ILLINOIS INSTITUTE OF TECHONOLOGY

CS 584 Machine Learning

Blindness Detection

Girish Rajani-Bathija

grajanibathija@hawk.iit.edu

A20503736

Shriya Prasanna

sprasanna@hawk.iit.edu

A20521733

Bhavesh Rajesh Talreja

btalreja@hawk.iit.edu

A20516822

Prof. Yan Yan

Submission Date: 1st May 2023

IT IS DECLARED WITH MUTUAL AGREEMENT THAT EACH MEMBER HAS EQUALLY CONTRIBUTED TO THE PROJECT

1. Introduction

Diabetic retinopathy is one of the most common causes of blindness in people who have diabetes. Having high blood sugar from diabetes for a prolonged period of time can lead to diabetic retinopathy and, if not treated early, can lead to vision loss. For early detection of DR, there needs to be a manual screening exam that can identify the risk and level of someone developing this disease. However, in some rural areas, this exam can be difficult to conduct due to a lack of access to infrastructure and skilled professionals. Additionally, this manual process of screening can be very time-consuming and labor-intensive. Due to the ever-increasing demand for diabetic retinopathy screening, there arises a need to automate such a process.

The goal of this project is to use algorithms in Machine Learning and Deep Learning, such as Convolutional Neural Networks to extract features from retina images to automate this process. This will be done by performing multi-class image classification using the APTOS 2019 Blindness Detection dataset. Machine learning models allow us to speed up the process of detecting DR early and make this service available to those who may not have access to the infrastructure.

The APTOS 2019 Blindness Detection dataset available on Kaggle will be used for this project. This real-world dataset consists of retina images from multiple clinics (3,662 training images and 1,928 testing images) in rural areas of India. Each training image has been carefully examined and labeled as belonging to 1 of 5 classes - No DR, Mild DR, Moderate DR, Severe DR, and Proliferative DR. However, the testing dataset is not labeled as the goal is to label it.

2. Background and Related Work

Research conducted by Niloy, Sanaullah, Abu, and Abdullah [1] use Ensemble Learning to perform early blindness detection using the retinal images from the APTOS 2019 Blindness Detection dataset. In this study, over 600 noisy images were detected and removed by manually examining each image. Additionally, as part of the preprocessing phase, unwanted information from the images was cropped, the images were all resized to a uniform dimension, image augmentation was performed on the classes that had fewer samples in order to obtain a more balanced dataset, tone mapping the images, and finally omitting black corners. For training purposes, an ET classifier which is an ensemble learning technique based on decision trees and bagging was used for this problem. The study was able to achieve an average accuracy of 91% using this approach. For this project, our group aims to implement some of the preprocessing techniques done in this research paper and see how our Deep Learning approach compares to that of the ET classifier.

The study conducted by Carson, Darvin, Margaret, and Tony [2] used convolutional neural networks (CNNs) and various Transfer Learning models such as VGG16, GoogLeNet, and AlexNet for image classification on the APTOS 2019 Blindness Detection dataset. In this study, the preprocessing phase involved copping the images, normalizing, and performing color adjustment. The various models implemented in this paper were able to achieve an accuracy of 57.2%, 68.6%, and 74.5%. For this project, our group aims to implement a custom CNN model as well as a few of the transfer learning models used in this study, and the results will be compared.

3. Methods

3.1 Preprocessing

The retina images within this dataset required various preprocessing techniques before using them in the Machine Learning models.

3.1.1 Train/Validation/Test Split

For this project, the validation set approach was selected, and so it was necessary to split the training dataset consisting of 3,662 samples into training set (2,462 samples) and validation set (1,200 samples).

Additionally, all of the training images were in one folder, but for this multi-class classification problem, it was necessary to have all images for each class in separate folders based on their label. For both the training

and validation sets, five folders were created (one for each class), and the data were split into their respective classes using the diagnosis feature within the train_images.csv file. Now that the training, validation, and test set have been split as needed, the images had to be preprocessed.

3.1.2 Image Resizing

It was observed that the dimensions of the images were different, ranging from 474 x 358 pixels to 3388 x 2588 pixels. For an equal comparison, it is crucial that all images are uniform in size and dimension. To accomplish this, all images within the dataset were resized to 200 x 200 pixels for uniformity. Sample images before and after resizing can be observed in Figures 1 and 2 below.

