 13: Architecture of EfficientNet – B0 [3]

Kernel size for convolution operations along with the resolution, channels and layers in EfficientNet – B0 is shown in **Table 1**. The idea of this table is a stage i with $\hat{L}_i$ layers, with input resolution $\langle \hat{H}_i, \hat{W}_i \rangle$ and output channels $\hat{C}_i$ for each row [2]. For all family members, resolution remains the same. The number of channels for each family model is shown in **Table 2**.

Stage i	Operator $\hat{\mathcal{F}}_i$	Resolution $\hat{H}_i \times \hat{W}_i$	#Channels $\hat{C}_i$	#Layers $\hat{L}_i$
1	Conv3x3	224×224	32	1
2	MBConv1, k3x3	112×112	16	1
3	MBConv6, k3x3	112×112	24	2
4	MBConv6, k5x5	56×56	40	2
5	MBConv6, k3x3	28×28	80	3
6	MBConv6, k5x5	14×14	112	3
7	MBConv6, k5x5	14×14	192	4
8	MBConv6, k3x3	7×7	320	1
9	Conv1x1 & Pooling & FC	7×7	1280	1

Table 1: EfficientNet-B0 baseline network

Stage	B1	B2	B3	B4	B5	B6	B7
1	32	32	40	48	48	56	64
2	16	16	24	24	24	32	32
3	24	24	32	32	40	40	48
4	40	48	48	56	64	72	80
5	80	88	96	112	128	144	160
6	112	120	136	160	176	200	224
7	192	208	232	272	304	344	384
8	320	352	384	448	512	576	640
9	1280	1408	1536	1792	2048	2304	2560

Table 2: Number of Channels for family models [3]

Details of all family models such as number of params and Top – 1 Accuracy are shown in **Table 3**.

Name	*# Params*	*Top-1 Acc.*	*Pretrained?*
`efficientnet-b0`	5.3M	76.3	✓
`efficientnet-b1`	7.8M	78.8	✓
`efficientnet-b2`	9.2M	79.8	✓
`efficientnet-b3`	12M	81.1	✓
`efficientnet-b4`	19M	82.6	✓
`efficientnet-b5`	30M	83.3	✓
`efficientnet-b6`	43M	84.0	✓
`efficientnet-b7`	66M	84.4	✓

Table 3: Efficientnet Models Parameters

3.1.2. Modifying EfficientNet Model to Fit with Data

As shown in Section 3.1.1.1, Pre – trained, EfficientNet – B0 is used for our model. Only the last layer of the model is modified. Output features dimension is changed from original, 1000 to 19 as number of labels are 19. The modified layer is shown below.

```
  (_conv_head): Conv2dStaticSamePadding(
    320, 1280, kernel_size=(1, 1), stride=(1, 1), bias=False
    (static_padding): Identity()
  )
  (_bn1): BatchNorm2d(1280, eps=0.001, momentum=0.010000000000000009, affine=True, track_running_stats=True)
  (_avg_pooling): AdaptiveAvgPool2d(output_size=1)
  (_dropout): Dropout(p=0.2, inplace=False)
  (_fc): Linear(in_features=1280, out_features=19, bias=True)
  (_swish): MemoryEfficientSwish()
)
```

Figure 14: Modified output layer

3.2. Training Processes

3.2.1. Data Pre – processing

Before training the data, **data pre–processing** is performed by doing following steps:

1. **Training Labels pre–processing**

   Training labels are transferred into one – hot encoder using MultiLabelBinarizer( ) from sklearn.preprocssing library. The original label was transferred from csv file into 19-

classes label file using one-hot encoding technique. (for example, a 19-dimension vector to represent the label of an image, if the image has label 1 and 8, then the value at the position 1 and position 8 will be 1, the rest 17 numbers will be all zero).

2. Image pre – processing:

Firstly, three channels (filters) are stacked. As shown in Section 2.3.2, Blend 1 Y, B, R or Blend 2 R, G, B is performed. After stacking, cropping and resizing is performed by using transforms from torchvision library. Image is resized to 256 (due to computation time). CenterCrop is 224, and mean and standard deviation values of Imagenet are used to normalize the tensor to pipeline to the model.

