Blindness Severity Classification using EfficientNet

Anusha Porwal
AI for Health Technologies
Aalto University

***Abstract*—This project attempts to classify blindness severity by looking at images of eyes. EfficientNet Image Encoders are used as base models, and they are modified slightly to train them to be able to make the classification. Experiments with the dataset were also conducted to see if they can improve the model performance in any way. The best performing model was a EfficientNet_b4 model with the random weight sampler, attaining a balanced accuracy score of 0.578 on the test set.**

I. Introduction

Many people suffer from diabetic retinopathy and is cited as the leading cause of blindness. Early detection is definitely helpful in slowing down the progression, which is why it's accurate detection is so important. To take a small step towards solving this, the APTOS-19 Blindness Detection Challenge [1] was launched in June 2019 in an attempt to predict the stage of blindness from images of eye scans. In this project various methods are attempted to get a model that performs well.

This report is structured as follows: Section II will talk about the dataset and the changes done to it. In Methodology, the base model (EfficientNet) is discussed and the 4 models trained are described. The training conditions and evaluation metrics are also discussed. This is followed by the result and conclusion sections. The code repository for this project can be found at this link.

II. Dataset

The Dataset was created by Aravind Eye Hospital in India. They collected images by traveling to rural areas of India and then getting them labeled with the help of doctors. There are 5 classes indicating the severity of blindness: 0 (no blindness detected) to 4 (severe blindness detected). Examples from each class are shown in the fig. 1. The blindness severity can be seen by looking at the increasing presence of spots towards the center of the image. They are also circled in red.

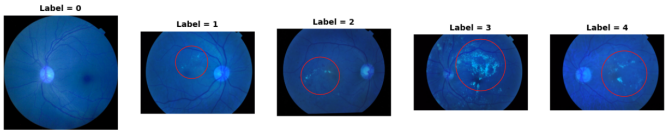

Fig. 1. Original training images

It was a highly unbalanced dataset - the class distribution is shown in the pie-chart in fig. 2. Since the competition is already over, the authors of the dataset have released the correct test labels as well, hence it was not needed to create the splits for the dataset. The images were of size approximately 3000x3000 pixels.

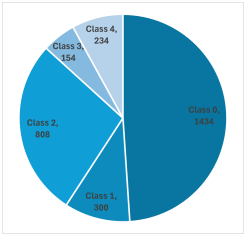

Fig. 2. Class Distribution of Dataset

A. Data Augmentation

To deal with the unbalanced dataset, data augmentation was done. It was attempted to get about 1600 images for each class. The transformations done to the images were: Random Gamma, Random Sharpness, Random Rotation, Introducing Noise, and Transposing the image. All these transformations are introduced with a probability of 0.5 The images from the training dataset are augmented and saved in a new folder. [2], [3]

B. Data Subset containing equal number of examples in each class

As an alternative to Data Augmentation, where the size of the dataset was increased, the Subset method was tried out where random images are chosen from each class to equal to the number of samples in the smallest class. This led to a small train dataset with about 150 images per class, so a total of 750 images in the whole dataset.

C. Data Preprocessing

While loading the dataset into DataLoaders, few more transformations were also introduced. They were Random Contrast, Random Brightness, Sharpness and Resizing the image. The idea was that by increasing and adjusting the brightness, contrast and sharpness, the spots that are used to detect the blindness severity will be seen more easily. The image in fig. 3 shows examples of this. The images were originally 3000x3000 in dimension - which is very large and will take a long time to process through the network. To make computation faster, the size was reduced to 300x300 or 380x380 depending on the model being trained. The ideal size for EfficientNet_b4 is 380x380, as that is the size the pretraining was done with. [4]

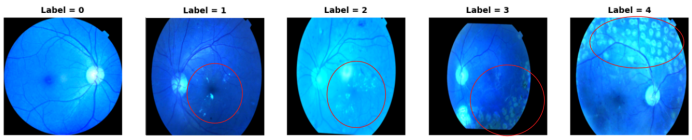

Fig. 3. Preprocessed Train Images

III. Methodology

This project was done using the pretrained EfficientNet models. It was picked because of a comparison study done on different image encoder models where the model sized were compared with their Top-1 accuracies. The EfficientNet family of models mostly gave the best performance. [5]

