DEEP LEARNING MODEL FOR DIAGNOSING DIABETIC RETINOPATHY

Abdullah Amin
Student# 1008803118
abdul.amin@mail.utoronto.ca

Abubaker Jassim
Student# 1008749749
a.jassim@mail.utoronto.ca

Sami Najim
Student# 1008717578
sami.najim@mail.utoronto.ca

Iftier Rahman
Student# 1009121745
iftier.rahman@mail.utoronto.ca

1 BRIEF DESCRIPTION

Approximately 537 million people globally suffer from diabetes in which 27% percent have to endure diabetic retinopathy (Federation) (Zegeye et al. (2023)). The project undertaken by our team involves the thorough examination of a plethora of retinal scans, and analyzing them in order to give a conclusive deduction on the health of an individual's eyes, as seen in Figure 1. The project was greatly motivated by the team's desire to tackle the challenge of detecting signs of Diabetic Retinopathy (DR) in its different stages as early as possible in order to expedite the treatment process for these patients. Current methods in retinopathy diagnosis are usually time consuming and expensive as it usually requires the manual inspection from ophthalmologists.

Figure 1: Retinal fundus image scans that were analyzed and classified (Karthik (2019))

The goal in this project is to develop an automated system that will be responsible for receiving retinal scan images of patients as input and outputs whether the eye is healthy or not, and if not at what stage of DR is the eye classified as. By utilizing deep learning techniques and networks, our team hopes to provide better facilitation for early prevention and treatment and increase the chances of protecting patient eyesight from blindness when suffering through DR.

The project can prove to be essential and provide important impacts to the overall improvement in the healthcare sector of society. First and foremost, the project directly deals with an issue of blindness that occurs as the result of a particular disease. Thus, our efforts will provide a significant reduction to the risk of blindness from diabetic patients further improving the healthcare quality for patients. Moreover, the success of this project can also prove to increase the accessibility of diagnosis of DR to numerous health care institutions especially in remote locations where ophthalmologists are a rarity. Furthermore, the project could not only prove to be efficient in reducing workload upon specialists, but also prove to be economically friendly upon patients themselves who don't have the means of acquiring retinal screening.

Deep learning techniques have proven to be very effective in image classification and analysis. A well trained and learned neural network can quickly access high level features of retinal scans and classify them from raw pixel data. The subtle differences between retinal scans that require extensive observations from ophthalmologists to be identified, is well suited for deep learning techniques that are proposed to accomplish such tasks. Just by passing retinal scans as input to the network, the network is easily able to analyze different features and output to the user the class representing the stage of DR that the patient possesses.

2 Individual Contributions and Responsibilities

The team meets up in discord to work on the project at least once a week on Wednesdays around 8 p.m. EST holding meetings multiple times a week when needed. We use Notion as our project management software to break down tasks that need to be done, estimate how long those tasks will take, who is doing what, and progress on the project. We use Github and Google Colab Notebooks to maintain our codebase as well as a local computer set up with a GPU for working on this project. We also have a google drive to compile our dataset and notebooks used. Due to possible issues with training and set-up, we have multiple members running training of the models on both Colab and our local computer set up for parallelization of tasks and redundancy. For example, running the baseline model on Colab and primary model on the local computer setup or testing one set up hyperparameters on colab and another locally. A further breakdown of the tasks that were completed by members can be seen in Table 1 below.