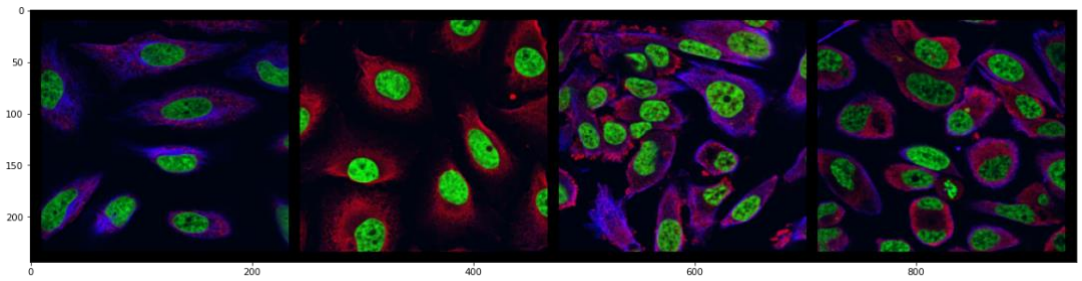


Figure 15: Blend 1 Y, B, R

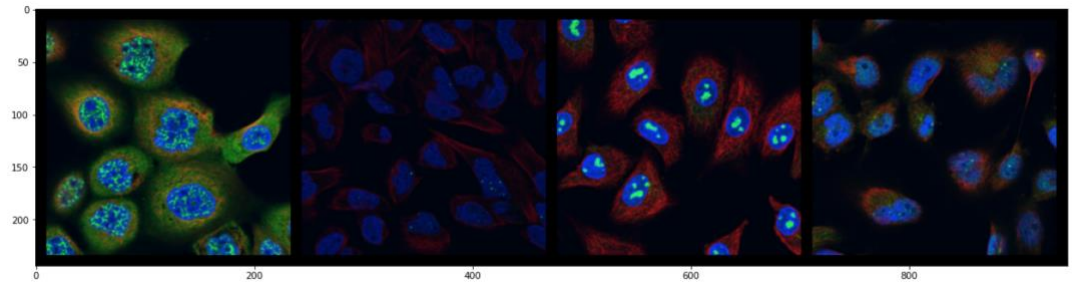


Figure 16: Blend 2 R, G, B

3. Splitting Data:

Splitting data is performed by using train_test_split from sklearn library. 25% of training data is used as validation data.

3.2.2. Training

In this section, training parameters, training procedures and training result analysis will be explained.

3.2.2.1. Training Parameters

1. Pre – trained EfficientNet – B0 is loaded. The last layer of the model is changed. Output features of the last layer of the model is changed from 1000 to 19 because labels is 19 in our case.
2. For loss, BCEWithLogitsLoss from Pytorch torch.nn is used. This loss function is the combination of Binary Cross Entropy (BCE) loss and Sigmoid layer in one single class. The reason for using BCEWithLogitsLoss is that it is more numerically stable than sigmoid function followed by traditional BCE loss. Due to the combination of these two, log – sum – exp trick can be applied for numerical stability.
3. For the optimizer, Adam optimizer from torch.optim library is used. The reason for using Adam optimizer is that its method is Stochastic Optimization, and it can update its own learning rate according to previous iterations' result. As it can update learning rate by itself, careful tuning of learning rates are not required. Moreover, as dataset is large in our case, Adam optimizer is the best choice.
4. One more sigmoid layer is added to the output to generate the confidence score for each class between 0 - 1.

3.2.2.2. Training Procedures

1. Firstly, YBR 3 – channel images (Blend1) were trained first for 10 epochs at one time and analysed the result. Training result of YBR 3 – channel is shown in **Figure 17**.
2. Secondly, RGB 3-channel images (Blend2) were trained for 2 times due to computation time allowed by Kaggle. Maximum allowed running time for one session at Kaggle is 9 hrs. Therefore, trained 2 times, first time 7 epochs and saved the model weights and the optimizer parameters. Then continued the second time training by loading the previous 7 epochs model weights and optimizer parameters for 8 epochs. Therefore, RGB 3-channel images were trained for total 15 epochs. The Training result of RGB 3 – channel is shown in **Figure 18**.

