Early Detection of Diabetic Retinopathy Using Deep Learning

Jibanul Haque, Natnael Daba, & Moazam Soomro

Abstract

Diabetic Retinopathy (DR) is an eye disease found in people with chronic diabetes. It is one of the leading of blindness among working-aged people around the world. Fast detection of this disease plays an important role in stopping or slowing down this disease. The current techniques of detection DR require a significant amount of time since the current process is manual. Additionally, expert clinicians are not always available or are limited in many parts of the world which makes the process even slower. Researchers have been working to make this process automated that will speed up the detection process significantly and also will make the decision process faster. In this project, we try several popular CNN-based models to handle this issue. Our models archived a maximum of 80.49% accuracy and 0.8951 quadratic weighted kappa in classifying DR.

***Keywords*—** Computer-aided diagnosis, Retinopathy detection, Deep learning, Retinal fundus images

1 Introduction:

Diabetic retinopathy (DR) is an eye disease that is caused by damage to the blood vessels in the tissue at the back of the eye (retina). It is the leading cause of blindness in the working-age population of the developed world. It is estimated to affect over 93 million people. Figure 1 below shows a clear picture of how diabetic retinopathy is visually different than a normal retina. In the DR retina, abnormal signs can be found like hemorrhages, abnormal growth of blood vessels, aneurysms, cotton wool spots, and hard exudates.

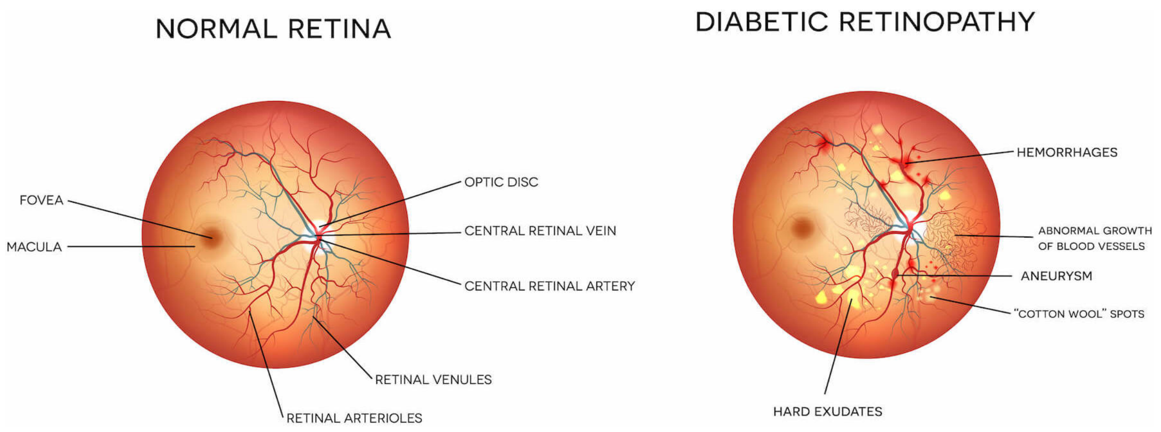


Figure 1: Image shows a Normal retina vs diabetic retinopathy [1]

Diabetic Retinopathy is a disease that progresses in four stages as listed below:

1. Mild DR: this is the first stage where microaneurysms are observed in the retina.
2. Moderate DR: this is the second stage in which the blood vessels become distorted or swelled up which halt blood moving.
3. Severe DR: this is the third stage of the disease in which the blood vessels are blocked which results in blood deprived retina. At this stage, the retina is signaled for the growth of new blood vessels.

4. Proliferate DR: this is the fourth stage of the disease. At this stage, the new blood vessels have grown inside the face of the retina in vitreous gel and blocking the eye.

1.1 Objective

There is no doubt that diabetic retinopathy is a serious disease that is affecting millions of people from all over the world. The progression to vision impairment can be slowed or averted if it is detected in time. However, there are quite a few roadblocks in this case as the disease often shows few symptoms until it is too late to provide effective treatment. Detecting it requires an expert physician to examine the digital color fundus photographs of the retina which take a significant amount of time and labor. This process takes a day or two, on average, and because of this delayed process, the treatment gets delayed. The key here is to detect early when there is enough time for treatment for precaution. In this project, we intend to build a deep learning model that can robustly detect diabetic retinopathy in retina images that can help physicians to start the treatment as soon as possible.

2 Related Work

Diabetic retinopathy (DR) identification from fundus images has been a well-studied problem in computer vision literature. This problem has been formulated as a binary classification [3] where the goal is to identify the presence or absence of DR. Some researchers have extended this problem to multi-class learning, where the severity of the disease ranges from 'No DR' to 'Proliferate' [6]. Initial research shows the application of classical methods [5] to solve detection of Diabetic retinopathy, where recent research has progressed towards application of deep learning and transfer learning [4]. Priya and Aruna [5], use raw color features from fundus images with SVM to perform binary classification of diabetic retinopathy. Conde et al. [2] use binary classification by application PCA for image dimensionality reduction and then followed by several machine learning methods: decision trees, naïve Bayes, neural network, k-NN, and SVM. Pratt and Coenen [4], developed CNN to identify fine details such as micro-aneurysms and hemorrhages. Asiri et al. [1] conducted a comprehensive survey to study various methods and datasets to address the problem of diabetic retinopathy including retinal blood vessel segmentation and detection of various lesions. Hagos and Kant applied transfer learning by using pre-trained InceptionNet-V3 for multi-class classification on a smaller sub-set of the Kaggle dataset. Sarki et al., trained several deep learning networks such as DenseNet, ResNet50, VGG, and Xception to achieve an accuracy of 86% on the No DR/Mild DR classification task.

