Aperture, ISO and shutter speed detection and classification

The goal of this project is to design a scheme that can detach the aperture, ISO and shutter speed of a given picture. Since this three component is the most important elements to take a good picture. For this project we are focus on two kind of the picture: the portrait photo and landscape photo. For the portrait photo, I want to use the http://vis-www.cs.umass.edu/lfw/ as my training dataset. For landscape photo, I want to use https://www.kaggle.com/puneet6060/intel-image-classification as my training dataset for landscape. I can find the information from EXIF metadata fields to label the aperture, ISO and shutter speed of each training picture.

I want to design this project into a classification problem. For each training pictures, I need to find parameter to represent the aperture, parameter to represent the ISO and parameter to represent the shutter speed. Then for each element, I will plot the parameter and use the best clustering technique to classify each picture. I will try to design a neural network to predict these elements. For testing, I will then take portrait photos and landscape photos with different element setting (set aperture, shutter speed to constant and try all different ISO in same condition etc.) to test my trained model.

To quantify the ISO of the pictures, I want to find the Signal-to-noise ratio (SNR) to represent the ISO of the picture. To quantify the aperture of the picture, I want to find the blurriness of the edge of the picture to represent the aperture of the picture. The shutter speed will be represent by the ISO and aperture together.

Dev_2 feature extraction:

We need to capture the features from the image to detach the ISO and aperture. The key feature to detach the ISO is the brightness of the picture and the feature to detach the aperture is the blurriness of the edge of the pictures.

Now the problem is I want to have a method that can use a single float number to represent the total blurriness of the graph. The existing method to find the blurriness is computing the Fast Fourier Transform (FFT) of the image. After FFT, we examine the distribution of low and high frequencies to see if there is a low amount of high frequencies. This method can only show weather the image is blur or not. We can choose the number of the high frequencies to represent the blurriness of the image. However, it is hard to determine what is a low number of high frequencies.

Another approach [1] uses the variation of the Laplacian transform. It takes the grayscale of the image and convolve it with a cv2.CV_64F kernel. Then take the standard deviation squared (variance) of the response. The variance can show the blurriness of the image. The Laplacian is often used for edge detection. If an image contains high variance then there is a wide spread of responses, which can represent a non-blur image. If the image contains low variance, indicating there are very little edges in the image. Therefore the image is blurred (less edges). Using the variance value of the Laplacian transform can represent the blurriness of the image and it is more accurate than the FFT method.

I using the second method to show the blurriness of the landscape image set from [https://www.kaggle.com/puneet6060/intel-image-classification.](https://www.kaggle.com/puneet6060/intel-image-classification) I convert the image to grayscale and apply Laplacian transform on it to find the variance of the Laplacian as the blurriness of the image. I printed the blurriness of the image and show the command line screen shot below:

```
Blurrness is
4155.666993777778
Blurrness is
13640.650990293334
Blurrness is
11978.12875413136
Blurrness is
6447.585838569878
Blurrness is
16117.646709876542
Blurrness is
12216.141150599506
Blurrness is
390.6704330646914
Blurrness is
2605.6987460266664
Blurrness is
31257.206552462223
Blurrness is
13079.7248022025
Blurrness is
5196.926706747655
Blurrness is
6931.530642826667
```

Figure 1- Screen shot to show the blurriness of the images

The next steps of the aperture finding is to connect the blurriness of the image to the actual aperture (extract from EXIF information). Then build a CNN to train to detach the aperture.

I used the python Pillow library to calculate the brightness of the image. I used the ImageStat module to calculates the global statistics for an image. We can calculate the brightness using the data from statistics of an image. I can use the average pixel brightness to represent the total brightness of the image. I can also use the RMS pixel brightness to represent the total brightness of the image. For now I used the RMS brightness to represent the total brightness. The screen shot below shows the brightness of the image of given dataset.

```
Blurrness is
1597.1161283950617
brightness is
114.13118258822223
Blurrness is
11228.683831672099
brightness is
66.51471111111111
Blurrness is
3843.3738236266668
brightness is
91.27893013824075
Blurrness is
14053.65580669432
brightness is
67.14325763765063
Blurrness is
8850.828665937777
brightness is
147.5764843733658
Blurrness is
5676.5099
```

Figure 2- Screen shot to show the brightness of the images

To further detach the ISO I need to have three data of an image. The brightness, the exact ISO (from EXIF information) and the peak signal to noise ratio (PSNR use to detach the noise of an image under high ISO). I will try to build the CNN based on brightness and SNR of the picture to predict the ISO of a image.