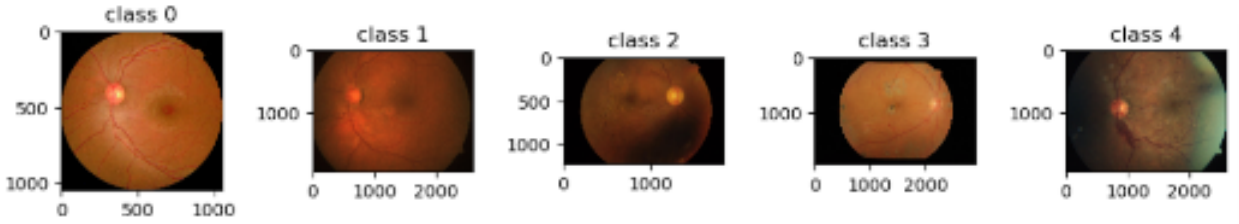

Figure 1: Showing a sample image from each class along with their dimensions before resizing

```
Class: [0] Height: 200 Width: 200
Class: [1] Height: 200 Width: 200
Class: [2] Height: 200 Width: 200
Class: [3] Height: 200 Width: 200
Class: [4] Height: 200 Width: 200
```

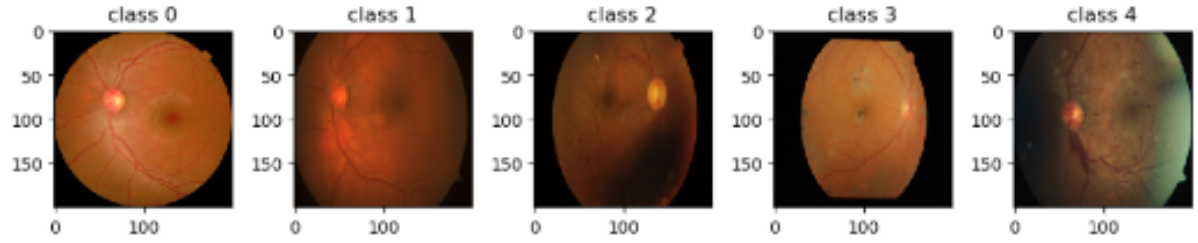

Figure 2: Showing a sample image from each class along with their dimensions after resizing

3.1.3 Image Augmentation

The final step in the data preprocessing stage was data augmentation. It was observed that the distribution of samples among the five classes varied significantly. The table below shows this distribution which allows us to conclude that the dataset is imbalanced.

As shown below, it can be seen that in both the training and validation set, classes 1, 3, and 4 have significantly fewer samples than classes 0 and 2. Therefore, to reduce the effect of bias in training, data augmentation (rotation) was performed on the classes that had fewer samples to increase the number of samples within those classes and achieve a more balanced distribution.

Number of Images in Training and Validation Set before Image Augmentation				
Class	Training Samples	Percentage Distribution	Validation Samples	Percentage Distribution
0 - No DR	1249	51%	556	46%
1 - Mild	244	10%	126	10.5%
2 - Moderate	641	26%	358	30%
3 - Severe	126	5%	67	5.5%
4 - Proliferative DR	202	8%	93	8%
Total:	**2462**		**1200**	

Figure 3: Showing the distribution of samples for each of the 5 classes before augmentation

Number of Images in Training and Validation Set after Image Augmentation				
Class	Training Samples	Percentage Distribution	Validation Samples	Percentage Distribution
0 - No DR	1249	30%	556	27%
1 - Mild	976	23%	504	24%
2 - Moderate	641	15%	358	17%
3 - Severe	504	12%	268	13%
4 - Proliferative DR	808	19%	372	18%
Total:	**4178**		**2058**	

Figure 4: Showing the distribution of samples for each of the 5 classes after augmentation

3.2 Data Augmentation

Although data augmentation had been performed within the preprocessing stage as explained above, that was only on 3 out of the 5 classes with the purpose of achieving a more balanced distribution of samples within each class to mitigate any effect of bias during training.

During training, each model that is created will be trained on training data with and without data augmentation to see how increasing all of the samples affects the model's ability to learn and generalize better. In this stage, when performing data augmentation, all samples within the training set will be augmented, and various methods will be used, such as rotation_range (40), width shift range (0.2), height shift range (0.2), shear range (0.2), zoom range (0.2), and horizontal flip.

