1 Baseline Experiments

1.1 Dataset

The image dataset used in our work, which is obtained from the website of Kaggle diabetic retinopathy competition provided by EyePACS, contains 35126 high resolution fun- dus photographs taken under various imaging conditions. These fundus photographs have been labeled by a trained clinician with a scale of 0 to 4 based on the severity of DR. As a result, it is a typical five-class classification task where the model needs to correctly predict the severity of DR.

Besides, it is noteworthy that these images are taken by different types of cameras under diverse conditions. Some of the images are inverted while some are incomplete due to the variant microscope imaging procedure. Also, noise in both images and labels is unavoidable in the raw data set, which puts forward a high demand to the robustness of our classification system.

Last but not least, as shown in table 1, the dataset is highly imbalanced. Consequently, a "lazy" model which gives 0 as prediction all the time can achieve an accuracy of around 72%. To show the capability of the model in classifying minor classes, we include balanced accuracy in our performance metrics.

Label	Number of samples	Percentage
0	25810	73.5%
1	2443	6.9%
2	5292	15.1%
3	873	2.5%
4	708	2.0%

Table 1: *Distribution of original dataset*

1.2 Image Preprocessing and Data Augmentation

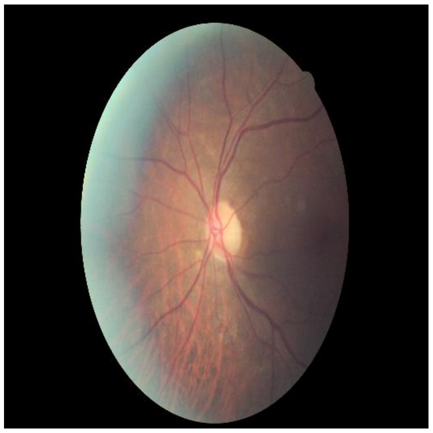
Image Augmentation. As mentioned above, the raw fundus images in the dataset have large variations in terms of brightness, contrast and the colour of eyeballs. Such irregularities constitute noise to the model and thus prevent it from effectively learning meaningful features. In order to standardise these images and reduce artifacts produced by irrelevant factors, we refer to the preprocessing scheme proposed by competition winner Graham. More precisely, we subtract each pixel value of images by the weighted means of its surrounding pixel values, and add it by 50% grayscale. This operation is similar to the 'high pass' processing in the PhotoShop software, which makes the blood vessels as well as the lesion areas in fundus images more explicit. Then, the fundus area in images will be clipped to 95% of the original size by covering a mask with a transparent circle on the center in order to remove the boundary effects arising from the last operation. Afterwards, pixel values of images are normalised to [-1 , 1]. Effects of the preprocessing procedure are demonstrated in figure 1.

Image Resize. The original images in the dataset are of different sizes around 3k$\times$2k. Loading a dataset with images of such resolution exceeds the capacity of the gpu memory and the the inconsistency of the resolution makes it impossible to train a model using the raw images. Consequently, we resize them to 256$\times$256. Initially, the resizing is done "on the fly", which means that the images are modified upon being read for training. This brought much repeated and unnecessary work and slowed down the training. For the sake of improvement, we resize the images and save them as a new dataset and directly train the model on it. In this way the training is sped up by around 73 times.

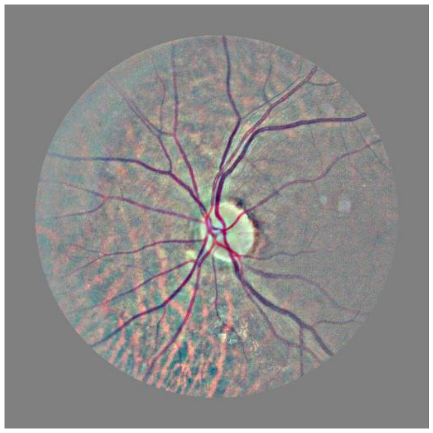
Data Augmentation. In order to enhance diversity of the dataset and therefore alleviate the problem of overfitting, we further apply various transformations to the images fed to the model during training, which includes:

- Randomly flip the images of left eye and right eye horizontally. Since human eyes are structurally mirror-symmetric, the fundus images can be flipped horizontally without affecting the label they correspond to.
- Randomly perform geometric transformation on images, including randomly scaling to 90%-110% of their size, translating by $\pm$0.05% of the total width, rotating from -30 to 10 degrees and shearing from -10 to 10 degrees.
- Randomly change the brightness to 85%-115%, decreasing or improving the contrast in the range of 85%-115%.

