Diabetic Retinopathy Detection and Classification using Different Models

1. Introduction and Problem Statement

The problem addressed in this proposal is the detection and classification of diabetic retinopathy. Diabetic retinopathy is a condition that arises due to the deterioration of blood vessels in the retina, predominantly affecting diabetic patients. If left untreated, it can lead to vision impairment and blindness. The project aims to develop an automated mechanism to identify the stage of diabetic retinopathy using different models accurately.

The real-world application of this project is in the field of medical diagnostics. By automating the detection and classification of diabetic retinopathy, healthcare professionals can save time and improve the accuracy of diagnosis. Early detection of diabetic retinopathy is crucial for timely intervention and treatment, which can prevent further progression of the disease and potentially preserve vision in diabetic patients.

2. Image Dataset Selection

The dataset used for this project is obtained from the Diabetic Retinopathy Detection competition on Kaggle. It consists of a collection of 35,126 retinal scan images specifically designed for detecting diabetic retinopathy. The original dataset can be found at the following location:

https://www.kaggle.com/datasets/tanlikesmath/diabetic-retinopathy-resized

The images in the dataset have an initial average size of 1024x724 pixels. However, they have been preprocessed and resized to a smaller size of 28x28 pixels. This resizing was done to ensure compatibility with various pre-trained deep-learning models.

The dataset is organized into five directories, each representing a different severity or stage of diabetic retinopathy:

- 0 - No_DR (No Diabetic Retinopathy)
- 1 - Mild
- 2 - Moderate
- 3 - Severe
- 4 - Proliferate_DR (Proliferative Diabetic Retinopathy)

Each directory contains retinal scan images corresponding to the respective severity level. This categorization allows for the classification of images based on the presence and severity of diabetic retinopathy.

The availability of this dataset provides an opportunity to develop and evaluate algorithms for diabetic retinopathy detection. The resized images facilitate easier and faster experimentation with deep-learning models. Researchers can utilize this dataset to explore various approaches and develop innovative solutions in the field of diabetic retinopathy diagnosis.

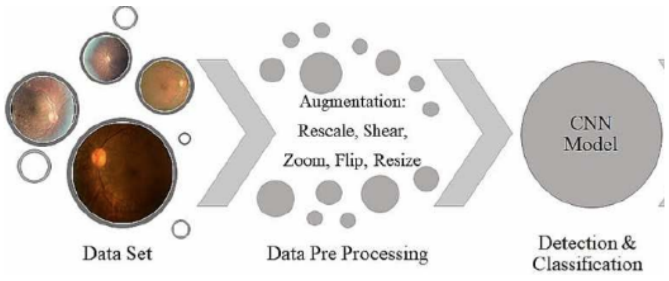


Figure 1. Proposed Model template

3. Proposed Methodology

Decision Tree Supervised Learning: Decision trees are a popular supervised learning technique that can be applied to classification tasks. In this approach, a decision tree model is trained using the dataset of color fundus photographs. The decision tree algorithm learns a series of hierarchical decision rules based on the features extracted from the images. These rules are then used to classify the retinal images into different stages of diabetic retinopathy.

Decision Tree Semi-Supervised Learning: In cases where the dataset has limited labeled data, a semi-supervised learning approach can be considered. In this methodology, a small portion of the dataset is labeled, and a decision tree model is trained using these labeled examples. The trained model can then be used to classify the remaining unlabeled data based on the learned decision rules. This approach leverages both labeled and unlabeled data to improve the overall classification accuracy.

Deep Neural Network (DNN) Supervised Learning: Convolutional Neural Networks (CNNs) are powerful models for image classification tasks. In this proposed methodology, a CNN-based supervised learning approach is chosen for the detection and classification of diabetic retinopathy using the dataset of color fundus photographs. CNNs are particularly effective in capturing spatial features and patterns in images, making them well-suited for this task.

These methodologies offer alternative approaches to the detection and classification of diabetic retinopathy. Researchers would need to evaluate and compare the performance of these methods carefully, considering factors such as available labeled data, dataset size, the complexity of the problem, and computational resources, to determine the most suitable approach for the task at hand.

4. Attempts at solving the problem (preliminary results)

In this phase of the project, we conducted preliminary experiments to solve the problem at hand. The process began by reading a sequence of image names from a CSV file. Each image was then processed by resizing it to 28x28 pixels and normalizing the pixel values. The flattened pixel values were saved to a new CSV file.

Next, we retrieved the column names from the file and added them as the first row in a DataFrame. To ensure proper alignment, we reset the index of the DataFrame to start from 0. This preprocessing step ensured that the column names were correctly incorporated into the DataFrame. Once the dataset was properly preprocessed, we split it into training and validation datasets using an 80:20 ratio. This allowed us to have separate datasets for training and evaluating our model.