Task Category	**Brief Description**	**Person Responsible**	**Deadline**
Data Loading and Organization	Retrieve, extract, and sort the datasets used for the project. This is expected to take a few days due to one of the datasets, the Kaggle DR dataset being almost 90 GB. This as a result caused a lot of issues such as Google drive timing out on Colab due to the sheer amount of data.	**Iftier**	06/28/2024
Data Augmentation and Preprocessing	From Google Drive, the images were resized to a size of 256x256. They were then preprocessed using various researched data augmentation techniques such as using Gaussian blur, changing the brightness and contrast of image, and rotations to the image. These images were then split into training and validation using split folders.	**Abu Bakr:** Set up a python extension called split-folders. Split-folder responsible for taking a supervised data set and performing stratified sampling of the data set and placing them into training, testing, and validation folders. Uploaded the data as an image folder and loaded them into data loaders. **Iftier:** Debugged code and made adjustments to work with SVM model.	06/30/2024
Baseline Model	Implementation of SVM using the Sci-Kit learn library, splitting the dataset into two image and label numPy array for SVM, and hyperparameter tuning using GridSearchCV	**Sami:** Researched and developed the code and initialized the SVM for use for our project **Iftier:** Split data and labels into numPy arrays and trained the SVM model on Google Colab. Making adjustments to code where needed to optimize for model training efficiency and RAM usage.	07/01/2024

Initial Primary Model	Created a Convolutional Neural Network with several convolutional and fully-connected layers. Have not yet implemented residual connections and dropouts.	**Abdullah:** Helped create the primary model and made fixes on the established model **Abu Bakr:** Created the initial model with the Convoultional layers and linear layers set.	07/01/2024
Training of Primary Model	Trained dataset on processed data with validation and training progress per epoch. A small subset of the dataset was used initially to test if the model even works in the first place before starting full training. As the model outlined in this report is a first iteration, showing it is possible to use CNNs in classifying DR, we are working towards improving our model to have it perform better utilizing various techniques, data and training, outlined in our proposal.	**Abdullah:** Trained the model locally on pc, downloaded the datasets. Tried to increase the resolution of the dataset to see if the model trains better.	In Progress, 07/31/2024
Hyperparameter tuning and Evaluation	As we work each iteration of our models, both primary and baseline, it is important we tweak and play around with different parameters to improve model performance.	**Abdullah**: Created multiple models with different number of layers, changed the learning rate, epochs, batch-size, etc. **Abu Bakr:** Will work on tuning the network architecture hyperparameters (Number of Layers, pooling, kernel sizes, etc) **Iftier and Sami:** Will assist with training models on different parameters and debug code where needed for the project going forward.	In Progress, 07/31/2024

Table 1: This table is a breakdown of the larger tasks that were completed by group members in the project.

3 Data Processing

The datasets used for this project are the publicly available 35,126 image Kaggle DR (Emma Dugas (2015)), 3662 image APTOS 2019 datasets (Karthik (2019)) for training and validation while the 1748 image Messidor-1 (Guillaume PATRY) and 1058 image Messidor-2 (Patry (n.d.)) will be used for testing. All of the aforementioned datasets contain images for the 5 stages of diabetic retinopathy.

Given the size of one of the two datasets used for training (Kaggle DR Dataset) is almost 90 GB, downloading the dataset using the Kaggle API, extracting the images and organizing the images into their respective classes took longer than expected (Emma Dugas (2015)). This is due to each image being high resolution. Although we used Google Drive to store and organize our dataset, due to the sheer amount of data being processed, the Google Drive API that Colab uses would exceed Google Drive's request quota and time out quite often. After a few attempts and adjustments to the code scripts enabling this, we were able to finally prepare our dataset for data processing organized in their respective stages of 0,1,2,3, and 4. Retrieving the other datasets was much faster as they were less than 10 GB in size each. The dataset used for training, Kaggle DR and APTOS, of 38,788 images was split using a 80:20 ratio for training and validation while a combined image set of 2806 images from Messidor-1 and Messidor-2 are to be used for testing the final model once completed (Emma Dugas (2015)), (Karthik (2019)), (Guillaume PATRY), (Patry (n.d.)).

In order to perform the data splitting and organize the collected data into validation and training data sets, our team utilized the split-folder python package. This package provides python commands which can extract folders containing the raw supervised data, and perform stratified sampling with fixed ratios in order to divide the data into training, validation, and testing data sets. These data sets are then placed into a separate output folder on Google Drive. After performing the appropriate splitting, our team used the imageFolder command on Python to extract the data existing in the training and validation folders, and turn them into imageFolders after performing the proper augmentations.