3.3 Custom Model - CNN Architecture

The custom model for this project consists of a Convolutional Neural Network (CNN). CNN's are one of the most commonly used techniques in Deep Learning for image recognition and classification. A CNN consists of various types of layers, such as convolution layer, pooling layer, fully-connected (FC) layer, batch normalization layer, and dropout layer.

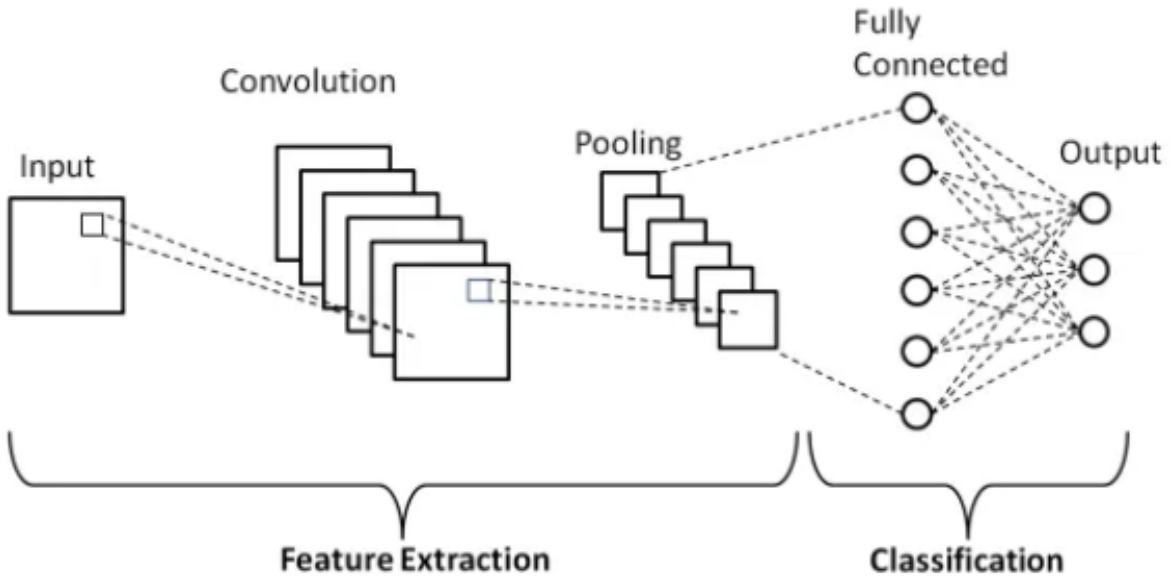

Figure 5: Showing a CNN Architecture for image classification(Source)

As shown above, the CNN performs feature extraction. The early convolution layers would extract simple features such as learning directional derivatives (they look for edges in a particular direction). The deeper convolution layers would learn more complex patterns other than edges and direction, and as the layers go deeper and deeper, it learns to identify even more complex patterns (higher-level features) that the early layers are not able to detect, such as hard exudate, a feature for classifying diabetic retinopathy.

The CNN Architecture for the custom model in this project consists of various convolution layers with different filters at each stage, 3x3 kernel size, a stride of 1, and relu activation. After every two convolution layers, a 2x2 max pooling layer was added along with a batch normalization layer and a dropout layer of 0.4. After the final convolution block, a flatten layer was added along with a few dense layers and a dropout layer of 0.2 as a regularization technique to help reduce overfitting. This network used softmax activation in the output layer, Adam optimizer with a learning rate of 0.001, and categorical cross-entropy as the loss function. More details on the custom CNN architecture will be discussed later on in section 4.1 Training Procedure.

3.4 Transfer Learning

Transfer learning is another popular technique in Deep Learning which consists of using a pre-trained model on a new problem. This is especially useful for reduced training time, increased performance, and when there is less data available. Since the pre-trained models have been trained on a large amount of data, they tend to generalize well on new tasks and can be used on datasets such as the one for this project.

In this project, various transfer learning models were implemented such as VGG16, ResNet50, and InceptionV3. These models will then be compared to the custom CNN model during evaluation.