Dev_3 feature extraction:

Introduction:

It is very hard to classify the image based on the feature extraction from previous deliverables. There are two reasons I gave up on these methods. 1. The image set I use is lack of labels. 2. The blurriness and brightness of the image is vary too much to do a classification. Therefore I decided to train a neural network to do the job for me. I will have one neural work train to classify aperture and one train to classify iso. Due to the non-standardize labels of shutter speed of the data set, I decided not consider classify the shutter speed in this project.

New Dataset:

The previous dataset is lack of labels of the EXIF information of the picture. Therefore I cannot tell the exact photographic information of the image. I use the MIRFLICKR-250000 dataset instead. This collection contains photographs from the photo-sharing website Flickr that have been highly annotated and retain a significant quantity of EXIF data. I pick 20000 image from the MIRFLICKR-250000 and classified them based on aperture and iso. I classified 18279 images belonging to 23 different aperture (aperture from f1.2 to f22) as my dataset for aperture recognition. 10576 of them (iso from 100 to 3200) used for training and 7703 of them used for validation. Similarly, 13066 of the image are used for iso classification. 6995 of them are for training and 6071 are for validation.

Image model:

The image model's job is to extract features connected to a picture, such as blurriness, brightness, high resolution and signal to noise ratio. These features are learned by standard CNN architectures from image recognition and classification. I mainly use two image model to train and validate the data.

The first model I use is a simple CNN model with 3 convolution layer and two fully connected layer. It is a relatively deep CNN model compare to what we build during the class and it can successfully classify the iso and aperture of the picture. However, the accuracy of this model is very low. I only get around 10% of accuracy when classify the aperture and 30% accuracy when classify the iso. The screen shot below shows the accuracy result during training and I also plot the learning curve of this model:

```
Epoch 1/10
106/106 [==============================] - 19s 172ms/step - loss: 6.0072 - accuracy: 0.1190 - val_loss: 3.2449 - val_accuracy: 0.1045
Epoch 2/10
106/106 [==============================] - 19s 175ms/step - loss: 5.7585 - accuracy: 0.1677 - val_loss: 3.1977 - val_accuracy: 0.0878
Epoch 3/10
106/106 [==============================] - 20s 185ms/step - loss: 5.7636 - accuracy: 0.1774 - val_loss: 3.1825 - val_accuracy: 0.0595
Epoch 4/10
106/106 [==============================] - 19s 181ms/step - loss: 5.7097 - accuracy: 0.1930 - val_loss: 3.1811 - val_accuracy: 0.0521
Epoch 5/10
106/106 [==============================] - 18s 165ms/step - loss: 5.7229 - accuracy: 0.1890 - val_loss: 3.1847 - val_accuracy: 0.0280
Epoch 6/10
106/106 [==============================] - 17s 163ms/step - loss: 5.8137 - accuracy: 0.1996 - val_loss: 2.9883 - val_accuracy: 0.1004
Epoch 7/10
106/106 [==============================] - 17s 163ms/step - loss: 5.7236 - accuracy: 0.1944 - val_loss: 3.0468 - val_accuracy: 0.0662
Epoch 8/10
106/106 [==============================] - 18s 170ms/step - loss: 5.6269 - accuracy: 0.2065 - val_loss: 3.2434 - val_accuracy: 0.0741
Epoch 9/10
106/106 [==============================] - 18s 173ms/step - loss: 5.6793 - accuracy: 0.1932 - val_loss: 3.1355 - val_accuracy: 0.0326
Epoch 10/10
106/106 [==============================] - 18s 168ms/step - loss: 5.7088 - accuracy: 0.2137 - val_loss: 3.0293 - val_accuracy: 0.0609
Traceback (most recent call last):
```

Figure 1. The screen shot of aperture training and validation.