3 Data Exploration

Retinal images were provided by EyePACS, a free platform for retinopathy screening. [2]

Images are high-resolution ranging from 5M-10M pixels. Dataset is divided into five classes:

- No DR
- Mild
- Moderate
- Severe
- Proliferate DR

The dataset consists of five classes in training data which are highly imbalanced as evident in Figure 2.

[2]`https://www.kaggle.com/c/diabetic-retinopathy-detection/data`

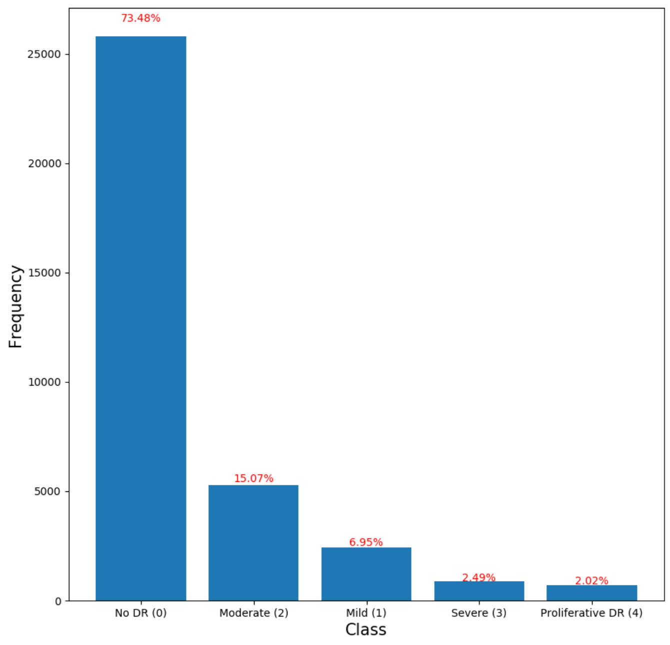


Figure 2: Distribution of training classes

3.1 Data Pre-processing

Various data augmentation and transformation techniques are applied to improve performance. Below is a list of models together with their corresponding data preprocessing techniques applied to the dataset on which they were trained.

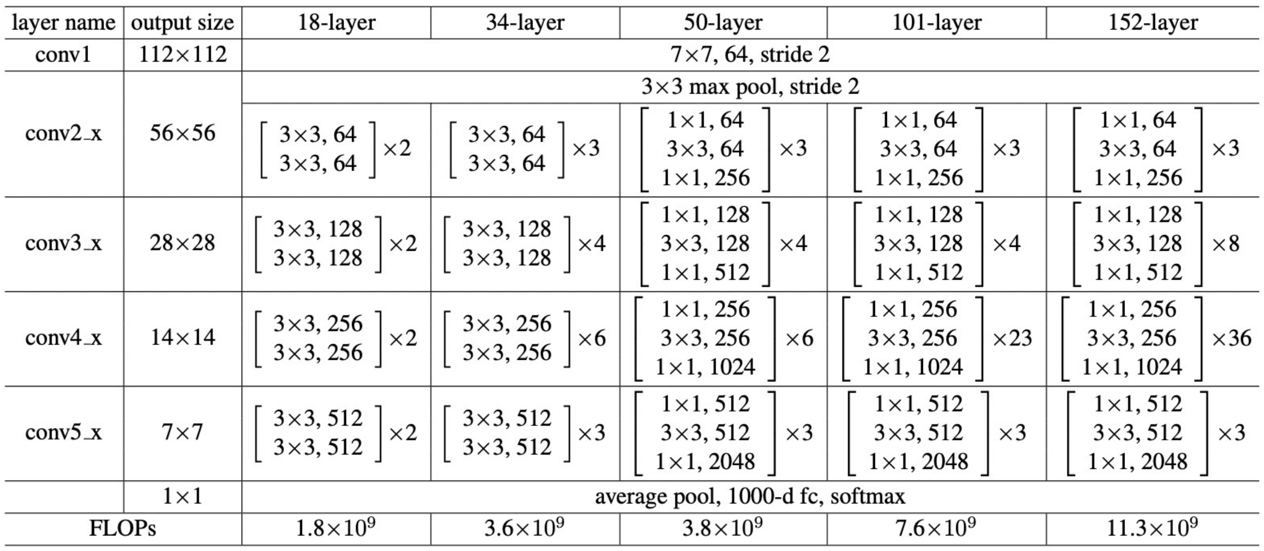
1. ResNet-18 Model
   (a) Random resizing crop
      - scale=(1 / 1.15, 1.15),
      - ratio=(0.7561, 1.3225))
      - size=(224, 224)

   (b) Normalization
   (c) Random Vertical Flip
   (d) Random Horizontal Flip
   (e) Random Affine transformation
2. EfficientNet-B4 Model
   (a) Same as ResNet-18 but with new input image resolution of (380, 380)
3. EfficientNet-B5 Model
   (a) Same as ResNet-18 but with new input image resolution of (456, 456)
4. Custom CNN Model
   (a) Input image resolution of (224,224)

4 Methodology

4.1 Models

ResNet-18 is a convolutional neural network that is 18 layers deep. Although a pre-trained version of the network is available which was trained on ImageNet, we trained ours from scratch. We resized our input images down to 224-by-224 to match the specifications of the model provided by the original authors.



layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times6$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times6$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times23$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure 3: Architecture of baseline model used [3]

EfficientNet is a CNN-based architecture. It's also a scaling method that uniformly scales all dimensions of depth/width/resolution using a compound coefficient as shown in Figure (4) below. (a) is a baseline network example; (b)-(d) are conventional scaling that only increases one dimension of network width, depth, or resolution. (e) is a compound scaling method that uniformly scales all three dimensions with a fixed ratio. We used two variants of the EfficientNet family of architectures. These are EfficientNet-B4 and EfficientNet-B5 each requiring the input resolutions of images to be 380-by-380 and 456-by-456 respectively.