Images in the training data set underwent numerous forms of augmentations using the transform library from PyTorch with the following being performed and composed on each image using transforms.Compose():

- Resizing the images: transforms.Resize() to resize all images in the dataset to be 256x256 and ensure the same dimension for all images, as seen in Figure 2.
- Flips and Rotations: To allow for better generalization of the images and not predict based on if an image is slightly crooked or if it is a left or right eye, images are randomly flipped using transforms.RandomVerticalFlip() and transforms.RandomHorizontalFlip() each with a 50% chance and rotate the image randomly between 0 to 360 degrees using transforms.RandomRotation(), as seen in top right of Figure 2.
- Brightness and Contrast: For the model to adapt to the variance in lighting conditions the retinal scans were taken in from the dataset, the brightness and contrast of each image is adjusted between -0.2 to 0.2 for contrast and -0.3 to 0.3 for brightness using transforms.ColorJitter(), as seen in Figure 2 bottom left.
- Sharpness and Blur: For the model to adapt to the different camera qualities of each image from the dataset, the image is slightly blurred using transforms. GaussianBlur() using a 5x5 kernel with a 30% chance. In addition, the image is also sharpened to enhance the exudates and blood vessels in the image by a factor of 4, as seen in the middle of the bottom row on Figure 2.

These transformations were all accomplished with the purpose of supplying our network with different representations that our images could have to better its performance in classifying DR images. Figure 2 shows how a retinal image scan may look after all augmentations are applied.

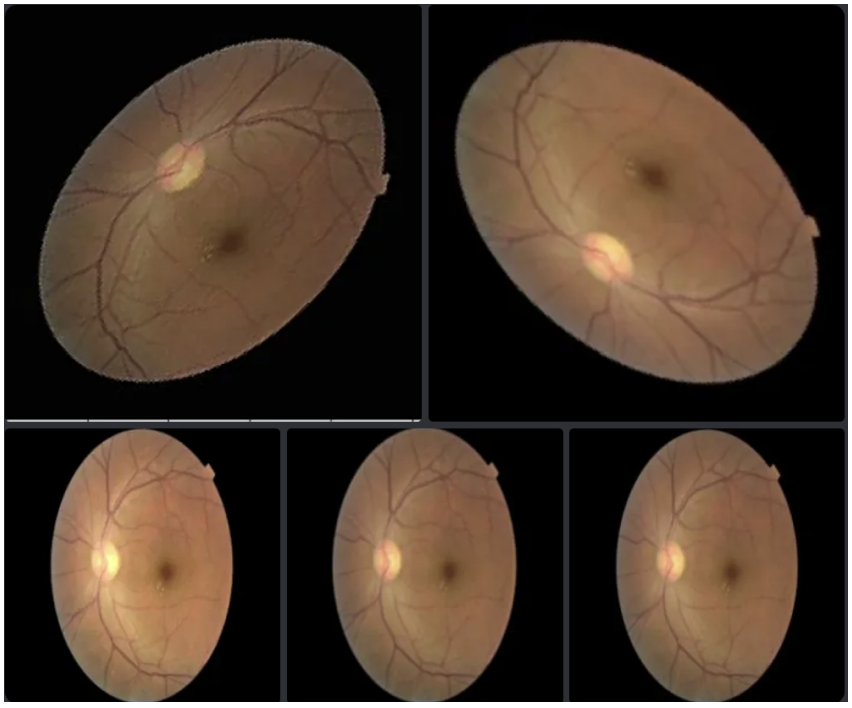

Figure 2: Retinal fundus image scans that were transformed during data augmentations (Karthik (2019)). The top right image is with a rotation applied. The bottom right image is the resized image. The bottom middle image is the image with Gaussian blur applied and sharpness manipulation. The bottom left is the image with adjusted brightness and contrast. The top left image is all the transformations applied together.

Reflecting on the entire process of collecting, extracting and sampling our data, it is without doubt that our team faced the most challenge when it came to collecting the data at the beginning stages of development, and creating a directory which sufficiently meets the requirements for supervised learning.