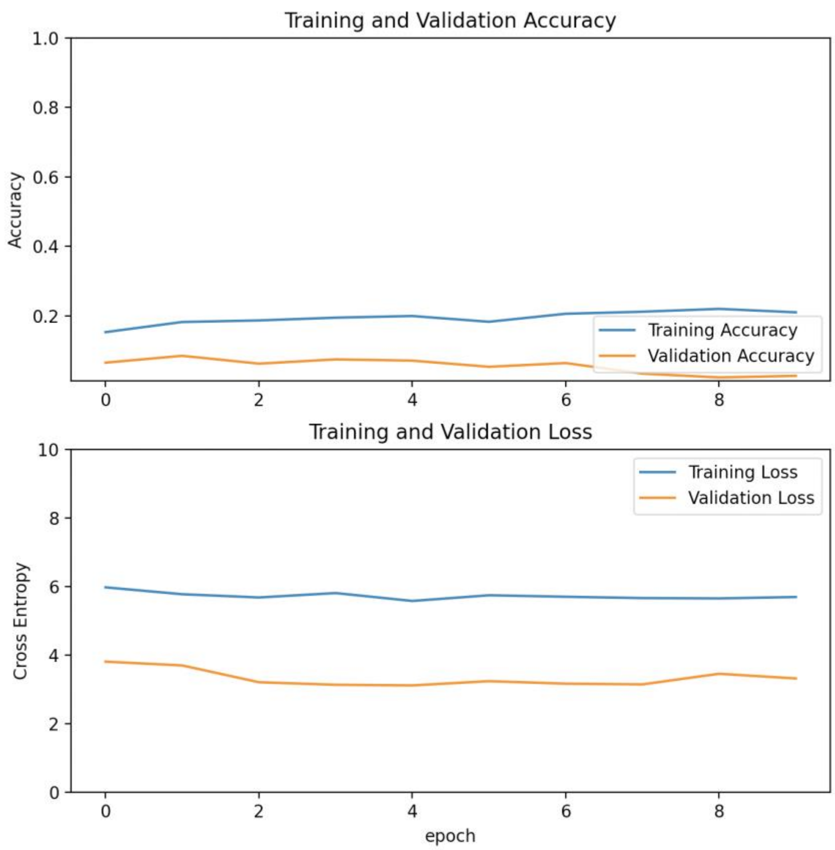
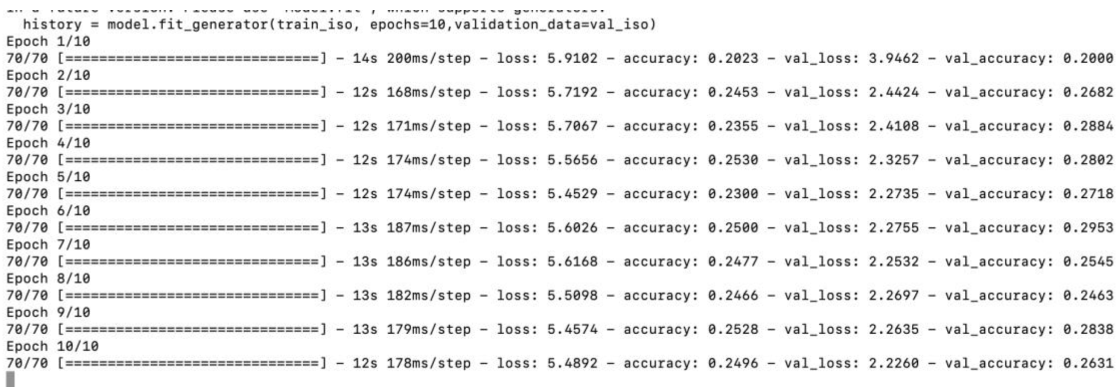


Figure 2- Learning curves of the training and validation accuracy/loss of aperture recognition



```
  history = model.fit_generator(train_iso, epochs=10,validation_data=val_iso)
Epoch 1/10
70/70 [==============================] - 14s 200ms/step - loss: 5.9102 - accuracy: 0.2023 - val_loss: 3.9462 - val_accuracy: 0.2000
Epoch 2/10
70/70 [==============================] - 12s 168ms/step - loss: 5.7192 - accuracy: 0.2453 - val_loss: 2.4424 - val_accuracy: 0.2682
Epoch 3/10
70/70 [==============================] - 12s 171ms/step - loss: 5.7067 - accuracy: 0.2355 - val_loss: 2.4108 - val_accuracy: 0.2884
Epoch 4/10
70/70 [==============================] - 12s 174ms/step - loss: 5.5656 - accuracy: 0.2530 - val_loss: 2.3257 - val_accuracy: 0.2802
Epoch 5/10
70/70 [==============================] - 12s 174ms/step - loss: 5.4529 - accuracy: 0.2300 - val_loss: 2.2735 - val_accuracy: 0.2718
Epoch 6/10
70/70 [==============================] - 13s 187ms/step - loss: 5.6026 - accuracy: 0.2500 - val_loss: 2.2755 - val_accuracy: 0.2953
Epoch 7/10
70/70 [==============================] - 13s 186ms/step - loss: 5.6168 - accuracy: 0.2477 - val_loss: 2.2532 - val_accuracy: 0.2545
Epoch 8/10
70/70 [==============================] - 13s 182ms/step - loss: 5.5098 - accuracy: 0.2466 - val_loss: 2.2697 - val_accuracy: 0.2463
Epoch 9/10
70/70 [==============================] - 13s 179ms/step - loss: 5.4574 - accuracy: 0.2528 - val_loss: 2.2635 - val_accuracy: 0.2838
Epoch 10/10
70/70 [==============================] - 12s 178ms/step - loss: 5.4892 - accuracy: 0.2496 - val_loss: 2.2260 - val_accuracy: 0.2631
```

