Aim: Create image generators and Deep ANN model

# Project structure

A screenshot of a computer

Description automatically generated

# Preprocess.py

class Dataprocess contains various methods for data processing, exploration, and preparation for a machine learning model, particularly for binary classification of diabetic retinopathy images.

1. generate\_images\_dataset: This method takes a data\_path as input and generates a DataFrame containing image paths and corresponding labels. It iterates through subdirectories in the data\_path and extracts image paths and labels.
2. Generate\_new\_feature\_in\_csv: This method takes a CSV file path as input, reads the data from the CSV, and generates new features based on mappings provided in the method.
3. data\_EDA: Performs exploratory data analysis (EDA) on the dataset by creating a count plot of image labels using the seaborn library.
4. data\_separation: Splits the dataset into training, validation, and testing sets using train\_test\_split.
5. mkdir\_separate\_image: Creates working directories for train, validation, and test sets, and copies images to their respective directories based on the dataset split.
6. ImageDataGenerator\_Data: Uses Keras' ImageDataGenerator to generate batches of images for training, validation, and testing from the directory structure created by mkdir\_separate\_image.

# Classifcation\_model.py

‘Classifcation\_model: Defines a class that will contain methods related to creating and working with classification models.

1. simple\_model: Defines a simple neural network model using the Sequential API. The model consists of a Flatten layer (for flattening the input), followed by several Dense layers with different numbers of neurons and ReLU activation functions. The output layer has 2 neurons with a softmax activation function, suitable for binary classification.
2. compile\_model: Takes a model as input and compiles it using the Adam optimizer, categorical cross-entropy loss function, and accuracy as the metric.
3. summary\_model: The summary\_model method uses matplotlib to plot the training history. The first plot shows the training and validation accuracy over epochs, and the second plot shows the training and validation loss over epochs.

# Main\_method.py

1. Data Preprocessing:

data\_path = r"C:\Users\Bhaskar Marapelli\Downloads\gaussian\_filtered\_images\gaussian\_filtered\_images": Specifies the path to the directory containing diabetic retinopathy images.

da=ps.Dataprocess(): Creates an instance of the Dataprocess class from the preprocess module.

dataset\_df = da.generate\_images\_dataset(data\_path): Generates a DataFrame (dataset\_df) containing image paths and labels by calling the generate\_images\_dataset method of the Dataprocess class.

csv\_filename = "diabetic\_retinopathy\_dataset.csv": Specifies the name of the CSV file to which the dataset information will be saved.

dataset\_df.to\_csv(csv\_filename, index=False): Saves the dataset information to a CSV file named "diabetic\_retinopathy\_dataset.csv".

data=da.Generate\_new\_feature\_in\_csv(csv\_filename): Reads the CSV file, generates new features, and creates a DataFrame (data) using the Generate\_new\_feature\_in\_csv method.

binary\_csv="binary\_dataset.csv": Specifies the name of the CSV file to which the binary dataset information will be saved.

data.to\_csv(binary\_csv, index=False): Saves the binary dataset information to a CSV file named "binary\_dataset.csv".

da.data\_EDA(binary\_csv): Performs exploratory data analysis on the binary dataset by calling the data\_EDA method.

1. Image Data Preparation:

train\_batches, val\_batches, test\_batches=da.ImageDataGenerator\_Data(binary\_csv): Generates batches of images for training, validation, and testing using the ImageDataGenerator\_Data method.

1. Model Creation and Training:

obj=cm.Classifcation\_model(): Creates an instance of the Classifcation\_model class from the classification\_model module.

model=obj.simple\_model(): Creates a simple neural network model using the simple\_model method.

model=obj.compile\_model(model): Compiles the model using the compile\_model method.

history=model.fit(train\_batches, epochs=10, validation\_data=val\_batches): Fits the model to the training data for 10 epochs using the training batches and validates it on the validation data.

obj.summary\_model(history): Visualizes the training history using the summary\_model method.