4 BASELINE MODEL

We made use of the support vector machine (SVM) for our choice of baseline model. The SVM was chosen over other models as our baseline model due to its ability to handle multi-class classification and its use in Diabetic Retinopathy classification research such as by M. K. Behera and S. Chakravarty (Behera & Chakravarty (2020)) and by E. V. Carrera, A. González and R. Carrera (Carrera et al. (2017)).

SVMs are generally straightforward to implement and require minimal tuning.They work by finding the most optimal hyperplane, similar to finding the line of best fit, that separates the data points by their respective classes in a higher dimensional space. Figure 3 shows how the hyperplane gives a decision boundary that splits the data for 2 classes with a margin of error. However, unlike CNNs, they do not learn patterns or geometric features as effectively.

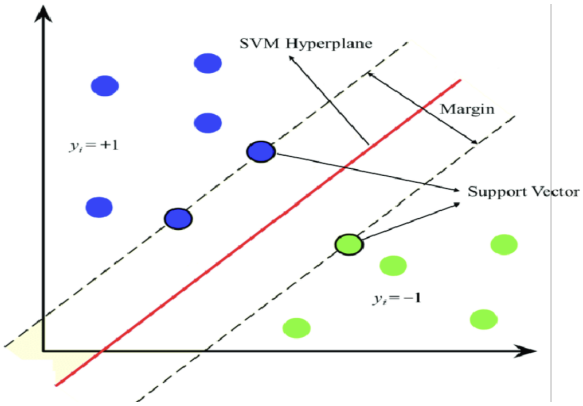

Figure 3: Hyperplane created for decision making by SVM Model (Yang & Prayogo (2022))

We used the SVM classifier built into the sci-kit learn library as our SVM model that we used to train and test

Before using the SVM, the dataset used, the Kaggle and APTOS DR datasets combined (38,788 images), was processed by grouping images into the 5 respective stages of DR, resizing, normalizing, and performing data augmentations as mentioned above and then converted into image and label numPy arrays. The data is then split into 80% training and 20% validation using train_test_split() from the sci-kit learn library.

A challenge was figuring out what hyperparameters to initialize and use in our SVM. We ended up automating the hyperparameter tuning process using GridSearchCV which allows one to input a set of parameters and cross-validate each combination to find the best combination that optimizes a model's performance. Using GridSearchCV allowed us to experiment with multiple values of C-scores and Gamma values at once using a RBF kernel. There are other kernels available such as polynomial and linear, however, RBF kernel is best for our application for its ability to generalize multi-class non-linear data effectively while requiring fewer parameters to tune.

Another challenge was ensuring our model did not use up all the RAM available. Despite resizing the images to 256x256, the system would use up all the RAM available and crash when performing hyperparameter tuning using GridSearchCV. One way we mitigated this is by using Principal Component Analysis (PCA) which reduces the dimensions of the images in the dataset. In our case, we reduced our dimensions to 100 principle components that capture the most significant information available from the dataset, where each component captures the highest sets of variance available in the data for each feature (i.e. pixels with the biggest variance in data). This as a result helped reduce the computational load when training the SVM.

As seen below in Figure 4, our baseline SVM has a validation accuracy of about 73.4%. Given the dataset is significantly imbalanced and the nature of what is being classified, the recall gives a better idea of how the model is performing as we want to see how well the model is correctly identifying positive cases. The recall of the SVM is also 73.4%.Upon analyzing the precision and recall for each class, it is seen that the model is predicting cases of no DR (stage 0) with high recall of 100%, identifying few false positives and virtually no false negatives. However, the model is struggling more to identify the other four stages of DR as seen by their lower precision and recall scores. This is especially apparent with stage 4 where the recall is 1% and the precision is 3%.