Figure 3. The screen shot of iso training and validation

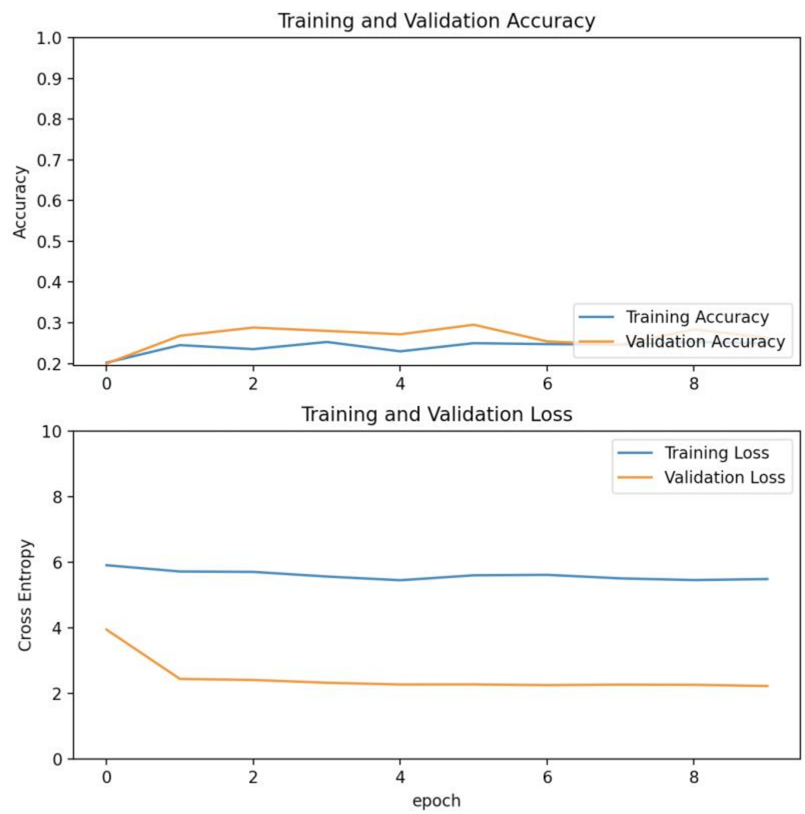


Figure 4- Learning curves of the training and validation accuracy/loss of iso recognition

As we can see on the learning curve, the accuracy is low and does not go high when adding more epoch of training. The loss is very high and do not decrease during the training. I think there are two major reasons: First, the model is not complex enough to learn the underlying patterns of the image. Second, the training dataset is too small to accurately generalize across classes.