epochs	train_loss	train_f1	train_acc	train_pre	train_rec	val_loss	val_f1	val_acc	val_pre	val_rec
1	2.56E-01	1.84E-02	1.75E-02	9.73E-02	1.15E-02	2.68E-01	2.50E-03	3.30E-03	1.02E-01	1.28E-03
2	2.41E-01	2.66E-02	2.64E-02	1.09E-01	1.70E-02	2.58E-01	2.70E-02	2.00E-02	1.03E-01	1.84E-02
3	2.35E-01	4.13E-02	3.51E-02	1.96E-01	2.71E-02	2.66E-01	4.39E-02	5.74E-02	1.11E-01	4.05E-02
4	2.27E-01	7.58E-02	4.76E-02	5.05E-01	4.99E-02	2.63E-01	3.46E-02	4.33E-02	1.27E-01	2.86E-02
5	2.12E-01	1.34E-01	7.38E-02	5.52E-01	9.01E-02	2.86E-01	4.62E-02	5.12E-02	1.65E-01	4.25E-02
6	1.88E-01	2.68E-01	1.36E-01	6.35E-01	1.88E-01	3.21E-01	5.77E-02	4.44E-02	2.29E-01	4.33E-02
7	1.55E-01	4.45E-01	2.49E-01	7.05E-01	3.39E-01	3.41E-01	5.97E-02	5.37E-02	1.71E-01	5.33E-02
8	1.19E-01	6.08E-01	3.88E-01	7.72E-01	5.11E-01	3.98E-01	9.29E-02	6.95E-02	2.21E-01	8.58E-02
9	8.65E-02	7.20E-01	5.29E-01	8.19E-01	6.50E-01	4.34E-01	9.65E-02	5.96E-02	2.57E-01	7.83E-02
10	6.47E-02	7.92E-01	6.30E-01	9.02E-01	7.36E-01	5.03E-01	9.67E-02	6.22E-02	2.54E-01	8.18E-02

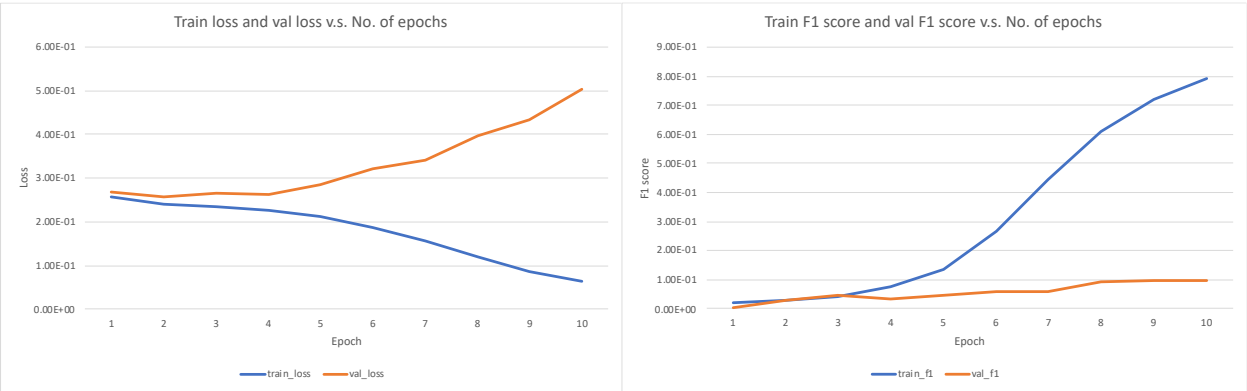


Figure 17: Training Results using YBR 3 – channel images for 10 epochs

epoch	train_loss	train_f1	train_acc	train_pre	train_rec	val_loss	val_f1	val_acc	val_pre	val_rec
1	1.91E-01	3.07E-01	2.12E-01	4.96E-01	2.36E-01	1.90E-01	3.28E-01	2.49E-01	6.79E-01	2.46E-01
2	1.38E-01	5.22E-01	3.81E-01	6.95E-01	4.35E-01	1.67E-01	4.31E-01	3.37E-01	7.22E-01	3.71E-01
3	1.18E-01	5.97E-01	4.46E-01	7.30E-01	5.16E-01	1.50E-01	4.89E-01	3.88E-01	7.42E-01	4.07E-01
4	1.01E-01	6.60E-01	5.09E-01	7.98E-01	5.87E-01	1.71E-01	4.87E-01	3.64E-01	6.71E-01	4.29E-01
5	8.39E-02	7.16E-01	5.76E-01	8.29E-01	6.50E-01	1.80E-01	5.10E-01	3.73E-01	6.74E-01	4.51E-01
6	6.98E-02	7.61E-01	6.34E-01	8.42E-01	7.05E-01	1.75E-01	5.23E-01	3.57E-01	7.02E-01	4.64E-01
7	5.47E-02	8.09E-01	7.00E-01	8.64E-01	7.65E-01	2.17E-01	4.91E-01	3.49E-01	7.28E-01	4.34E-01
8	8.04E-02	7.51E-01	6.24E-01	8.27E-01	6.94E-01	1.04E-01	6.55E-01	5.09E-01	8.82E-01	5.61E-01
9	5.32E-02	8.24E-01	7.16E-01	9.01E-01	7.80E-01	1.02E-01	7.09E-01	5.53E-01	8.07E-01	6.70E-01
10	4.17E-02	8.53E-01	7.68E-01	9.12E-01	8.15E-01	1.05E-01	6.95E-01	5.43E-01	7.64E-01	6.67E-01
11	3.30E-02	8.90E-01	8.08E-01	9.49E-01	8.56E-01	1.19E-01	6.97E-01	5.39E-01	7.77E-01	6.74E-01
12	3.01E-02	9.08E-01	8.23E-01	9.45E-01	8.78E-01	1.47E-01	6.89E-01	4.80E-01	7.68E-01	6.53E-01
13	2.70E-02	9.04E-01	8.40E-01	9.36E-01	8.80E-01	1.19E-01	7.08E-01	5.56E-01	8.15E-01	6.65E-01
14	2.37E-02	9.22E-01	8.56E-01	9.55E-01	8.99E-01	1.44E-01	6.90E-01	5.20E-01	8.42E-01	6.22E-01
15	2.12E-02	9.36E-01	8.72E-01	9.59E-01	9.17E-01	1.42E-01	6.96E-01	5.22E-01	8.34E-01	6.40E-01