The possible reasons for these results come down to the following: the capabilities of the SVM, the massive dataset imbalance. Unlike neural networks, SVMs are not the best when it comes to capturing details and patterns as it requires more computations to come up with the optimal hyperplane as a result of more features to process. A lot of research in using SVMs in diabetic retinopathy such as in M. K. Behera and S. Chakravarty and by E. V. Carrera, A. González and R. Carrera, employs feature engineering techniques to extract the exudates in retinal

fundus scans - a key indicator for diabetic retinopathy which is not done in our case (Behera & Chakravarty (2020)), (Carrera et al. (2017)). This is one advantage of CNNs as it can do such feature extractions on its own from an image.

The dataset itself is also significantly imbalanced as seen in Figure 5 where stage 0 makes a large majority of the dataset while class 1, 2, 3, and 4, the stages of DR, make are significantly smaller. This is seen through the model performance as it can learn stage 0 samples with ease but struggles to correctly identify stages of DR. This means that we may have to artificially increase the dataset on stages 1,2,3, and 4 samples through data augmentations to reduce the impact of the imbalance. We may also have to find more datasets to train the model on or change our classification from the 5 stages of DR to distinguish between no DR, non-proliferative DR (stage 1 and 2), and proliferative DR (stage 3 and 4). This may also apply to our primary model as it too can struggle classifying the stages of DR as a result of the class imbalance. We may retrain our SVM if we decide to implement these changes for the primary model.

At the current stage, we will proceed with the results defined in Figure 4 as our benchmark in comparing our neural network primary model.

```
Validation Accuracy: 0.7338030589448359
```

	precision	recall	f1-score	support
0	0.74	1.00	0.85	4143
1	0.43	0.06	0.10	422
2	0.53	0.12	0.19	944
3	0.83	0.03	0.06	160
4	0.50	0.01	0.03	150
accuracy			0.73	5819
macro avg	0.61	0.24	0.25	5819
weighted avg	0.68	0.73	0.65	5819

```
Precision: 0.683081001973129
Recall: 0.7338030589448359
F1 Score: 0.6477256922656597
```

Figure 4: The SVM model achieved a Validation accuracy of 73.38%, Precision of 68.31%, Recall of 73.38%, and F1 Score of 64.77%

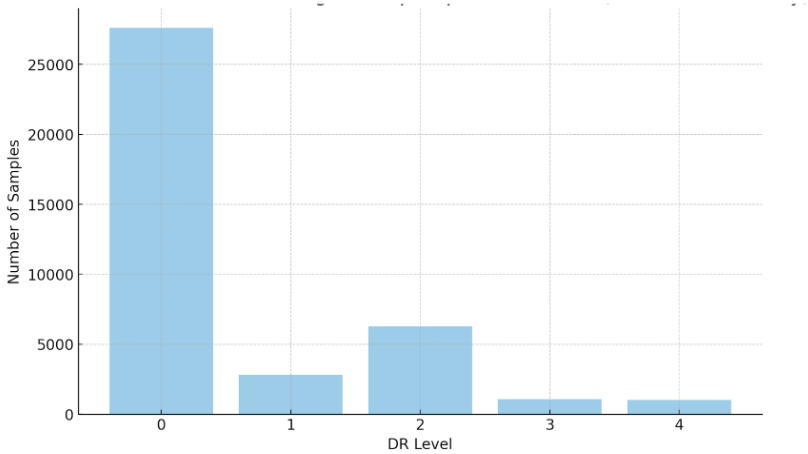

Figure 5: The breakdown of the distribution of images across the 4 classes. For context, the percentage distribution of each class is as follows: Stage 0: 71.19%, Stage 1: 7.25%, Stage 2: 16.22%, Stage 3: 2.75%, and Stage 4: 2.59%

5 PRIMAY MODEL

The architecture of this neural network for diabetic retinopathy was based on well researched neural architectures for the same application, such as the study on Automated Identification of Diabetic Retinopathy using Deep Learning by Gargeya and Leng (Gargeya & Leng (2017)). The tested model consisted of 6 convolutional layers, each with 3 by 3 filters. The filter counts for each layer were 32, 64, 128, 256, 512, and 512 respectively, as can be seen in Figure 6. The number of filters were increased in later layers to help detect low level features earlier in the network and then detect high level features. This is also why max pooling was applied with filter size 2 by 2 with stride of 2 to lower the resolution and allow the later filters a more holistic view of the eye scans. Batch normalization was also applied after every convolution layer to make sure all weights are on a relative metric and reduce the chances of over-fitting.