A. EfficientNet [6]

EfficientNet is a family of Convolutional Neural Network based image encoder models. It was proposed in 2019 by Google AI researchers and is now commonly used for various image-related tasks like object recognition and image segmentation. The most important feature of this family of models is it's ability to scale up in size while still maintaining accuracy and efficiency. EfficientNet_b0 is the baseline model proposed and EfficientNet_b7 is the current largest of this family. B0 has 5.3 million parameters and B7 has 66.7 million parameters. B4 is used in this project for the improved models, and it has 19.5 million parameters. [7], [8]

B. Training Experiments

Four main experiments were conducted here, using the EfficientNet pretrained models as well as the different training datasets that were created. They are described below in detail.

Experiment 1: Baseline using training data subset and EfficientNet_b4
Due to the high class imbalance in the dataset, the easiest and quickest way to get a balanced dataset was to reduce the size of the dataset so that each class has the same number of samples, which is equal to the size of the smallest class. After this the EfficientNet_b4 model was picked, and its weights were frozen and its final classification layer was modified by adding two linear layers and a ReLU activation in the middle.

Experiment 2: EfficientNet_b0
This model was trained using the dataset in its original form, without any augmentation. The Data Preprocessing was done before the model was loaded into the PyTorch DataLoaders. The smallest EfficientNet model was used, the b0 baseline version. The weights were frozen, and the final classification layer was replaced by a simple linear layer with the output features equaling 5, to predict one of the 5 classes.

Experiment 3: Using Weighted Random Sampler and EfficientNet_b4
The balanced accuracy obtained in the previous models was still quite low, even if the accuracy was good. A possible reason for this could be that the dataset is so highly unbalanced that it is affecting the learning and the results. The Weighted Radom Sampler [3], [9]–[12] gives a weight to each element of the dataset, based on which class that element belongs to and what percentage of the dataset belongs to that class. So, if a dataset belongs to class 0 (which is the largest class in this dataset - about 50%), it has a lower chance of getting picked into the DataLoader. This helps in making the class distribution more even than before. Apart from this, the Efficient_b4 pretrained model is used, and the same changes that were made to the classification layer in the baseline model (expt. 1) have also been done here.

Experiment 4: Using Augmented Dataset and EfficientNet_b4
Data Augmentation was done for the classes that had low number of examples - finally each class had about 1600 samples. The entire dataset now contains 8000 samples and is completely balanced. The same model as used in Experiment 1 was used, including the modification of the final classification layer.

Model Training
All models were trained for a standard 20 epochs with a learning rate of 0.001 and a batch size of 8. The model from Experiment 1 was trained for a few additional epochs because the loss and accuracy only started to get better near the 20th epoch, so it seemed like a good idea to keep training further. Adam Optimizer was used along with Cross Entropy Loss for multi-class classification. Balanced Accuracy Scores were also calculated along with the accuracy for the training and validation sets to see how well the models are performing.

Local computer's CPU was used for training all the models because the GPU quota on google Colab kept running out before the model training completed and the wait for the free quota to refresh was too much. Each epoch took about 30 minutes to 1 hour for training. The working Jupyter notebook to run the complete code on Google Colab is provided separately in the GitHub Repo (ProjectCode.ipynb). To see the results, please refer to the other separate Jupyter notebooks in the repository.

C. Evaluation Metrics

This section describes the different evaluation metrics used to see how well the model is performing. They are also used to compare the models with each other.

Accuracy
Accuracy is defined as the percentage of correct predictions for a given set of data. The labels must match exactly to be counted as a correct prediction. sklearn.metrics.accuracy_score was used to calculate it. [13]

Balanced Accuracy
Balanced accuracy is important in cases where the class

distribution of the data is imbalanced. In such cases, the accuracy score could be misleading because it could just be favoring the majority class, and we do not find out how the model is performing on the predicting the smaller classes. Balanced Accuracy looks at the true positive rates and the true negative rates giving a better overview of the results [14]. sklearn.metrics.balanced_accuracy_score was used to calculate it [15].