3.4.1 VGG 16

VGG16 is a type of CNN that is one of the most popular models today. It utilizes very small 3x3 convolution filters with a stride 1 and contains approximately 138 trainable parameters. It is most commonly used for object detection and classification, being able to achieve an average accuracy of 92.7%. The architecture of VGG16 is explained further below.

The VGG16 model contains 16 learnable parameter layers but a total of 21 layers (13 convolutional layers, 5 max-pooling layers, and 3 dense layers). Each max pooling layer consists of a 2x2 filter with a stride of 2. In

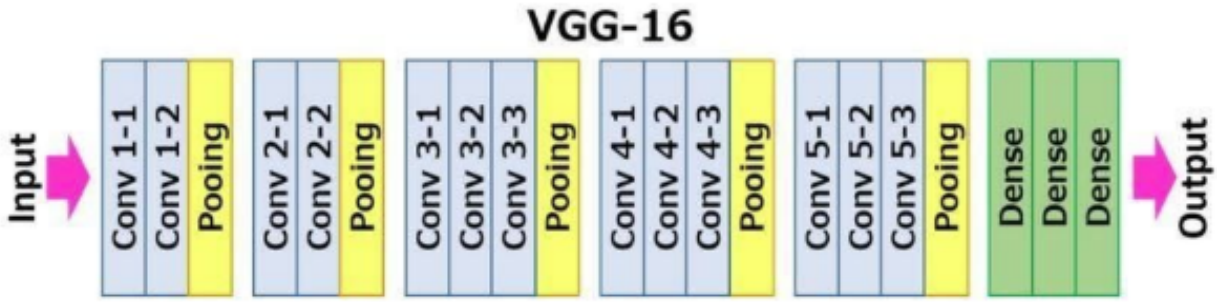

Figure 6: Showing the VGG16 architecture

Figure 6 above, the Conv 1 layers have 64 filters, the Conv 2 layers have 128 filters, the Conv 3 layers have 256 filters, and the Conv 4 and Conv 5 layers have 512 filters.

3.4.2 ResNet

The ResNet model, which was introduced in 2015, introduced a concept called Residual Network which solves the problem of vanishing/exploding gradient. The vanishing gradient problem occurs when we increase the number of layers which causes the gradient to become 0 or too large. This gradient, in turn, increases the error while training and testing. The technique of skip connections was introduced in this model, which skips some layers and connects earlier layers to later layers hence allowing information to bypass layers. This creates a residual block, and a stack of residual blocks was used to create the ResNet model. The purpose of this skip connection is so that if there are layers that will hinder the performance of the model, those layers will be skipped during regularization, which solves the problem of vanishing/exploding gradient. An illustration of this architecture is shown below in Figure 7.