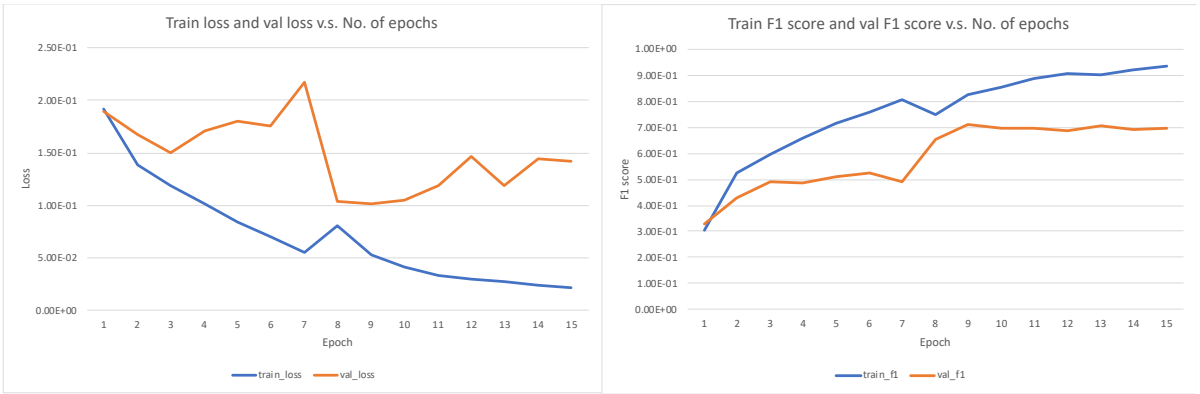


Figure 18: Training Results using RGB 3 – channel images for 15 epochs

3.2.2.3. Training Result Analysis and Comparison of YBR and RGB

It can be clearly seen in **Figure 17** that the results of the training of using YBR 3-channel images are not good. The training loss decreases, and validation loss increases when the number of epochs increases. It is clearly overfitted. And the validation F1 score does not have an obvious improvement compared to training F1 score.

According to **Figure 18**, the results of the training using RGB 3-channel images are much better. We can see that the best performance is at epoch = 9 when the validation loss is at the lowest value and the validation F1 score is at the highest value.

Therefore, the model weights saved at epoch = 9 will be loaded and applied this model onto the testing images data for the prediction output.

4. Inference Results and Discussion

Our multi-label classification model will be added with the single cell classification from *Darien Schettler* [4] to improve the performance score with weighted average of the confidence score. *Darien Schettler's* model basic concept is to identify slide-level images containing only one label. First, segment all the slide-level images (get RLEs for all cells for the slide-level images). Then Crop RGBY image around each cell. Pad each RGBY image tile to square and size to 256x256. Augment the image dataset (rotation, flipping (horizontal and vertical), minor-skew). Finally, train a model to classify those tile-level images. For the segmentation of all cells in an image, *Darien Schettler* used HPA-Cell-Segmentation tool [5]. After getting the confidence score results for the tile-level images, combine with our image-level confidence score by using weighted average of 0.5 and 0.5. 0.5 is for the tile-level images confidence score and 0.5 is for the image-level confidence score. We have tried and submitted using different weights and get the public scores respectively. Please refer to below table:

Notebook Version	Weights of tile-level/image-level	Public Score
v11	0.8/0.2	0.374
v10	0.7/0.3	0.384
v3	0.6/0.4	0.39
v7	0.5/0.5	0.394
v8	0.4/0/6	0.394
v9	0.3/0.7	0.392
v12	0.2/0.8	0.382