We then created a Convolutional Neural Network (CNN) model to classify the images. The model was initialized as a Sequential model, which enables stacking layers in a linear manner.

The first layer added to the model was a 2D Convolutional layer (Conv2D) with 32 filters, each with a kernel size of 5x5 pixels. The ReLU activation function was used, and the input shape of the layer was set to (28, 28, 1), indicating that grayscale images of size 28x28 pixels were expected.

Another 2D Convolutional layer with the same specifications as the previous one was added to the model. To reduce the spatial dimensions of the feature maps, a Max Pooling layer (MaxPool2D) with a pool size of 2x2 pixels was added. This layer helped in capturing the most important features while reducing computational complexity. To prevent overfitting, a Dropout layer (Dropout) with a rate of 0.25 was introduced. Dropout randomly sets a fraction of input units to 0 during training, which helps in regularization.

Two more pairs of Convolutional layers, Max Pooling layers, and Dropout layers were added, with increased filter size and reduced spatial dimensions. This allowed the model to learn more complex patterns and further reduce the spatial dimensions. The feature maps obtained from the previous layers were flattened into a 1D vector using the Flatten layer. A fully connected Dense layer (Dense) with 256 neurons and ReLU activation was added to the model.

To further prevent overfitting, another Dropout layer with a rate of 0.5 was added.

Finally, a Dense layer with five neurons (equal to the number of classes) and softmax activation was added. This layer produced the final output probabilities for the classification task.

For training the model, RMSprop was used as the optimizer with a learning rate of 0.001, rho of 0.9, and epsilon of 1e-88. The categorical loss function was employed, and accuracy metrics were used for evaluation. The model was trained on the dataset for 10 epochs with a batch size of 100. After training, the loss and accuracy values were plotted against the epochs to analyze the model's performance.

The accuracy achieved on the training dataset was 73.45%, while the validation accuracy reached 73.50%. These results provide an initial assessment of the model's performance, but further analysis and optimization may be required to improve its accuracy and generalization capabilities.

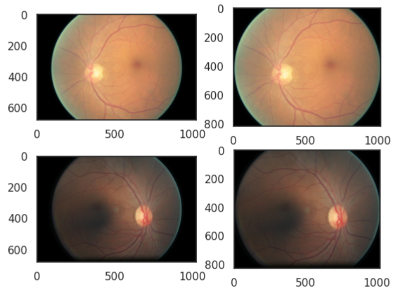
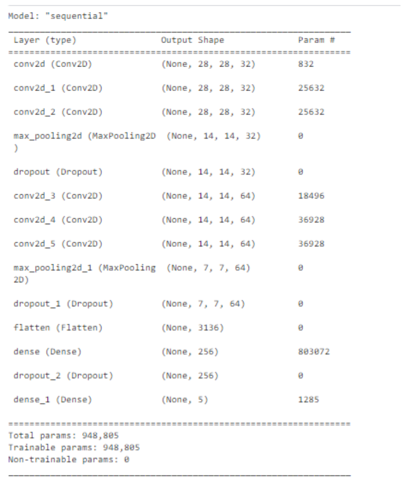


Figure 2. Preprocessed Data

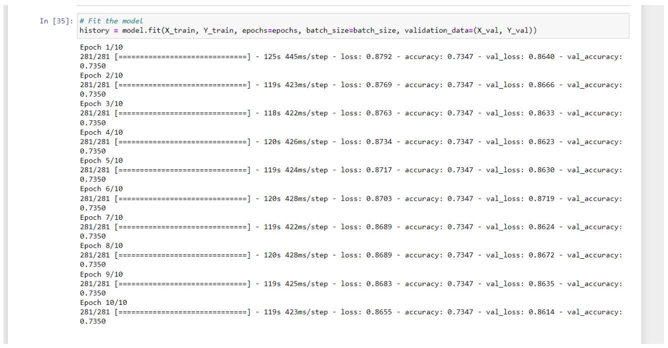


```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 28, 28, 32)        832

 conv2d_1 (Conv2D)           (None, 28, 28, 32)        25632

 conv2d_2 (Conv2D)           (None, 28, 28, 32)        25632

 max_pooling2d (MaxPooling2D  (None, 14, 14, 32)       0
 )

 dropout (Dropout)           (None, 14, 14, 32)        0

 conv2d_3 (Conv2D)           (None, 14, 14, 64)        18496

 conv2d_4 (Conv2D)           (None, 14, 14, 64)        36928

 conv2d_5 (Conv2D)           (None, 14, 14, 64)        36928

 max_pooling2d_1 (MaxPooling  (None, 7, 7, 64)         0
 2D)

 dropout_1 (Dropout)         (None, 7, 7, 64)          0

 flatten (Flatten)           (None, 3136)              0

 dense (Dense)               (None, 256)               803072

 dropout_2 (Dropout)         (None, 256)               0

 dense_1 (Dense)             (None, 5)                 1285

=================================================================
Total params: 948,805
Trainable params: 948,805
Non-trainable params: 0
_________________________________________________________________
```