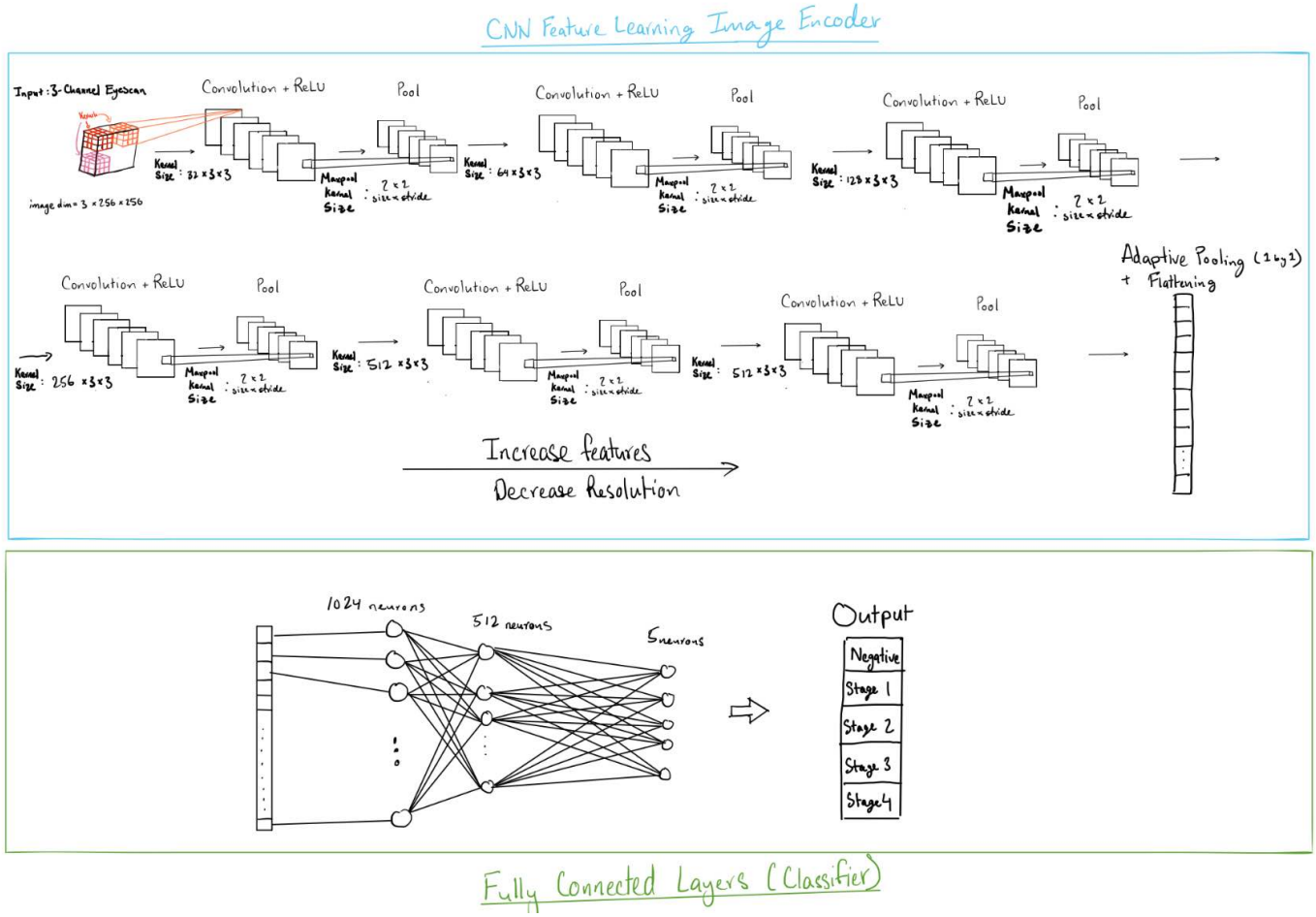

Figure 6: The neural architecture for the diabetic retinopathy classifier neural network developed.