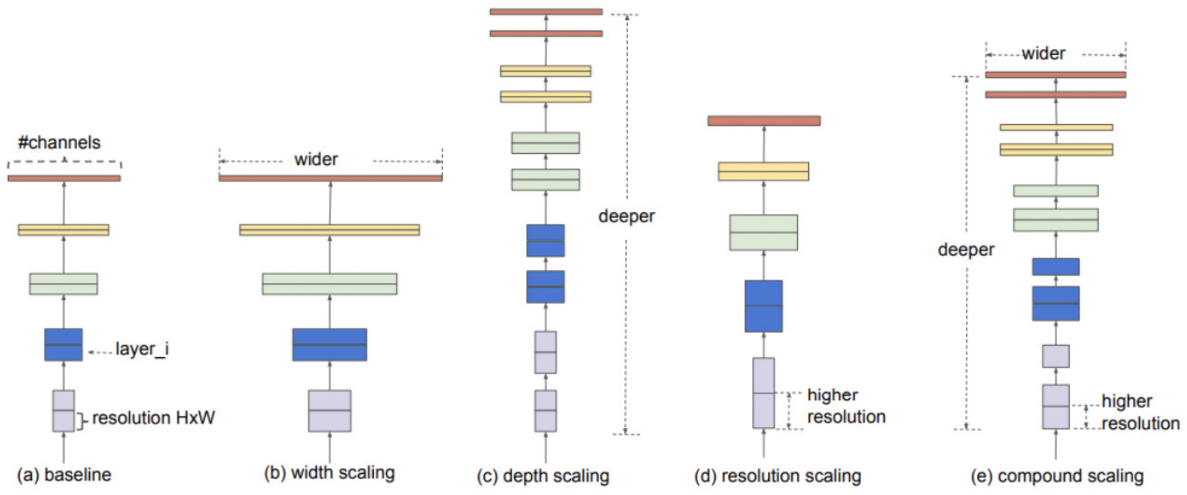


Figure 4: Model Scaling. (a) is a baseline network example; (b)-(d) are conventional scaling that only increases one dimension of network width, depth, or resolution. (e) is compound scaling method that uniformly scales all three dimensions with a fixed ratio.

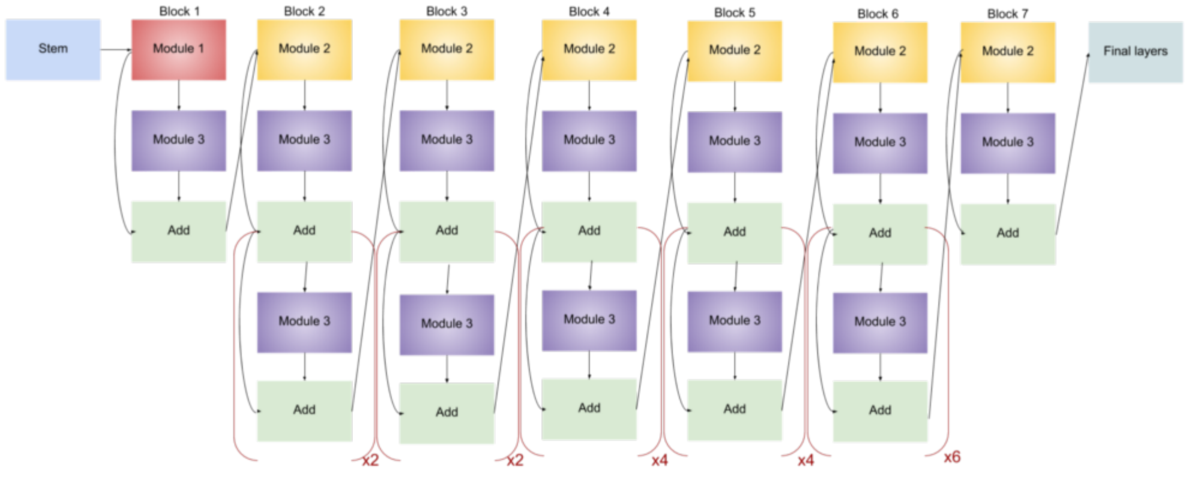


Figure 5: Architecture of EfficientNet-B4 [4]