In order to have higher accuracy of image model, I need to have more dataset and more complex network. So I decided to modify the pre-trained model as my major image model. I choose the MobileNet V2 model from google. This model is pre-trained on the ImageNet dataset, which consisting over 1 million images to 1000 classes. I think the MobileNet V2 is complex enough to handle my task and it is not that complicate (compare with VGG16), so my CPU can handle it. I use the very last layer before the flatten operation for feature extraction. This layer is called the "bottleneck layer". The bottleneck layer features retain more generality as compared to the final/top layer. I freeze this layer and add a Global average layer to convert the feature to a single 1280-element vector per image and apply a Dense layer to convert the features into a single prediction per image. The accuracy of this Image model is high as 95% when classify the aperture and 95% when classify the iso. The screen shot below shows the accuracy result during training and I also plot the learning curve of this model:

```
Epoch 1/10
106/106 [==============================] - 31s 273ms/step - loss: 0.5582 - accuracy: 0.8130 - val_loss: 0.2356 - val_accuracy: 0.9562
Epoch 2/10
106/106 [==============================] - 32s 307ms/step - loss: 0.2320 - accuracy: 0.9549 - val_loss: 0.2075 - val_accuracy: 0.9565
Epoch 3/10
106/106 [==============================] - 30s 280ms/step - loss: 0.2164 - accuracy: 0.9563 - val_loss: 0.2052 - val_accuracy: 0.9565
Epoch 4/10
106/106 [==============================] - 30s 280ms/step - loss: 0.2126 - accuracy: 0.9564 - val_loss: 0.2037 - val_accuracy: 0.9565
Epoch 5/10
106/106 [==============================] - 30s 287ms/step - loss: 0.2109 - accuracy: 0.9563 - val_loss: 0.2019 - val_accuracy: 0.9565
Epoch 6/10
106/106 [==============================] - 32s 302ms/step - loss: 0.2087 - accuracy: 0.9564 - val_loss: 0.2000 - val_accuracy: 0.9565
Epoch 7/10
106/106 [==============================] - 31s 289ms/step - loss: 0.2074 - accuracy: 0.9564 - val_loss: 0.1983 - val_accuracy: 0.9565
Epoch 8/10
106/106 [==============================] - 35s 331ms/step - loss: 0.2057 - accuracy: 0.9564 - val_loss: 0.1966 - val_accuracy: 0.9565
Epoch 9/10
106/106 [==============================] - 35s 334ms/step - loss: 0.2042 - accuracy: 0.9565 - val_loss: 0.1953 - val_accuracy: 0.9565
Epoch 10/10
106/106 [==============================] - 33s 311ms/step - loss: 0.2031 - accuracy: 0.9564 - val_loss: 0.1940 - val_accuracy: 0.9565
```