All the steps of augmentation listed above are performed with a probability of 50%.



(a) *original image*



(b) *processed image*

Figure 1: *Comparison between a typical original image and a typical preprocessed image. it can be found that fundus region is standardized to be a circular area, containing the optic disk, macula and main vessels. Furthermore, the background color of retina is faded, while the venules, hard exudates and hemorrhages are emphasized.*

1.3 Model Architecture

Model Architecture. For this typical image classification task, we employ two popular network structure families: resnet[**he_deep_2015**], efficientnet[**tan_efficientnet_2019**] and InceptionV3[**szegedy_rethinking_2015**]. For the resnet family, resnet18 and resnet50 are used while for the efficientnet family we use efficientnet b0 - b3. Table2 shows the number of parameters of different networks. We also make use of transfer learning which consists in replacing the classification layer with a 5-unit one and in loading pretrained weights of the mentioned networks on Imagenet before the start of training.

Model	Number of Params
resnet18	11179077(∼11M)
resnet50	23518277(∼23M)
efficientnet b0	4013953(∼4M)
efficientnet b1	6519589(∼6M)
efficientnet b2	7708039(∼7M)
efficientnet b3	10703917(∼10M)
InceptionV3	24357354(∼24M)

Table 2: *Number of parameters of different models*

1.4 Evaluation Methods and Results

To assess the performance of models tested, the original dataset is split into training set and validation set with a ratio of 7:3 and the resulting datasets are used throughout the whole series of experiments. Since the split is random, the resulting datasets approximately preserve the distribution of the original dataset(table3).

To compare the performance of the models obtained with Kaggle competitors, Quadratic weighted kappa score is adopted as one of the evaluation metrics. It is able to describe the agreement between predicted labels and true labels more effectively, since it cost-sensitively penalizes the wrong predictions in terms of the degree of classification error. It ranges from 0 to 1 and is the higher the better.

Training epochs are all set to be 20. Training is performed on 3 Tesla K40c gpus and the batch size is adjusted such that it is the greatest that does not exceed the memory capacity of the gpus. As for implementations, the balanced accuracy and cohen kappa score are calculated using corresponding sklearn functionalities and the deep learning library employed is Pytorch.

We test the model on the validation set after each epoch. The results shown in table4 are the ones at the best epoch(early stopping). Several observations can be made:

- The models have better performance on images with higher resolution(512×512 in our case).
- The efficientnet family has a better performance than the resnet family with much less parameters.
- efficientnets with image size of 512×512 achieves a cohen kappa score comparable to the top teams in the kaggle competition leaderboard (for reference, the score of 10^{th} team is 0.814.)
- The implementation of efficientnet in pytorch is not memory efficient thus the allowed batch size is much smaller than it should be, which results in a longer training time.

Considering training time and performance, we select efficient b0 as the baseline model and use image size of 512×512 for the following experiments.

Label	Number of samples	Percentage
0	18134	73.8%
1	1710	6.9%
2	3659	14.8%
3	603	2.5%
4	480	2.0%

(a)

Label	Number of samples	Percentage
0	7673	72.9%
1	733	6.9%
2	1632	15.5%
3	270	2.5%
4	228	2.2%

(b)

Table 3: *Distribution of training dataset(a) and validation dataset(b)*

	Image Size	Batch Size	Accuracy	Balanced Accuracy	Cohen Kappa Score	Training Time(20 epochs)
resnet18	256×256	128	0.812	0.476	0.701	2h40min
resnet18	512×512	64	0.836	0.521	0.760	8h26min
resnet50	256×256	64	0.819	0.519	0.725	5h59min
resnet50	512×512	64	0.826	0.467	0.725	19h25min
efficientnetb0	256×256	192	0.815	0.440	0.696	2h29min
efficientnetb0	512×512	16	**0.852**	0.586	0.805	14h59min
efficientnetb1	256×256	192	0.817	0.455	0.706	2h29min
efficientnetb1	512×512	16	0.850	0.587	0.805	19h15min
efficientnetb2	256×256	96	0.823	0.549	0.739	2h20min
efficientnetb2	512×512	16	**0.852**	**0.610**	**0.811**	19h49min
efficientnetb3	256×256	96	0.828	0.539	0.746	5h50min
efficientnetb3	512×512	24	0.851	0.585	0.803	10h1min
InceptionV3	299×299	64	0.816	0.497	0.706	7h12min

Table 4: *Performance of different models*

2 Dataset Balancing

The model obtained in section1 is trained on an imbalanced dataset. Therefore, the distribution of the prediction is also imbalanced and significantly leans to the major class (i.e. class with label 0). This effect can be seen in figure2(a) which is the confusion matrix of efficient b0 tested on the validation set. In practice, this implies that patients with with DR would be wrongly diagnosed as Non-DR, which would cause a delay of treatment and aggravate the illness. From the point of view of robustness, the decision boundary of the dominant class is considerably large and accordingly the one of the minor class is tiny and brittle: a small perturbation to the images with minor label leads to a change in prediction, which makes the model non robust.