Cohen-Kappa Score

Cohen-Kappa is used to measure the inter-annotator agreement rate [16]. In this case, the agreement rate of the different model predictions and the true test labels are calculated. This will show us how much the predictions agree with the true labels. sklearn.metrics.cohen_kappa_score was used to calculate it [16].

Hypothesis Testing Setup

To compare if an improvement was actually made upon the baseline, a statistical test was performed and the setup is as follows. The test accuracy, test balanced accuracy and the Cohen-kappa scores were used as the metrics to calculate the p-value.

H0*: There is no statistical significance between the baseline and the best model of the remaining 3 experiments.*
H1*: There is a statistically significant difference between the two models.*

IV. Results

The results of all the experiments are presented in Table I. The best performing model was the one trained using the Random Weight Sampler, by comparing the Test Metrics (Accuracy, Balanced Accuracy and Cohen-Kappa Scores). The Hypothesis Testing suggests that the Null Hypothesis can be rejected with a confidence level of 41%.

Data Augmentation was expected to work the best because of the addition of more samples, which would ideally have helped to teach the model better. But looks like something went wrong during the training, and it has ended up becoming the worst model - performing poorer than the baseline model.

The training and validation accuracies and losses are plotted for the baseline and the random weight sampler model. The plots for the other two models are present on the GitHub in the /outputs folder.

V. Conclusion

In this project, it was attempted to train a model which is able to detect blindness severity by looking at images of eyes. The EfficientNet family of models was used as the image encoder. Different techniques to process the data were also tried out. A statistical test was also performed to see if an improvement was made over the baseline model. The best model was the one that uses EfficientNet_b4 along with the Random Weight Sampler. It gave a Balanced Accuracy Score of 0.578 and a Cohen-Kappa Score of 0.687 when compared with the true labels on the test set.