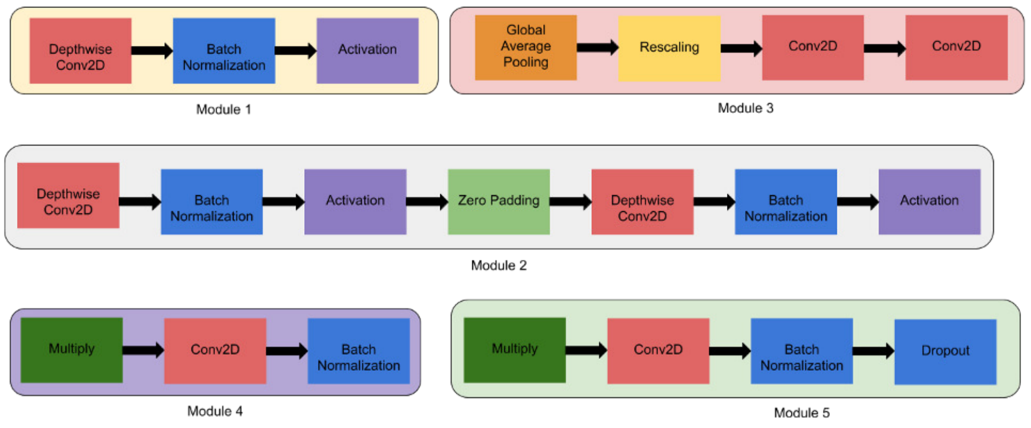
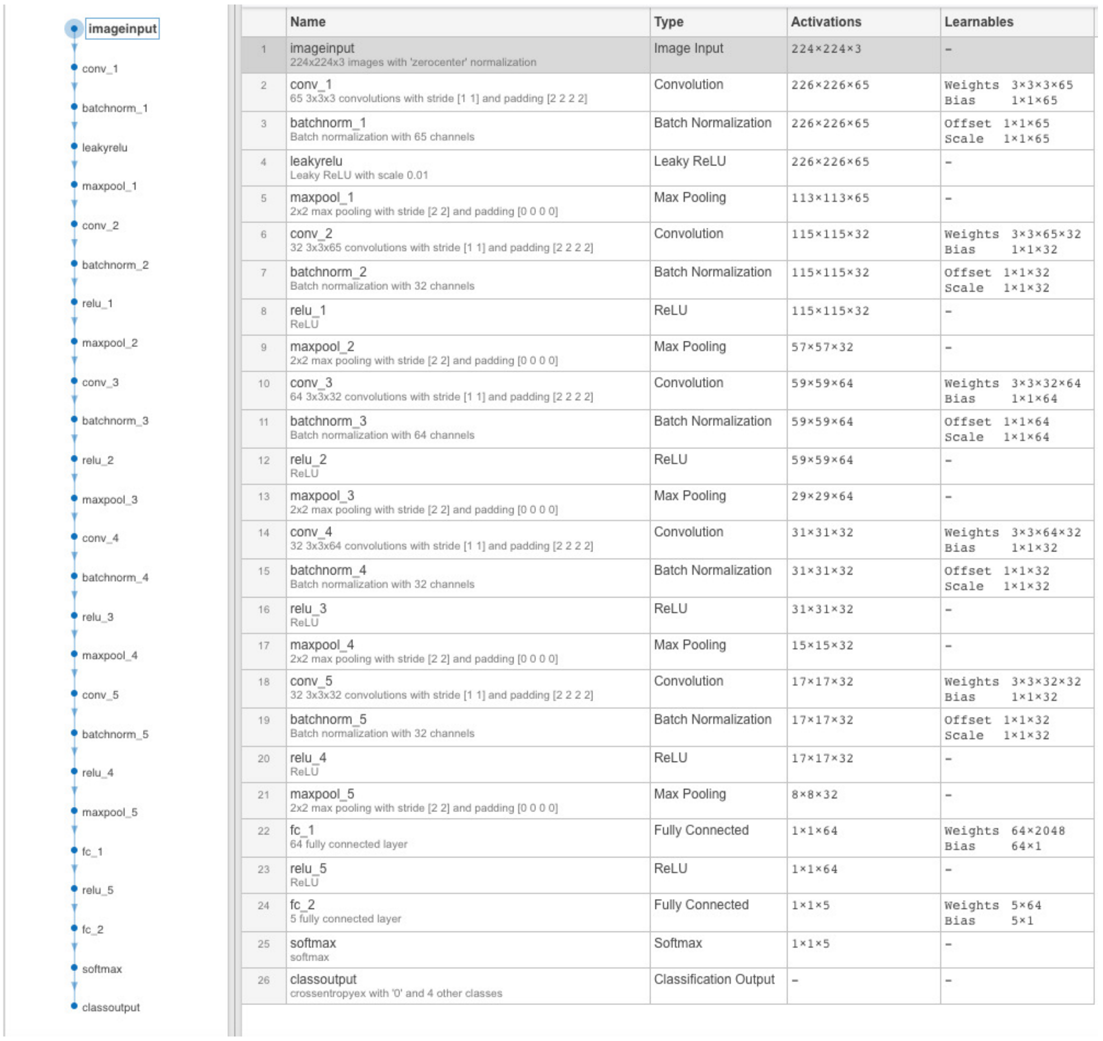


Figure 6: Modules of EfficientNet-B4 [5]

The custom CNN architecture is our own customized model build for this problem which seems to work very well considering it's a simple implementation. The network expects an image input size of 224-by-224.