The classifier passed the output of the convolutional layers (encoder), which was made of 2 hidden layers and the output layer. The input layer of the classifier had an input of size 512, which then produced 1024 outputs, which was then taken by the next layer and outputted back to 512 layers. Finally, the output layer produced 5 layers. ReLu was used as the activation layer for the first two layers and then SoftMax was used for the final output layer to get percentages for the prediction.

5.1 MODEL COMPLEXITY

Calculations showed that the tested model had the following parameters:

- Convolutional Layers: 3,928,384
- Batch Normalization Layers: 2,968
- Fully Connected Layers: 1,052,677

The model therefore had a grand total of 4,983,017 parameters.

5.2 RESULTS

To regularize the model to prevent over-fitting as much as possible, dropout was employed in between the fully connected with a ratio of 0.1. The learning rate of the model was tested between 0.1, 0.01, 0.001, and 0.0001, where 0.001 produced the best results. We employed mini-batch training testing a range of different batch sizes but ultimately settling with a batch size of 128, as that maximized use of the NVIDIA RTX 2080 gpu memory. The Adam optimizer was used for training and updating the loss function. The loss functions indicate overfitting of the model after about 27 epochs, as seen in Figure 7. However, prior to that, both the training loss and validation loss decrease, indicating the model was successfully training. Upon the rise of validation loss, early stopping was conducted to save the best model with the set hyper parameters.

Figure 7: This graph illustrates the training loss against the validation loss

Looking closer at the predictions by analyzing the confusion matrix (Figure 8), it was observed that the model was mostly predicting class 0 (no dr), sometimes class 1 (mild dr) and class 2 (moderate dr), and never guessing class 3 (severe dr), and class 4 (proliferative dr). This is again due to the imbalance in the training set of the data.

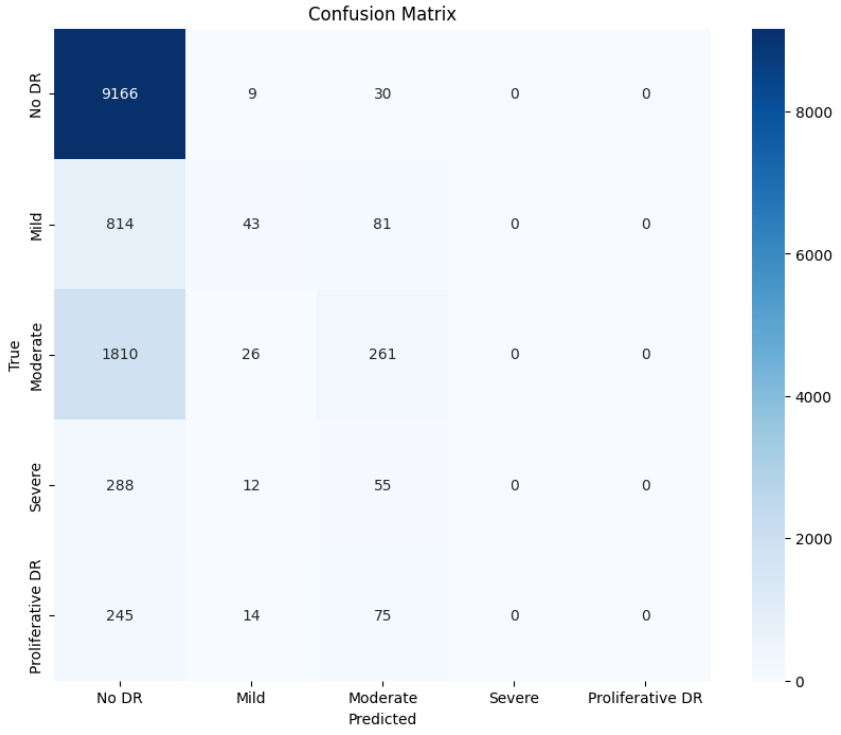

Figure 8: The confusion matrix illustrates the distribution of predictions of classes based on true labels