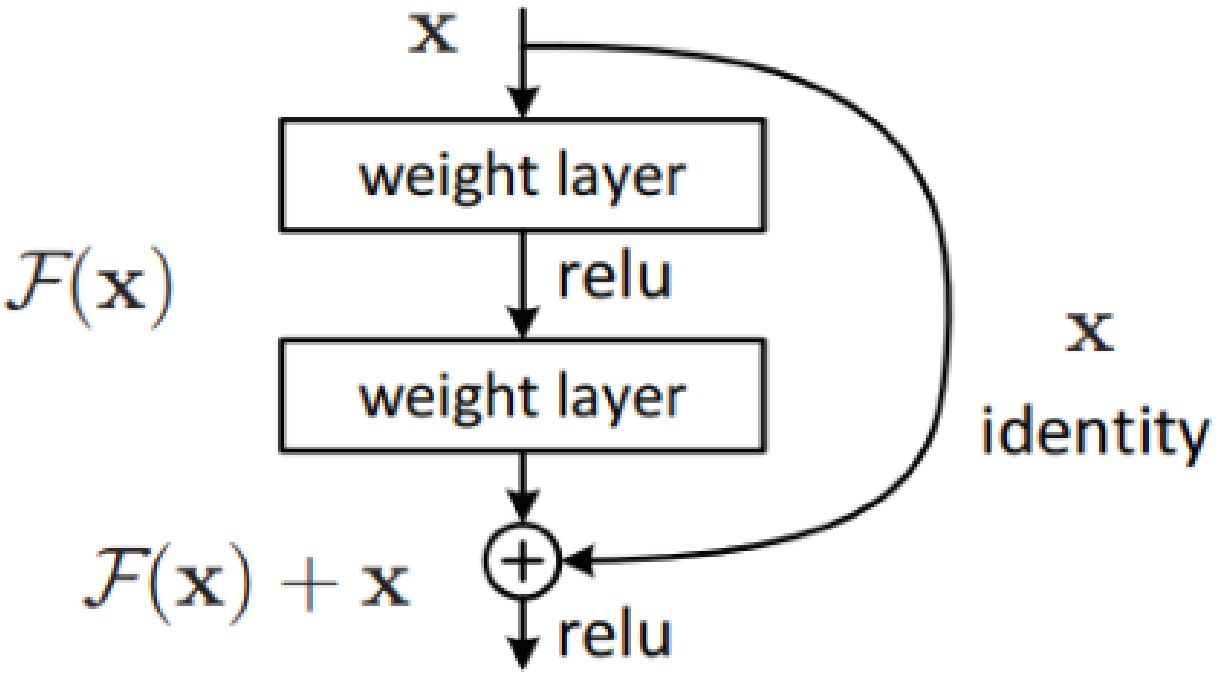

Figure 7: Showing the ResNet architecture

3.4.3 InceptionV3

The InceptionV3 is built from the older InceptionV1 (GoogLeNet) architecture from 2014. Instead of using deep layers, the InceptionV1 architecture uses parallel layers to make it wider rather than deeper. The motivation behind this is to avoid overfitting, which is caused by using multiple deep layers. Four parallel layers are used in this architecture: 1x1 convolution, 3x3 convolution, 5x5 convolution, and 3x3 max pooling

In addition, a 1x1 convolution layer was added before each of the above layers to help overcome the high computational cost associated with this architecture. The InceptionV3 takes this model a step further by containing 42 layers and optimizes it using factorization into smaller convolutions, spatial factorization into asymmetric convolutions, utility of auxiliary classifiers, and efficient grid size reduction.

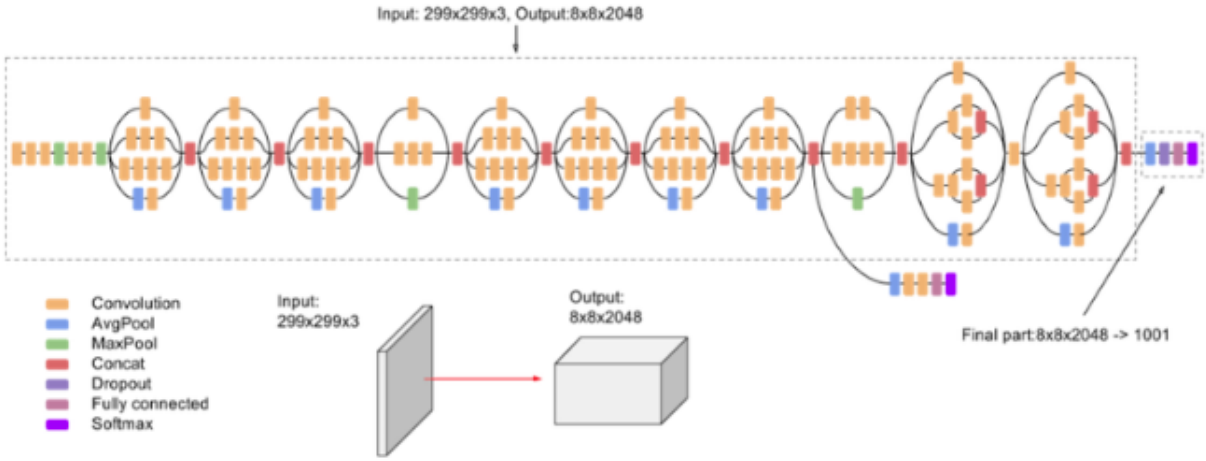

Figure 8: Showing the InceptionV3 architecture

4. Results/Discussion

4.1 Training Procedure

After preprocessing was completed, for training purposes, the training set contained 4178 images, the validation set contained 2058 images, and the testing set contained 1928 images. An initial learning rate of 0.001, Adam optimizer, and categorical crossentropy loss were used to train the models for 20 epochs.

For model evaluation, various metrics were used, such as accuracy, loss, precision, recall, F1 score, and AUC score. For this project, a total of four unique models were built (a custom CNN model, a VGG16 model, a ResNet50 model, and an InceptionV3 model). Each model was then trained twice (once without data augmentation and once with data augmentation), resulting in eight models for comparison.