	Name	Type	Activations	Learnables
1	imageinput 224x224x3 images with 'zerocenter' normalization	Image Input	224×224×3	–
2	conv_1 65 3x3x3 convolutions with stride [1 1] and padding [2 2 2 2]	Convolution	226×226×65	Weights 3×3×3×65 Bias 1×1×65
3	batchnorm_1 Batch normalization with 65 channels	Batch Normalization	226×226×65	Offset 1×1×65 Scale 1×1×65
4	leakyrelu Leaky ReLU with scale 0.01	Leaky ReLU	226×226×65	–
5	maxpool_1 2x2 max pooling with stride [2 2] and padding [0 0 0 0]	Max Pooling	113×113×65	–
6	conv_2 32 3x3x65 convolutions with stride [1 1] and padding [2 2 2 2]	Convolution	115×115×32	Weights 3×3×65×32 Bias 1×1×32
7	batchnorm_2 Batch normalization with 32 channels	Batch Normalization	115×115×32	Offset 1×1×32 Scale 1×1×32
8	relu_1 ReLU	ReLU	115×115×32	–
9	maxpool_2 2x2 max pooling with stride [2 2] and padding [0 0 0 0]	Max Pooling	57×57×32	–
10	conv_3 64 3x3x32 convolutions with stride [1 1] and padding [2 2 2 2]	Convolution	59×59×64	Weights 3×3×32×64 Bias 1×1×64
11	batchnorm_3 Batch normalization with 64 channels	Batch Normalization	59×59×64	Offset 1×1×64 Scale 1×1×64
12	relu_2 ReLU	ReLU	59×59×64	–
13	maxpool_3 2x2 max pooling with stride [2 2] and padding [0 0 0 0]	Max Pooling	29×29×64	–
14	conv_4 32 3x3x64 convolutions with stride [1 1] and padding [2 2 2 2]	Convolution	31×31×32	Weights 3×3×64×32 Bias 1×1×32
15	batchnorm_4 Batch normalization with 32 channels	Batch Normalization	31×31×32	Offset 1×1×32 Scale 1×1×32
16	relu_3 ReLU	ReLU	31×31×32	–
17	maxpool_4 2x2 max pooling with stride [2 2] and padding [0 0 0 0]	Max Pooling	15×15×32	–
18	conv_5 32 3x3x32 convolutions with stride [1 1] and padding [2 2 2 2]	Convolution	17×17×32	Weights 3×3×32×32 Bias 1×1×32
19	batchnorm_5 Batch normalization with 32 channels	Batch Normalization	17×17×32	Offset 1×1×32 Scale 1×1×32
20	relu_4 ReLU	ReLU	17×17×32	–
21	maxpool_5 2x2 max pooling with stride [2 2] and padding [0 0 0 0]	Max Pooling	8×8×32	–
22	fc_1 64 fully connected layer	Fully Connected	1×1×64	Weights 64×2048 Bias 64×1
23	relu_5 ReLU	ReLU	1×1×64	–
24	fc_2 5 fully connected layer	Fully Connected	1×1×5	Weights 5×64 Bias 5×1
25	softmax softmax	Softmax	1×1×5	–
26	classoutput crossentropyex with '0' and 4 other classes	Classification Output	–	–

Figure 7: Custom CNN model architecture

4.2 Metrics

We use six different metrics to monitor and evaluate the performance of our models on the validation and test set. These are validation loss, validation accuracy, precision, recall, F1-score, and Quadratic Weighted Kappa (QWK). Validation accuracy and loss are computed at the end of every epoch on the validation dataset and are presented as line plots in Fig. 8 and 9. The remaining four metrics are computed on the holdout test set and are presented in tabular format for the different experiments as shown in Table 1. Since accuracy, loss, precision, recall, and F1-score are very common and well defined in many literature, we will give a brief intro to what Weighted Kappa is and what it measures.

4.2.1 Quadratic Weighted Kappa (QWK)

A weighted Kappa is a metric that is used to calculate the amount of similarity between predictions and actuals. A perfect score of 1.0 is granted when both the predictions and actuals are the same. Whereas, the least possible score is -1 which is given when the predictions are furthest away from actuals. I.e. for example, a prediction of class 1 (No DR) when the ground truth is class 5 (Proliferate DR) is penalized more harshly than a prediction of 4 (severe) when the ground truth is class 5 (proliferate DR) because the first prediction is saying that the patient has no DR when they actually have the most serious case of DR (which is a dangerous misdiagnosis and can have serious consequences in the real world) while the second prediction (which is better but still wrong) is saying that the patient has a severe case of DR. The aim is to get as close to 1 as possible. Generally, a score of 0.6+ is considered to be a really good score.

The quadratic weighted kappa is calculated as follows. First, an N x N histogram matrix O is constructed, such that $O_{i,j}$ corresponds to the number of i (actual) that received a predicted value j. An N-by-N matrix of weights, W, is calculated based on the difference between actual and predicted values:

$$W_{i,j} = \frac{(i-j)^2}{(N-1)^2} \tag{1}$$

Next, an N-by-N histogram matrix of expected outcomes, E, is calculated assuming that there is no correlation between values. This is calculated as the outer product between the actual histogram vector of outcomes and the predicted histogram vector, normalized such that E and O have the same sum.

Finally, from these three matrices, the quadratic weighted kappa is calculated as:

$$QWK = 1 - \frac{\sum_{i,j} W_{i,j} O_{i,j}}{\sum_{i,j} W_{i,j} E_{i,j}} \tag{2}$$

The interested reader can read [15] for a detailed implementation note and script.

5 Experiments and Results

5.1 Experiments

We group our experiments in the following categories (the capital letters in the braces at the beginning of the description of every experiment will be used to refer to that experiment in later sections of this report):

- (A) Baseline
    - Model: ResNet-18
    - Image are resized to (224,224)
    - Batch Size: 128
    - Optimizer: SGD with initial learning rate of 10^{-3} and momentum of 0.9
    - Loss: categorical cross entorpy loss
    - Learning rate schedule: decay every 20 epochs by 0.1