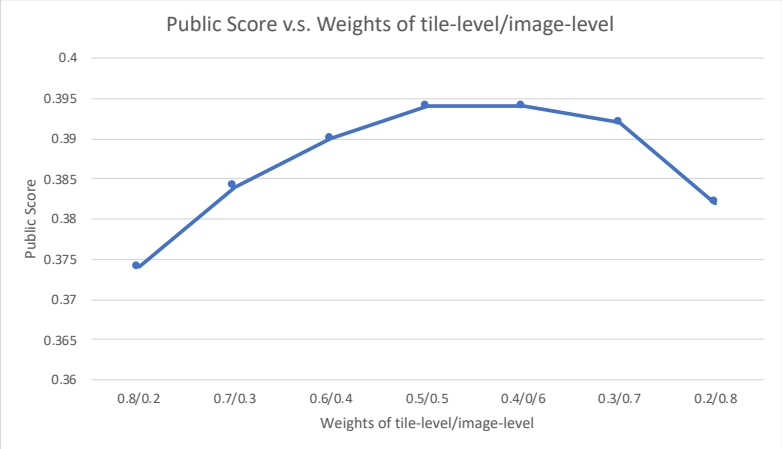


Figure 19: Notebook scores using different weights

We can see that the public scores result versus the weights of tile-level/image-level has the highest when the weights for tile-level classification confidence score and image-level classification confidence score are balanced, like 0.5/0.5 and 0.4/0.6. Which means that the

classification results of tile-level and image-level almost have the same level importance to contribute to the final public score.

This is Final submission result screenshot:

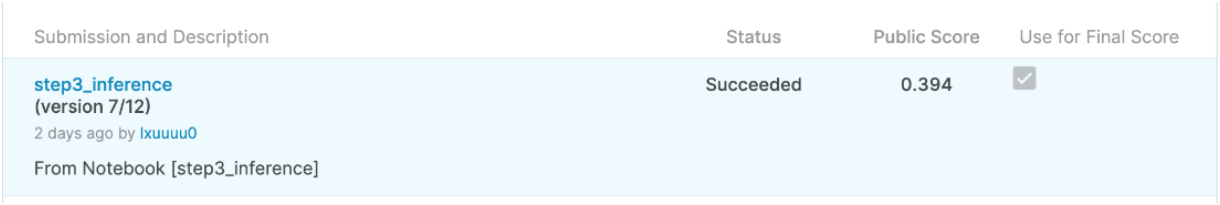


Figure 20: Final Kaggle Submission

5. Future Improvements

Due to the limited GPU usage time and GPU memory, we can only use the EfficientNet-B0 since it has the least number of parameters. And we keep the number of training epochs to be within 20 epochs. To improve the performance in the future, we may use EfficientNet-B7 and train more epochs. By using TPU and Multilabel Stratified K Fold Cross validation using not more 5 Folds in training, training time and accuracy can be improved. In Pytorch, EfficientNet is already implemented using ImageNet weights, if other weights are desired to use, like coco dataset weights, the original Keras EfficientNet is recommended because the user can manipulate the layer easier than Pytorch. Moreover, all the family members of EfficientNet can be used in training as a Transfer learning. It can improve the accuracy more than the remaining methods. Data Augmentation was tested for training set by doing Horizontal and Vertical flipping. The result shows it can reduce overfitting. However, it takes more than 15 Hrs to get the accuracy around 60%. Therefore, Data Augmentation is not recommended.

Reference

[1] Lukemelas, “lukemelas/EfficientNet-PyTorch,” [Online]. Available: https://github.com/lukemelas/EfficientNet-PyTorch.

[2] M. a. L. Q. Tan, “Efficientnet: Rethinking model scaling for convolutional neural networks,” pp. 6105--6114, 2019.

[3] V. Agarwal, “Complete Architectural Details of all EfficientNet Models,” [Online]. Available: https://towardsdatascience.com/complete-architectural-details-of-all-efficientnet-models-5fd5b736142.

[4] D. Schettler, “HPA - Cellwise Classification [INFERENCE],” [Online]. Available: https://kaggle.com/dschettler8845/hpa-cellwise-classification-inference.

[5] Oeway, “CellProfiling/HPA-Cell-Segmentation,” [Online]. Available: https://github.com/CellProfiling/HPA-Cell-Segmentation.

[6] F. JiaShi, “Lecture Slides”.

[7] J. T. Zhou, “Lecture Slides”.