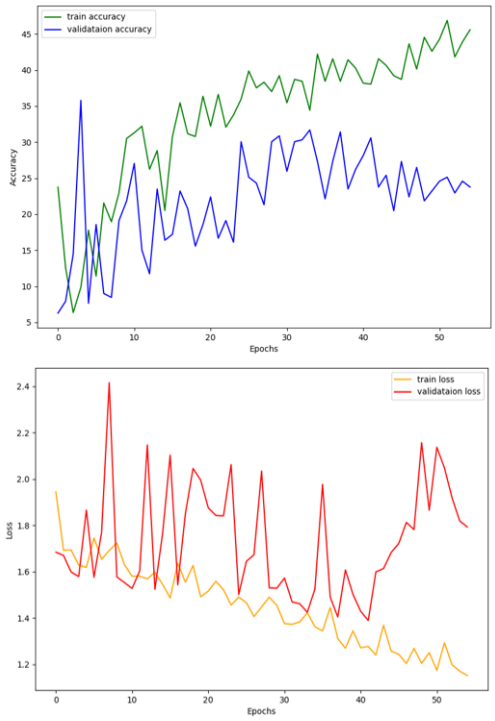

Fig. 4. Accuracy and Loss curves for Baseline Model

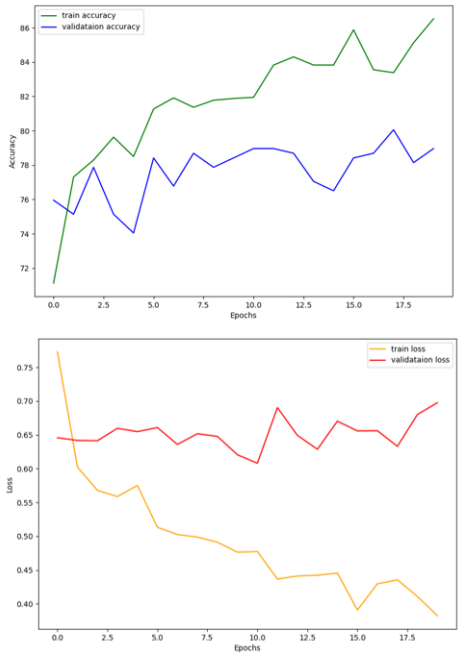

Fig. 5. Accuracy and Loss curves for RandomWeightSampler Model

References

[1] A. P. T.-O. Society, "APTOS 2019 Blindness Detection," 2019. [Online]. Available: https://www.kaggle.com/c/aptos2019-blindness-detection/overview

TABLE I
RESULTS TABLE

(Expt)Model	% TrainParams	BaseModel	Dataset	trainACC	valACC	trainBACC	valBACC	testACC	testBACC	testCK
(1)Baseline	2.55	b4	Subset	38.442	31.421	0.384	0.302	30.328	0.272	0.107
(2)b0_Linear	0.16	b0	Original	73.584	76.776	NA*	0.554	78.415	0.533	0.645
(3)WeightSampler	2.55	b4	Original	83.379	**80.055**	0.681	**0.578**	**80.601**	**0.578**	**0.687**
(4)DataAug	2.55	b4	Augmented	26.250	10.383	0.260	0.212	9.290	0.201	0.002

*Value wasn't calculated during the implementation.

[2] E. Randellini, "Data Augmentation," 2022. [Online]. Available: https://github.com/enrico310786/brain_tumor_classification/blob/-master/augment_train_dataset.py

[3] Multiple, "How to handle class imbalance." [Online]. Available: https://discuss.pytorch.org/t/how-to-handle-imbalanced-classes/11264/11

[4] C. Lepelaars, "EfficientNetB5 with Keras (APTOS 2019)," 2019. [Online]. Available: https://www.kaggle.com/code/carlolepelaars/efficientnetb5-with-keras-aptos-2019#Preprocessing-

[5] "Model Size vs Accuracy Comparison." [Online]. Available: https://www.researchgate.net/figure/Model-Size-vs-Accuracy-Comparison-EfficientNet-B0-is-the-baseline-network-developed-by_fig5_352346653

[6] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," *ICML*, 2019.

[7] P. Potrimba, "What is EfficientNet? The Ultimate Guide." 2023. [Online]. Available: https://blog.roboflow.com/what-is-efficientnet/

[8] A. Sarkar, "Understanding EfficientNet — The most powerful CNN architecture," 2021. [Online]. Available: https://arjun-sarkar786.medium.com/understanding-efficientnet-the-most-powerful-cnn-architecture-eaeb40386fad

[9] Multiple, "How to Prevent Overfitting." [Online]. Available: https://discuss.pytorch.org/t/how-to-prevent-overfitting/1902?u=smth

[10] pytorch, "Data in pytorch." [Online]. Available: https://pytorch.org/docs/stable/data.html

[11] Multiple, "Intuition behind weighted random sampler in pytorch." [Online]. Available: https://stackoverflow.com/questions/69318733/intution-behind-weighted-random-sampler-in-pytorch

[12] ——, "Proper way of using weighted random sampler." [Online]. Available: https://discuss.pytorch.org/t/proper-way-of-using-weightedrandomsampler/73147

[13] scikit learn, "accuracy_score." [Online]. Available: https://scikit-learn.org/1.5/modules/generated/sklearn.metrics.accuracy_score.html

[14] C. M. C. da Silva, "A Data Scientists guide to balanced accuracy." [Online]. Available: https://www.blog.trainindata.com/a-data-scientists-guide-to-balanced-accuracy/

[15] scikit learn, "balanced_accuracy_score." [Online]. Available: https://scikit-learn.org/1.5/modules/generated/sklearn.metrics.balanced_accuracy_score.html

[16] ——, "Cohen-Kappa Score." [Online]. Available: https://scikit-learn.org/1.5/modules/generated/sklearn.metrics.cohen_kappa_score.html