    - Balanced class distribution: No

- (B) Baseline + Balanced: same as baseline but with the following changes/updates:
    - Oversampling of minority class to balance the class distribution
    - Learning rate: decay rate=0.5, step size=25 epochs
- (C) Baseline + Balanced + Finetuned: same as experiment B but with the following changes/updates:
    - Data preprocessing: all techniques shown in section 5.1.
    - Optimizer: Nesterov accelerated SGD
    - Weight decay: 5e-5
    - Learning rate: decay rate=0.1, decay step size=20 epochs
    - Xavier weight inititalization
- (D) Baseline + Balanced + Finetuned - Data augmentation: same as baseline + balanced + finetuned but with the following changes/updates:
    - Data preprocessing: all techniques shown in page 7 but without the data augmentations i.e. without random resized crop, random affine, and horizontal and vertical flips
    - Weight decay: 5e-4
- (E) EfficientNet-B4 baseline:
    - Model: EfficientNet-B4
    - Optimizer: Nesterov accelerated Stochastic Gradient Descent with momentum 0.9
    - Learning rate: inital = 0.001 decayed by a factor of 0.1 every 20 epochs
    - L2 weight decay factor: $5\text{x}10^{-4}$
    - Loss: categorical cross entropy loss
    - Class-balanced data: True
    - Input image resolution: (380,380)
    - Batch size: 15
    - Input image normalization
    - Xavier weight initialization
- (F) EfficientNet-B4 baseline + Data augmentation with the following modifications:
    - Data augmentations: random horizontal and vertical flips
    - L2 weight decay: $5\text{x}10^{-3}$
    - No bias decay
    - Learning rate warm up with warming period of 10 epochs
    - Multi-step learning rate decay with decay milestones of 30, 60, and 110 with decay factor of 0.1
- (G) Same as experiment F but with the following changes:
    - L2 weight decay factor: $5\text{x}10^{-2}$
    - Loss: Categorical Cross Entropy loss with label smoothing applied to target distribution
    - Multi-step learning rate decay with decay milestones of 30, 60, and 80 with decay factor of 0.1

- (H) Our ustom CNN architecture.
    - Model: custom CNN model shown in Fig. 6
    - Image size: 224×224
    - Optimizer: Root Mean Square Propagation (RMSProp)
    - Initial Learning Rate: 0.001
    - Batch size: 100
    - Learning Rate Schedule: Piece-wise
    - Learning Rate Decay Factor: 0.3
    - Learning Rate Drop Period: 10
    - L2Regularization: 0.001
    - Balanced class distribution: No

The code for this CNN model can be found here [6]

- (I) EfficientNet-B5: Here, training parameters are the same as EfficientNet-B4 except for the following changes:
    - Input image resolution: 456-by-456
    - Learning rate warmup period: 15 epochs
    - Learning rate decay milestones: 40th, 50th, 55th, and 60th epoch

5.2 Results

The above experiments are conducted for epochs of different duration due to reasons such as early stopping to prevent overfitting. Table 1 below shows a summary of the evaluation of our models on a holdout test set that was not used both during training and validation. As can be seen in Table 1, the family of EfficientNets (i.e. EfficientNet-B4 and Efficient-B5) showed superior performance in almost all metrics. This is expected as the very design of EfficientNets takes into account the resolution of the input image and accordingly adjusts the dimensions of a network i.e. width, depth, or resolution as shown in Fig. 3. This is done with the help of a neural architecture search using the AutoML MNAS framework, which optimizes both accuracy and efficiency (FLOPS) [14].

Comparing individual models, EfficientNet-B5 performed higher than all other models. This is expected for several reasons. First, it was trained on a relatively high-resolution 456-by-456 image compared to other models which were trained on smaller resolution images. The smaller the resolution of the image, the more we lose important information and details in the content of the image because we are basically downsampling the image from a higher resolution, where both the height and the width of the original images is in the order of 10^3, to a lower resolution which is in the order of 10^2. Also, EfficientNet-B5 is a deeper and wider model and thus has a higher capacity to learn the patterns in the training images. In line with our expectation was the performance of our ensemble of models as shown in Table 1 below. We ensembled top-2 highest performing EfficientNet-B5 models saved at different epochs by simply taking a majority vote across the prediction of the two models as our final prediction.

Table 1 Results of experiments on a holdout test set.

Experiment	Precision	Recall	F1-Score	Accuracy	Weighted Kappa
A (ResNet-18)	0.4958	0.4309	0.4112	74.25%	0.7161
B (ResNet-18)	0.5750	0.5317	0.5381	76.69%	0.7360
C (ResNet-18)	0.4597	0.4918	0.4577	63.96%	0.6601
D (ResNet-18)	0.5066	0.4747	0.4790	73.17%	0.7545
E (EfficientNet-B4)	0.5267	0.5241	0.5233	73.71%	0.7697
F (EfficientNet-B4)	0.6238	0.5978	0.5962	78.59%	0.8374
G (EfficientNet-B4)	0.5798	0.5880	0.5812	77.23%	0.8220
H (Custom CNN)	0.5210	0.5351	0.5245	73.17%	0.7861
I (EfficientNet-B5)	**0.6119**	**0.5904**	**0.5973**	**79.67%**	**0.8935**
G (Ensemble of top-2 of I)	**0.6299**	**0.6100**	**0.6146**	**80.49%**	**0.8951**

Figures 8 and 9 tell the same story as Table 1 except that now they show the performance of our model after the end of every epoch on the validation set instead of the test set (unlike Table 1). It is evident again from Figures 8 and 9 that EfficientNet-B5 performs higher than the rest of the models on the validation set. In the legend of the plots, the label **data augm*** indicates that the augmentation includes only random horizontal and vertical flips of the images and no other augmentations.