5.3 CHALLENGES

There were two main challenges in training the model. Firstly, training the model was resource intensive. Initially, we used google colab, which took too long on the free version. So, the model was then trained on a local machine with an RTX 2080 gpu, which was a little faster at times but still took top long to train. After increasing the depth and number of layers on the model, the time increased even more. Secondly, the dataset utilized to train and validate the model was highly disproportionate. This imbalance clearly impacted the models ability to learn the classes adequately and thus it was not able to distinguish between the classes properly.

5.4 FUTURE

The model's architecture is to be further expanded, with addition of residual shortcuts as planned to deal with the vanishing gradient problem, when the number of layers in the network is increased. Additionally, the hyperparameters of the model are to be further fine-tuned. And along with that we are to use a larger dataset to balance the number of images used to train each class. The goal is to have an even distribution of samples across each class. As not all the samples we plan to use were used to train this model, we could not limit the model's training set to an even distribution across the classes.

6 LINK TO GITHUB OR COLAB NOTEBOOK

https://github.com/UltimateForce21/DiabeticRetinopathyCNN

REFERENCES

Manoj Kumar Behera and S. Chakravarty. Diabetic retinopathy image classification using support vector machine. In *2020 International Conference on Computer Science, Engineering and Applications (ICCSEA)*, pp. 1–4, 2020.

Enrique V. Carrera, Andrés González, and Ricardo Carrera. Automated detection of diabetic retinopathy using svm. In *2017 IEEE XXIV International Conference on Electronics, Electrical Engineering and Computing (INTERCON)*, pp. 1–4, 2017.

Jorge Will Cukierski Emma Dugas, Jared. Diabetic retinopathy detection, 2015. URL https://kaggle.com/competitions/diabetic-retinopathy-detection.

Internation Diabetes Federation. IDF Diabetes Atlas 2021 – 10th edition. https://diabetesatlas.org/idfawp/resource-files/2021/07/IDF_Atlas_10th_Edition_2021.pdf. accessed Jun. 6, 2024.

Rishab Gargeya and Theodore Leng. Automated identification of diabetic retinopathy using deep learning. *Ophthalmology*, 124(7):962–969, 2017. ISSN 0161-6420. doi: https://doi.org/10.1016/j.ophtha.2017.02.008. URL https://www.sciencedirect.com/science/article/pii/S0161642016317742.

G.G. Guillaume PATRY. Messidor. https://www.adcis.net/en/third-party/messidor/. accessed Jun. 6, 2024.

Sohier Dane Karthik, Maggie. Aptos 2019 blindness detection, 2019. URL https://kaggle.com/competitions/aptos2019-blindness-detection.

Guillaume Patry. Messidor-2. ADCIS, n.d. URL https://www.adcis.net/en/third-party/messidor2/. Accessed: Jun. 6, 2024.

I-Tung Yang and Handy Prayogo. Efficient reliability analysis of structures using symbiotic organisms search-based active learning support vector machine. *Buildings*, 12(4), 2022. ISSN 2075-5309. doi: 10.3390/buildings12040455. URL https://www.mdpi.com/2075-5309/12/4/455.

Abayneh F. Zegeye, Yared Z. Temachu, and Chalie K. Mekonnen. Prevalence and factors associated with diabetes retinopathy among type 2 diabetic patients at northwest amhara comprehensive specialized hospitals, northwest ethiopia 2021. *BMC Ophthalmology*, 23(1):9, 2023. doi: 10.1186/s12886-022-02746-8. URL https://doi.org/10.1186/s12886-022-02746-8.