The goal of performing data augmentation is to compare how each model performs with and without data augmentation and to observe if any improvements are made after training with data augmentation. For data augmentation, various parameters were used, such as rotation, width shift, height shift, shear, zoom, horizontal flip, and vertical flip. Additionally, a rescale of 1./255 was used to normalize the pixels from a range of 0-255 to 0-1.

When training using the custom CNN model, overfitting was observed after 10 epochs. To mitigate the effects of overfitting, various techniques were implemented in the model. Such methods include adding an l2 regularizer (with a learning rate of 0.01) to some layers in the model, adding weight he initialization to the conv layers, and adding dropout layers (using a rate of 0.4 in the hidden layers and 0.2 for the output layer). The effects of the mentioned regularization techniques can be seen below.

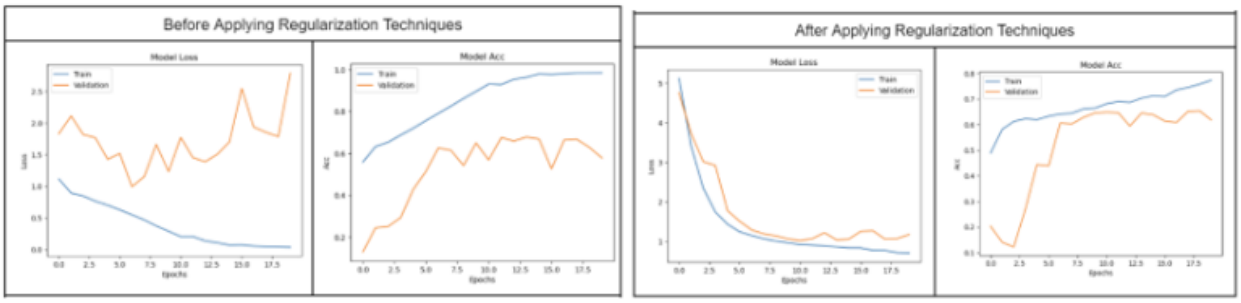

Figure 9: Showing custom CNN model results before and after regularization techniques were added to overcome overfitting.