Figure 5. The screen shot of aperture training and validation of MobileNet V2

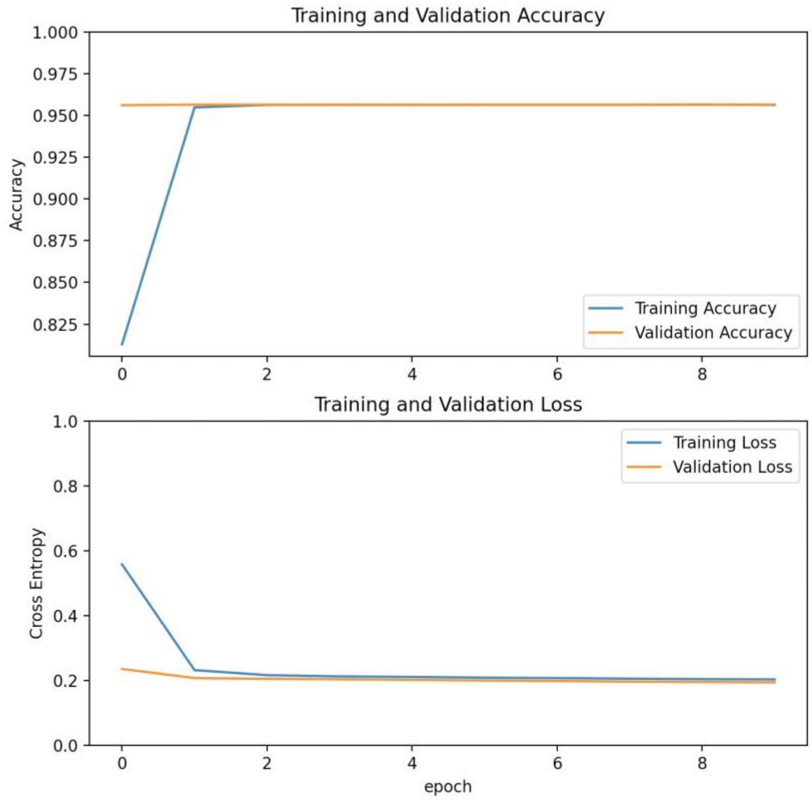


Figure 6- Learning curves of the training and validation accuracy/loss of aperture recognition of MobileNet V2

```
  history = model.fit_generator(train_iso, epochs=10,validation_data=val_iso)
Epoch 1/10
70/70 [==============================] - 21s 270ms/step - loss: 0.3436 - accuracy: 0.9151 - val_loss: 0.2296 - val_accuracy: 0.9521
Epoch 2/10
70/70 [==============================] - 17s 248ms/step - loss: 0.2372 - accuracy: 0.9506 - val_loss: 0.2219 - val_accuracy: 0.9524
Epoch 3/10
70/70 [==============================] - 18s 260ms/step - loss: 0.2308 - accuracy: 0.9520 - val_loss: 0.2198 - val_accuracy: 0.9524
Epoch 4/10
70/70 [==============================] - 18s 263ms/step - loss: 0.2282 - accuracy: 0.9519 - val_loss: 0.2177 - val_accuracy: 0.9524
Epoch 5/10
70/70 [==============================] - 19s 270ms/step - loss: 0.2264 - accuracy: 0.9523 - val_loss: 0.2153 - val_accuracy: 0.9524
Epoch 6/10
70/70 [==============================] - 19s 269ms/step - loss: 0.2248 - accuracy: 0.9524 - val_loss: 0.2131 - val_accuracy: 0.9524
Epoch 7/10
70/70 [==============================] - 19s 273ms/step - loss: 0.2222 - accuracy: 0.9520 - val_loss: 0.2112 - val_accuracy: 0.9524
Epoch 8/10
70/70 [==============================] - 21s 301ms/step - loss: 0.2199 - accuracy: 0.9524 - val_loss: 0.2092 - val_accuracy: 0.9524
Epoch 9/10
70/70 [==============================] - 21s 307ms/step - loss: 0.2182 - accuracy: 0.9523 - val_loss: 0.2077 - val_accuracy: 0.9524
Epoch 10/10
70/70 [==============================] - 21s 307ms/step - loss: 0.2169 - accuracy: 0.9523 - val_loss: 0.2063 - val_accuracy: 0.9524
```

Figure 5. The screen shot of iso training and validation of MobileNet V2

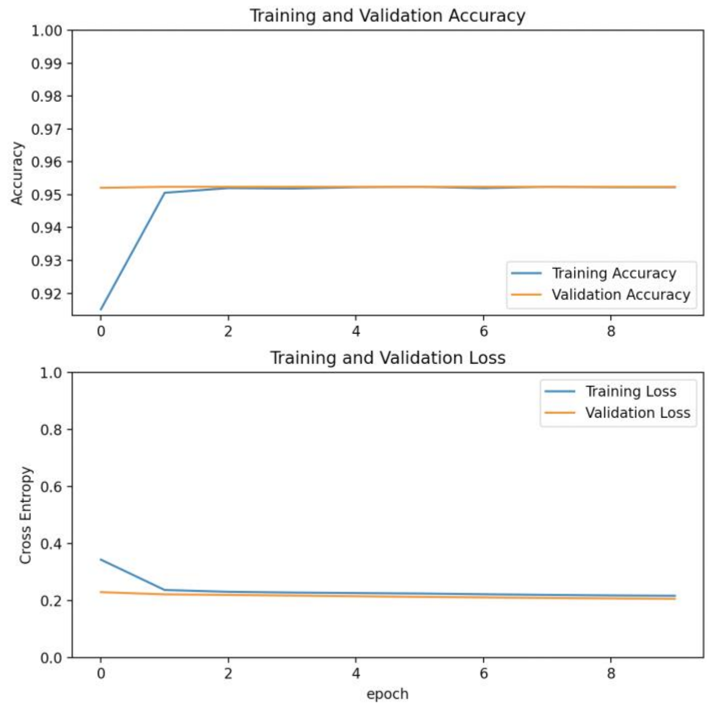


Figure 8- Learning curves of the training and validation accuracy/loss of iso recognition of MobileNet V2

As we can see, due to the large dataset of pre-trained data and more deeper network, the accuracy of this Image model is much better than previous.

Next steps:
I will use my camera to shoot some picture of different iso and aperture and use the MobileNet V2 and my simple model to see whether the classification is real accuracy or not.

Reference:

[1] J. L. Pech-Pacheco, G. Cristobal, J. Chamorro-Martinez and J. Fernandez-Valdivia, "Diatom autofocusing in brightfield microscopy: a

comparative study," Proceedings 15th International Conference on Pattern Recognition. ICPR-2000,