In order to alleviate this model learning bias, we extensively experiment with two families of dataset-balancing methods: Modify loss to give more emphasis on the minority classes and rebalance the class prior distributions in a pre-processing procedure. In detail,

- Loss Modification
  - Focal Loss[**lin_focal_2018**]
  - Class balanced loss[**cui_class-balanced_2019**]
  - Inverse Frequency Loss (The loss of images is reweighed by the inverse of the frequency of the class they represent.)
- Dataset Balancing
  - Oversampling(Add repeated minor class samples until all classes have the same number of samples. The resulting dataset in our case counts 103240(25810*5) samples in total.)

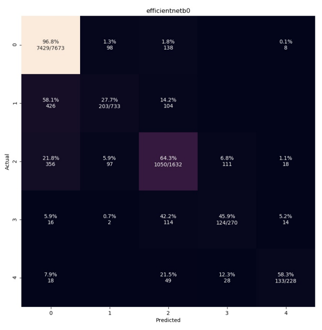
2.1 Results

The results of various methods are listed in table5. The setting is the same as section1 and the model used is efficientnet b0. Some observations are made:

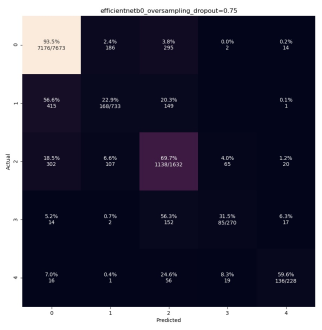
- In spite of data augmentation, training on an oversampled dataset exhibits model overfitting probably owing to repeatedly visiting duplicated samples. Overfitting causes a sharp drop in model performance. Although the problem of overfitting can be mitigated through dropout, judging from the confusion matrix (figure2(b)), the resulting model has a distribution of decision similar to the one trained on imbalanced dataset but with a lower accuracy in predicting the major class. In addition, due to drastic increase of number of samples (from 20k to 100k), the training time climbs considerably.
- If one starts the training from a pre-trained weight obtained from the above baseline experiments and applies oversampling, the model still overfits but at the first few epochs its decision distribution becomes more balanced(figure2(c)) at the expense of prediction accuracy of the major class and the overall accuracy stays at an acceptable level. The shown confusion matrix is that of the model obtained after 1 epoch of training.
- On the contrary, training with a reweighed loss function helps to balance decision distribution without overfitting problem and the training is notably faster than oversampling. Overall, focal loss with γ=2 and class balanced loss with β=0.99 would be good choices among all tested loss functions since they yield the highest balanced accuracy with merely a minor loss in normal accuracy. According to the confusion matrix they result in (figure2(d, e)), they successfully force the model to attach more importance to minor classes, which results in a better accuracy in predicting certain minor classes. However, they have little influence on balancing the decision distribution and it can still be found that the model tends to give 0 as prediction.

	Batch Size	Accuracy	Balanced Accuracy	Cohen Kappa Score	Training Time (20 epochs)
crossentropy(baseline)	16	0.852	0.586	0.805	14h59min
focal loss@γ=1	48	0.847	0.587	0.800	6h02min
focal loss@γ=2	48	0.848	**0.629**	0.814	6h08min
focal loss@γ=3	48	0.843	0.602	**0.815**	6h12min
focal loss@γ=4	48	0.842	0.593	0.803	5h50min
class balanced loss@β=0.9	48	0.848	0.597	0.809	6h05min
class balanced loss@β=0.99	48	**0.852**	0.621	0.802	6h09min
class balanced loss@β=0.999	48	0.848	0.605	0.807	6h20min
inverse frequency loss	16	0.798	0.596	0.764	14h30min
oversampling+dropout=0	16	0.704	0.399	0.612	46h39min
oversampling+dropout=0.75	48	0.826	0.555	0.775	21h37min
oversampling+dropout=0.9	48	0.780	0.578	0.780	20h05min
oversampling+restart	48	0.760	0.630	0.765	5h4min (5 epochs)