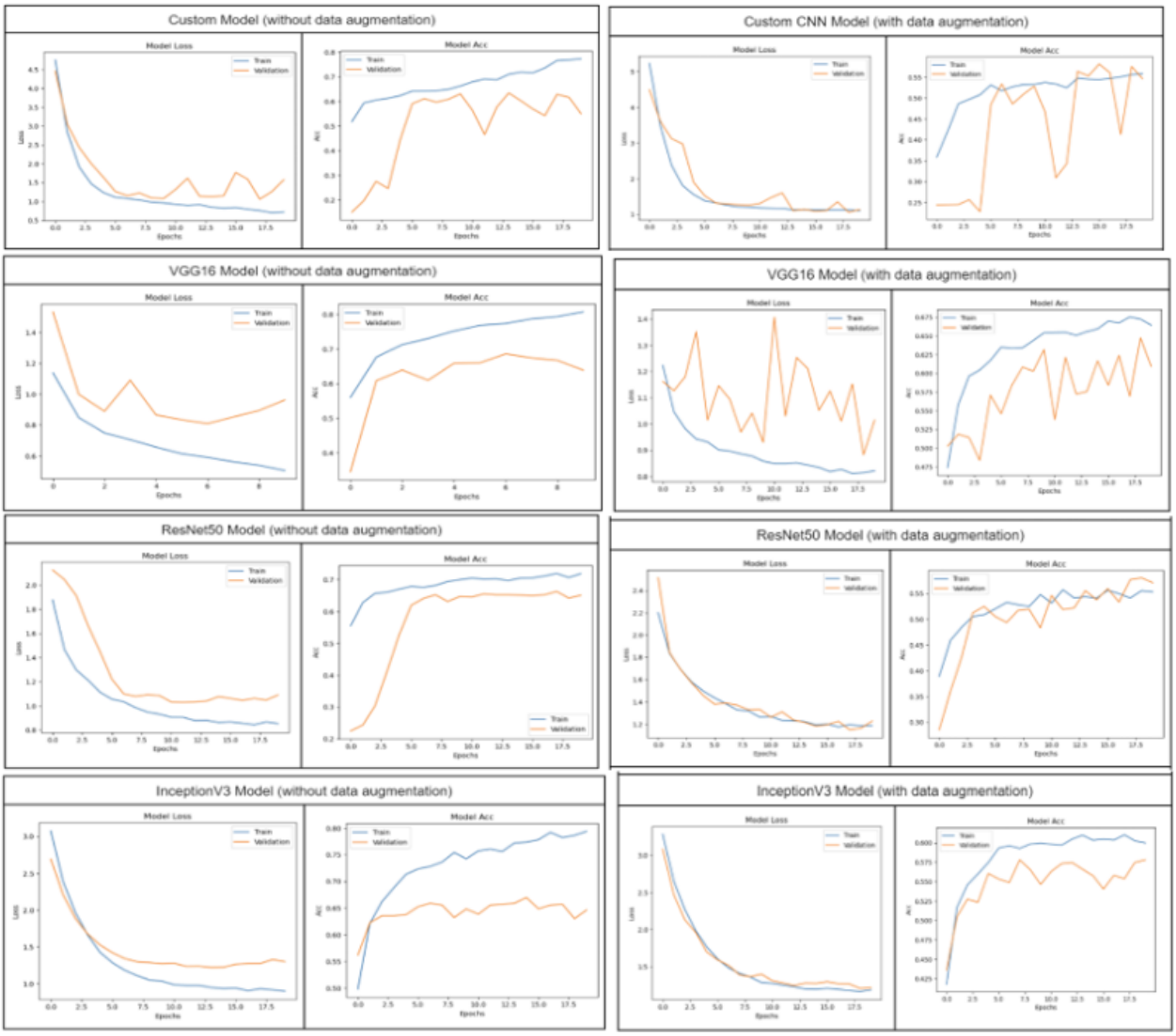

Figure 10: Showing the training/validation loss and accuracy as a function of epoch for each model trained

4.2 Training Results

From Figure 10 above, it can be seen that the VGG16 model (without augmentation) was only trained for 10 epochs while the others were trained for 20 epochs. This is because after 10 epochs, the model started to

overfit, and at that point, it had already achieved satisfactory accuracy and loss, hence, early stopping was performed. It can also be observed that minimal overfitting is seen in the models above that were trained without data augmentation.

On average, the training accuracy for the models without data augmentation ranges from 70%-80%, but the validation accuracy in all models is only able to reach 65%. After researching and numerous attempts at improving the model, it was found that due to the complexity of the retina images in this dataset, further preprocessing is required to achieve better results.

However, when looking at the models after data augmentation, we noticed that some overfitting occurred, and the performance of each model dropped. However, the training and validation accuracy is closer to each other after performing data augmentation. This gives us a more realistic idea as to how our model will perform on unseen noisy data. We can also say that without data augmentation, the models may have been memorizing the data rather than learning the patterns, which is why the training accuracy was much higher than the validation accuracy on models without data augmentation.

4.3 Evaluation Results

Model	Accuracy	Loss	Precision	Recall	F1_Score	AUC	Parameters
Custom Model	62%	1.17	0.69	0.54	0.60	0.88	5,050,149
VGG16	66%	0.91	0.71	0.59	0.64	0.90	14,789,205
ResNet50	65%	1.09	0.70	0.58	0.64	0.90	25,826,693
InceptionV3	65%	1.29	0.70	0.60	0.65	0.90	21,952,805
Custom Model (augmented)	55%	1.13	0.90	0.24	0.38	0.86	5,050,149
VGG16 (augmented)	61%	1.02	0.67	0.51	0.58	0.88	14,789,205
ResNet50 (augmented)	57%	1.23	0.72	0.37	0.49	0.86	25,826,693
InceptionV3 (augmented)	58%	1.21	0.73	0.39	0.50	0.87	21,952,805

Figure 11: Showing the evaluation of each model on validation data

The models could not be evaluated on test data. Since this dataset belongs to a Kaggle competition, the goal is to label the test data while not being provided with the true labels. Therefore, the models were evaluated on validation data.