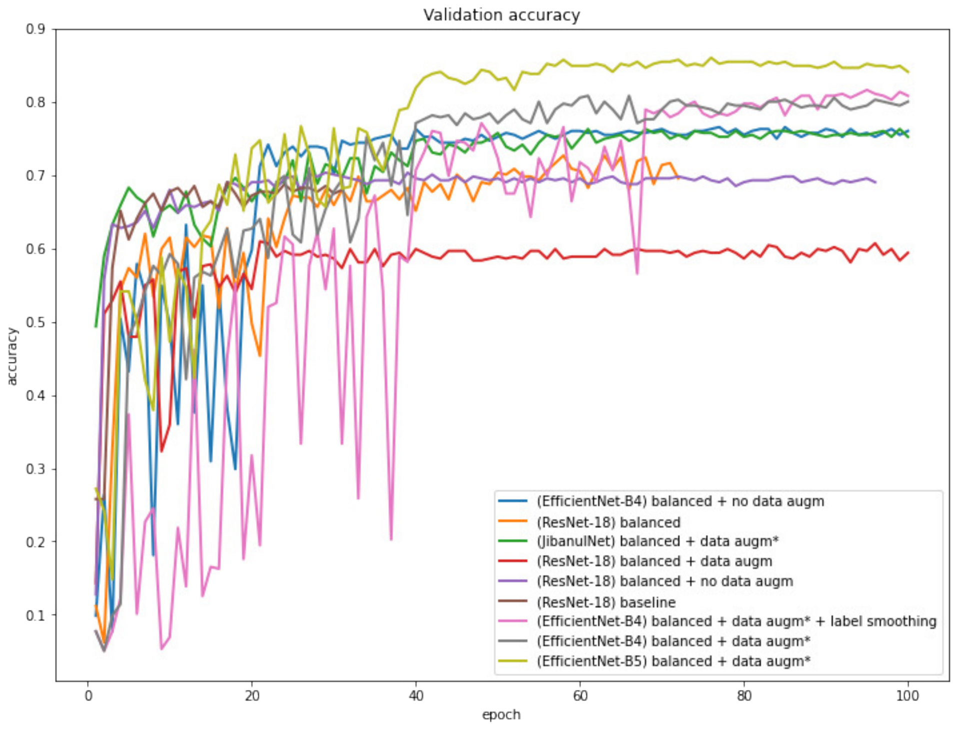


Figure 8: Validation accuracy of different experiments

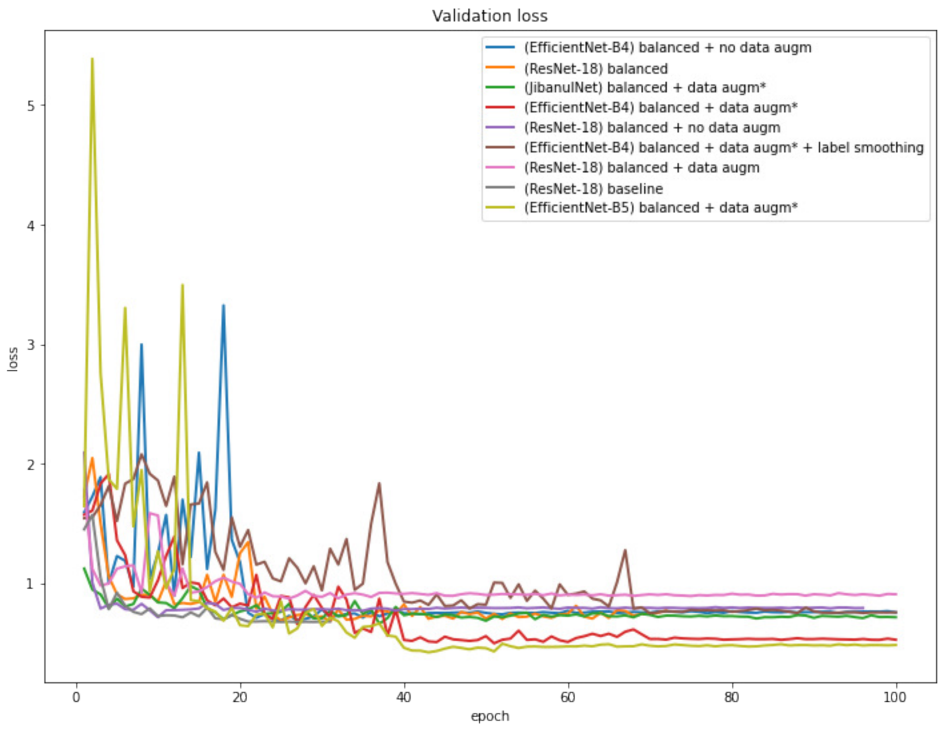


Figure 9: Validation loss of different experiments

6 Discussion

We conducted nine different experiments exploring five different models ResNet-18, EfficientNet-b4, EfficientNet-B5, Our own custom CNN, and MobileNet-V1. In all of the experiments, we tried different combinations of hyperparameter values and tricks to improve the performance of each model. Most tricks and techniques for improving ther performance of CNNs worked while some didn't work. Below we will discuss those that worked and those that didn't.

6.1 What worked

The first challenge we faced and solved was the class imbalance problem shown in Figure 1 above. A simple oversampling technique where images from minority classes are oversampled and a near-uniform distribution of classes per mini-batch was maintained. Another factor that affected the efficiency of training in terms of time is the learning decaying heuristic. We learned that it is better to have predefined milestones, epochs at which we decay the learning rate, with a higher frequency of decay on later epochs instead of fixed decay step size. The addition of the L2 weight decay on the categorical cross-entropy loss also helped in preventing our models from overfitting as our models grew in size.