Figure 3. Model Summary

5. Future plans and improvement

To improve the CNN model, we'll explore the following strategies and techniques: Increase model capacity: Increase the complexity of the model by adding more convolutional layers, increasing the number of filters in each



```
In [35]: # Fit the model
         history = model.fit(X_train, Y_train, epochs=epochs, batch_size=batch_size, validation_data=(X_val, Y_val))

         Epoch 1/10
         281/281 [==============================] - 125s 445ms/step - loss: 0.8792 - accuracy: 0.7347 - val_loss: 0.8640 - val_accuracy:
         0.7350
         Epoch 2/10
         281/281 [==============================] - 119s 423ms/step - loss: 0.8769 - accuracy: 0.7347 - val_loss: 0.8666 - val_accuracy:
         0.7350
         Epoch 3/10
         281/281 [==============================] - 118s 422ms/step - loss: 0.8763 - accuracy: 0.7347 - val_loss: 0.8633 - val_accuracy:
         0.7350
         Epoch 4/10
         281/281 [==============================] - 120s 426ms/step - loss: 0.8734 - accuracy: 0.7347 - val_loss: 0.8623 - val_accuracy:
         0.7350
         Epoch 5/10
         281/281 [==============================] - 119s 424ms/step - loss: 0.8717 - accuracy: 0.7347 - val_loss: 0.8630 - val_accuracy:
         0.7350
         Epoch 6/10
         281/281 [==============================] - 120s 428ms/step - loss: 0.8703 - accuracy: 0.7347 - val_loss: 0.8719 - val_accuracy:
         0.7350
         Epoch 7/10
         281/281 [==============================] - 119s 422ms/step - loss: 0.8689 - accuracy: 0.7347 - val_loss: 0.8624 - val_accuracy:
         0.7350
         Epoch 8/10
         281/281 [==============================] - 120s 428ms/step - loss: 0.8689 - accuracy: 0.7347 - val_loss: 0.8672 - val_accuracy:
         0.7350
         Epoch 9/10
         281/281 [==============================] - 119s 425ms/step - loss: 0.8683 - accuracy: 0.7347 - val_loss: 0.8635 - val_accuracy:
         0.7350
         Epoch 10/10
         281/281 [==============================] - 119s 423ms/step - loss: 0.8655 - accuracy: 0.7347 - val_loss: 0.8614 - val_accuracy:
         0.7350
```

Figure 4. Model Fitting for 10 epochs

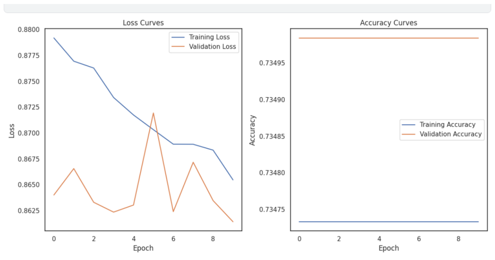


Figure 5. Loss Accuracy Vs Epoch

layer, or increasing the number of neurons in the fully connected layers. Adjust hyperparameters: Experiment with different hyperparameters such as learning rate, batch size, optimizer, and regularization techniques (e.g., dropout rate) to find optimal settings that improve model performance. Data augmentation: Apply data augmentation techniques to expand the training dataset artificially. This can include random rotations, translations, flips, and zooms on the training images, which helps the model generalize better and reduces overfitting. Optimize training process: Experiment with different optimization algorithms or adaptive learning rate techniques (e.g., Adam, RMSprop) to improve convergence and training speed.

6. Bibliography

References

[1] Russell, S., & Norvig, P. (2020). *Artificial Intelligence: A Modern Approach* (4th ed.). Upper Saddle River, NJ: Pearson. Available at: `https://aima.cs.berkeley.edu/index.html`

[2] Luger, G. F. (2005). *Artificial Intelligence: Structures and Strategies for Complex Problem Solving* (5th ed.). Addison Wesley.

[3] Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT Press. Available at: `https://www.deeplearningbook.org/`

[4] Dive into Deep Learning. Available at: `http://d2l.ai/`

[5] Géron, A. (2019). *Hands-on machine learning with Scikit-Learn, Keras and TensorFlow: concepts, tools, and techniques to build intelligent systems* (2nd ed.). O'Reilly.