From Figure 11 above, it can be seen that the VGG-16 and ResNet50 models (without data augmentation) perform the best when evaluated on validation data. They were both able to achieve a validation accuracy of 66% and a validation loss of 0.91 and 0.97, respectively. The other two models still do a very good job, and their performance is very close to that of VGG-16 and ResNet50.

When looking at the model results after performing data augmentation, we notice a decrease in performance throughout all four models. A possible reason for this can be that due to the possibility that further preprocessing of the image could have been done to increase robustness (based on knowledge in the domain),

the quality of images augmented would not have been very robust hence resulting in lower performing models after data augmentation.

4.4 Labelling Test Data

The final goal of this project is to label the test images. For this, the model.predict() function was used to predict each image in the test dataset using each of the eight (8) models. The labels of each model were then appended to the test.csv file hence successfully labeling the test data using all models. This labeled file can be found within the project GitHub repository. There is no way to know if the predictions are correct since the true label for the test data was not provided in the Kaggle competition.

5. Conclusion

In conclusion, our custom CNN model was able to perform relatively well when compared to popular Transfer Learning models such as VGG16, ResNet50, and InceptionV3. Adding data augmentation technique did not improve model performance, and a potential reason for this can be due to noisy images and insufficient image preprocessing because of a lack of domain knowledge. Various regularization techniques such as l2 regularizer, weight he initializer, and dropout layers helped to mitigate the effects of overfitting.

In the future, the team plans to conduct more research and study this domain properly to better preprocess the retina images. This will enable the data to be more robust, resulting in better model performance. Additionally, due to time constraints, ET Decision Trees was not able to be done, so this is something that will be looked into further. Lastly, our group plans to extend this project by developing a model that uses retinal fundus images to predict the age at which a patient on an unhealthy dietary plan is likely to suffer from blindness due to retinopathy.

6. GitHub Link for Project

Project code and data are available at

https://github.com/Girish-Rajani/Blindness_Detection_Deep_Learning

7. Contributions

Girish Rajani Bathija

- Built custom CNN model and VGG-16 model (with and without data augmentation)
- Performed Hyperparameter tuning and added regularization techniques
- Training and Evaluation and Labelled Test data
- Worked on the report

Shriya Prasanna

- Performed Image Resizing
- Built ResNet50 model and InceptionV3 model (with and without data augmentation)
- Training and Evaluation
- Worked on the report

Bhavesh Rajesh Talreja

- Performed Train/Validation Split
- Performed Image Augmentation on imbalanced classes
- Built ResNet50 model and InceptionV3 model (with and without data augmentation)
- Worked on LaTeX conversion of the report

8. References

Dataset:

https://www.kaggle.com/competitions/aptos2019-blindness-detection/data

[1] https://arxiv.org/ftp/arxiv/papers/2006/2006.07475.pdf

[2] Lam C, Yi D, Guo M, Lindsey T. Automated Detection of Diabetic Retinopathy using Deep Learning. AMIA Jt Summits Transl Sci Proc. 2018 May 18;2017:147-155. PMID: 29888061; PMCID: PMC5961805.

[3] Ramachandran, N., Hong, S.C., Sime, M.J. and Wilson, G.A. (2018), Diabetic retinopathy screening using deep neural network. Clin. Experiment. Ophthalmol., 46: 412-416. https://doi.org/10.1111/ceo.13056.

[4] Sreng, S.; Maneerat, N.; Hamamoto, K.; Panjaphongse, R. Automated Diabetic Retinopathy Screening System Using Hybrid Simulated Annealing and Ensemble Bagging Classifier. Appl. Sci. 2018, 8, 1198. https://doi.org/10.3390/app8071198.

[5]https://github.com/adityasurana/APTOS-Blindness-Detection-Kaggle/blob/master/APTOS_Blind_Detection.ipynb

[6] https://iq.opengenus.org/inception-v3-model-architecture/

[7] https://www.geeksforgeeks.org/residual-networks-resnet-deep-learning/#

[8]https://medium.com/@mygreatlearning/everything-you-need-to-know-about-vgg16-7315defb5918

[9] https://www.upgrad.com/blog/basic-cnn-architecture/

[10]https://www.analyticsvidhya.com/blog/2020/09/overfitting-in-cnn-show-to-treat-overfitting-in-convolutional-neural-networks/

[11]https://aakashgoel12.medium.com/how-to-add-user-defined-function-get-f1-score-in-keras-metrics-3013f979ce0d