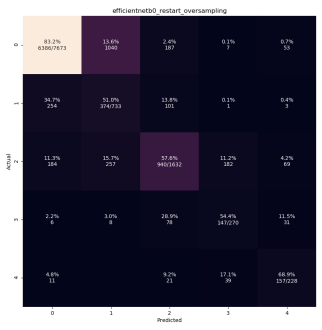
Table 5: *Performance of efficient b0 under different balancing methods*



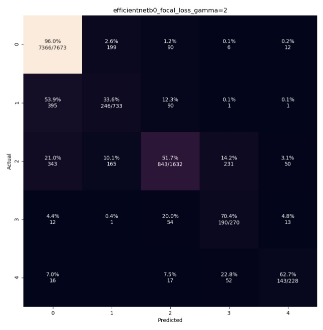
(a)



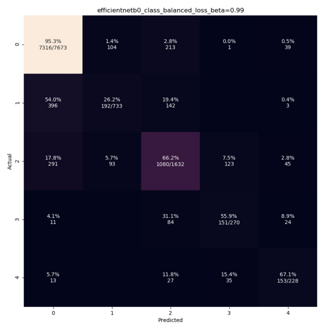
(b)



(c)



(d)



(e)

Figure 2: *Comparison of confusion matrices of efficient b0 trained on imbalanced dataset(a), on oversampled balanced dataset with dropout rate=0.9(b), on oversampled balanced dataset starting from pre-trained weights(c), on imbalanced dataset using focal loss with γ=2(d) and on imbalanced dataset with β=0.99(e). The matrices are obtained by testing on the 30% validation dataset.*

3 Training with Corrupted Images

In view of equipping the model with robustness against low-quality or even corrupted images, we train the model on a dataset of images that undergo different types of corruptions(light leakage, motion blur, under expose and over expose) at different levels (from 1 to 4, the higher the level, the worse the images). The effects of corruptions are demonstrated in figure3. The corruption processing is Zeiss-proprietary.

The training is proceeded using the same training-validation split as the above sections. The baseline model is efficientnet b0, the images are of size 512×512, the batch size is set to be 48 and dropout is not used.

Although the underlying corruptions can be served as a new data augmentation and be applied newly to each image at each epoch, due to the lengthy image processing which takes around 50 seconds per image, we process the images of the (oversampled) original dataset and save them as new datasets. No other data augmentation is added. The corrupted datasets are then directly loaded for training without any further operation. During the generation of the corrupted datasets, four types of corruptions are randomly applied with random parameters that conform to their randomly chosen level in the level range.

Test on original validation set. The results on the original imbalanced dataset and oversampled balanced dataset obtained after 20 epochs and validated on the untouched imbalanced validation set are respectively summarised in table6 and table7. Compared with model trained on imbalanced dataset, we notice a drop in performance of the model because of the loss of information due to corruption. Nonetheless, when acting on an oversampled balanced dataset, the corruptions play a role of an enhanced data augmentation that mitigates the problem of overfitting.

Test on corrupted validation set. To check if training on a corrupted dataset can bestow resistance to corruption on the model, we further test the models on corrupted imbalanced validation dataset whose corruption level range is 1 – 4. The results are listed in table8.

As revealed in the results, training on a corrupted dataset allows the model to gain robustness against the corruptions of the same types.

Resistance against adversarial attack. To investigate the ability of the corrupted-trained model to withstand an adversarial attack, we measure such ability by recording the success rate of an attack at different “box size" ϵ of the l_∞ PGD attack[**madry_towards_2017**]. The success rate of an attack is defined by the number of the change of predictions after the attack over the total number of samples. Briefly speaking, ϵ represents the size of the box in the sample space inside which the adversarial samples are searched. As a result, the greater ϵ is, the larger the box will be, the stronger the attack will be and the higher the success rate will be. At the same ϵ, if the success rate of an attack is higher, the model is considered to be weaker against the attack.

The experiment is performed on 240 samples randomly chosen from the original validation set, the number of steps of the PGD attack is set to be 5 .

Results are presented in table9. It turns out that training on corrupted dataset does not seem to help the model gain resistance against gradient-based adversarial attacks. On the contrary, the corrupted-trained model becomes more vulnerable in face of such attacks. This behaviour might be due to the fact that corrupted images still contain some non-robust features while lose some robust features of the original images. Consequently, compared with baseline, the model learns some non-robust features of the corrupted images and at the same time have no access to certain robust features of the original dataset, which results in a worse performance on adversarial samples generated from the original dataset. It would be better if the model is trained on a mixed dataset where both original and corrupted images are present.