Another important technique that also helped improve the performance of all models is the proper initialization of the weights of our models. We used Xavier initialization for all of our models. The goal of Xavier initialization is to initialize the weights such that the variance of the activations is the same across every layer. This constant variance helps prevent the gradient from exploding or vanishing. Proper weight initialization together with input image normalization also helped our models converge faster and prevented getting stuck in a hard-to-escape local minimum. Another observation is training on larger size images gives a much better result than training on smaller size images. This is particularly because when we resize images to low-resolution images, important details such as the thickness of blood vessels, hemorrhages (if it exists) around the blood vessels will disappear. This hurts performance as these details might be important in distinguishing among different classes of images.

6.2 What didn’t work

To our surprise, one technique that didn’t work well with our models and in fact resulted in the decline of performance is the application of different data augmentation techniques such as jittering the brightness, hue, and saturation, random cropping and resizing, and different affine transformations including shear, translation, and rotation. This is shown in Figures 8 and 9 where one of the ResNet-18 models with data augmentations of affine transforms, and random resized crops perform poorly compared to its baseline counterpart trained on images with no data augmentation. Another technique that didn’t work as we expected it the use of label smoothing on the target variable. As mentioned in this paper [16], label smoothing is reported to be one of the techniques for preventing overfitting by discouraging the output scores from being dramatically distinctive. However, as our experiments show, it performs almost the same as the model trained with a simple categorical cross-entropy loss and only makes the convergence time to be longer.

6.3 Challenges

One challenge we faced while performing our experiments is computational resource limitation. Our GPUs are not computationally capable of storing and training large models such as EfficientNet-B4 and B5 on high-resolution images using a larger batch size (which is beneficial for reasons such as less noisy training and better approximation of gradients compared to smaller batch size training). Therefore, we had to use batch sizes as small as 7 for EfficientNet-B5 so that both our data and models can fit into memory. This resulted in a much noisy training as can be seen in Figures 8 and 9 where the accuracy and loss curves oscillate with a big range even with the use of smaller initial learning rate and learning rate warm up.

7 Conclusion

In conclusion, we worked on building a deep learning method for categorizing /rating the level of Diabetic Retinopathy (if it exists) in a retinal scan image. We trained and finetuned different models and ensembled their predictions to get reasonable performance. We used ResNet-18 as our first baseline model to make sure that we have a working pipeline. We also trained other more powerful models, EfficientNet-B4 and EfficientNet-B5, on images with higher resolution. These bigger models gave us superior performance in terms of all metrics we used for comparison. Ensemble of two different versions of EfficientNet-B5s gave us the highest accuracy and quadratic weighted kappa. We used various techniques and tricks to gradually improve our model’s performance. Some techniques that worked include careful data augmentation, L2 weight decay, and learning rate warm-up. Also, some techniques that we expected to improve performance did not help us. For example, label smoothing did not help with preventing overfitting. In the future, we would like to try some new techniques such as the use of pseudo labels to further improve our models’ performance.

Reference

1. Kaggle link. `https://www.kaggle.com/c/diabetic-retinopathy-detection/overview`
2. MIT 6.S191: Introduction to Deep Learning, Lecture: Computer Vision. `http://introtodeeplearning.com/`
3. Rishab Gargeya and Theodore Leng. Automated identification of diabetic retinopathy using deep learning. Ophthalmology, 124(7):962–969, 2017.
4. Yi-PengLiu, ZhanqingLi, CongXu, JingLi, and Ronghua Liang. Referable diabetic retinopathy identification from eye fundus images with weighted path for convolutional neural network. Artificial intelligence in medicine, 99:101694, 2019.
5. Wei Zhang, Jie Zhong, Shijun Yang, Zhentao Gao, Junjie Hu, Yuanyuan Chen, and Zhang Yi. Automated identification and grading system of diabetic retinopathy using deep neural networks. Knowledge-Based Systems, 175:12–25, 2019.
6. Feng Li, Zheng Liu, Hua Chen, Minshan Jiang, Xuedian Zhang, and Zhizheng Wu. Automatic detection of diabetic retinopathy in retinal fundus photographs based on deep learning algorithm. Translational vision science technology, 8(6):4–4, 2019.
7. Xianglong Zeng, Haiquan Chen, Yuan Luo, and Wenbin Ye. Automated diabetic retinopathy detection based on binocular siamese-like convolutional neural network. IEEE Access, 7:30744– 30753, 2019.
8. Priya, R. and Aruna, P. (2012). Svm and neural network based diagnosis of diabetic retinopathy.
9. Conde, P., de la Calleja, J., Medina, M., and Benitez Ruiz, A. B. (2012). Application of machine learning to clas- sify diabetic retinopathy.
10. Harry Pratt, Frans Coenen, D. M. B. S. P. H. Y. Z. (2016). Convolutional neural networks for diabetic retinopathy.
11. Asiri, N., Hussain, M., and Aboalsamh, H. A. (2018). Deep learning based computer-aided diagnosis systems for diabetic retinopathy: A survey. CoRR, abs/1811.01238.
12. Hagos, M. T. and Kant, S. (2019). Transfer learning based detection of diabetic retinopathy from small dataset. CoRR, abs/1905.07203.
13. Rubina Sarki, Sandra Michalska, K. A. H. W. Y. Z. (2019). Convolutional neural networks for mild di- abetic retinopathy detection: an experimental study. bioRxiv.
14. ”EfficientNet: Improving Accuracy and Efficiency through AutoML and Model Scaling”, Google AI Blog, 2021. [Online]. Available: https://ai.googleblog.com/2019/05/efficientnet-improving-accuracy-and.html. [Accessed: 30- Apr- 2021]
15. ”Evaluation metrics 187 quadratic weighted kappa”, Kagglesolutions.com, 2021. [Online]. Available: http://kagglesolutions.com/r/evaluation-metrics–quadratic-weighted-kappa. [Accessed: 30- Apr- 2021]