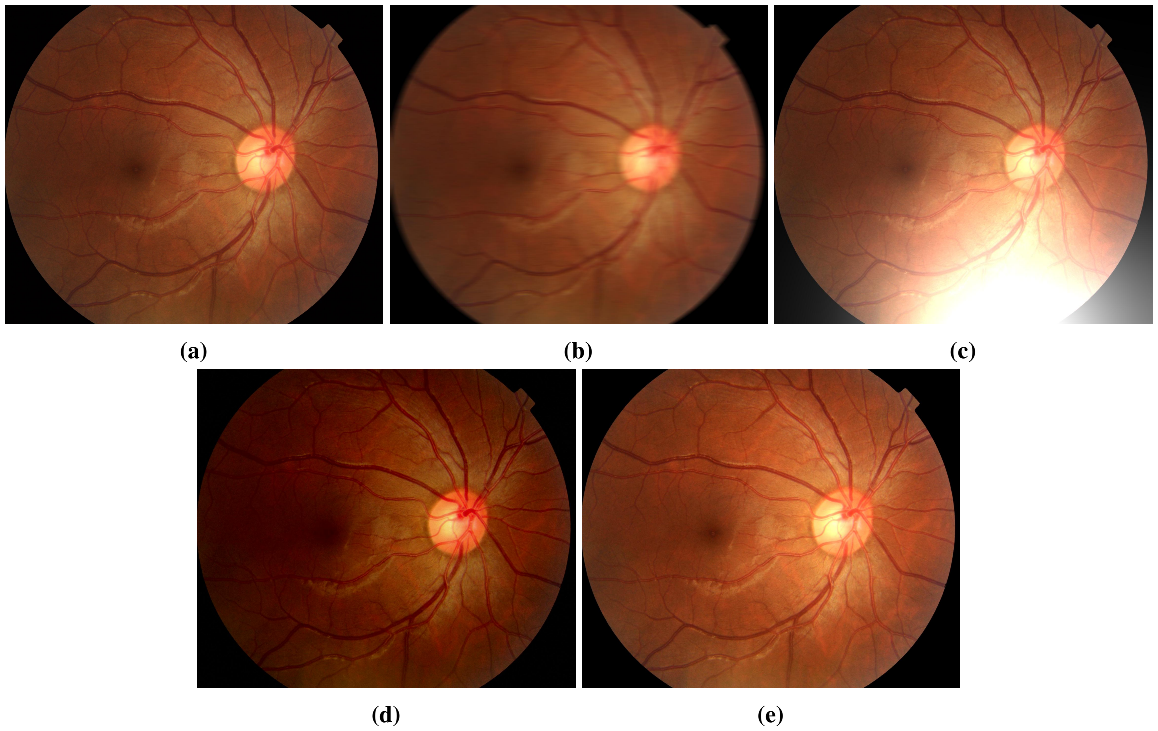


Figure 3: *Different kinds of corruptions on a test image(a): motion blur(b), light leakage(c), under expose(d) and over expose(e).*

Corruption Level Range	Accuracy	Balanced Accuracy	Cohen Kappa Score
baseline	0.852	0.586.	0.805
1	0.812	0.475	0.701
1 – 2	0.817	0.497	0.725
1 – 3	0.826	0.500	0.718
1 – 4	0.817	0.515	0.731

Table 6: *Performance of efficient b0 trained on imbalanced dataset at different level ranges*

Training Corruption Level Range	Accuracy	Balanced Accuracy	Cohen Kappa Score
baseline	0.704	0.399	0.612
1	0.806	0.477	0.705
1 – 2	0.812	0.471	0.703
1 – 3	0.807	0.481	0.707
1 – 4	0.803	0.466	0.695

Table 7: *Performance of efficient b0 trained on balanced dataset at different level ranges*

Training Corruption Level Range	Accuracy	Balanced Accuracy	Cohen Kappa Score	Accuracy Drop
baseline	0.764	0.391	0.501	0.088
1	0.778	0.417	0.575	0.034
1 – 2	0.785	0.446	0.619	0.032
1 – 3	0.776	0.393	0.586	0.050
1 – 4	0.791	0.403	0.598	0.026

Table 8: *Performance of corrupted-trained efficient b0s on corrupted(1 – 4) imbalanced validation dataset*

Training Corruption Level Range	ϵ=0.2	ϵ=0.4	ϵ=0.6
baseline	0.196	0.238	0.341
1	0.467	0.867	0.896
1 – 2	0.588	0.875	0.883
1 – 3	0.554	0.867	0.871
1 – 4	0.579	0.871	0.879

Table 9: *Success rate of PGD attack at different ϵ for different models*