# INTRODUCTION

Diabetic Retinopathy (DR) is a severe eye condition that can lead to blindness if not detected early. This project aims to classify retinal images into different severity levels using deep learning techniques. The notebook follows a structured approach, including data preprocessing, model training, evaluation, and optimization to achieve high accuracy. The goal is to optimize the model’s performance by freezing earlier layers, training only the final layers, and implementing advanced training techniques such as label smoothing, learning rate scheduling, and hyperparameter tuning. The final trained model is evaluated on validation accuracy, and strategies for further improvement are discussed.

**Setup and Dependencies**

The environment is set up using various libraries essential for deep learning and data processing:

NumPy & Pandas: For numerical operations and data manipulation.

Matplotlib: For visualization.

Torch & Torchvision: Deep learning framework for building and training neural networks.

PIL: For image processing.

TQDM: For progress tracking.

KaggleHub: For downloading datasets directly from Kaggle.

The computation is performed using GPU if available, ensuring faster training and model execution.

# Dataset Overview

The dataset contains images categorized into five severity levels of diabetic retinopathy. The data is split into three sets:

Training Set: Used to train the model.

Validation Set: Used for model evaluation and hyperparameter tuning.

Test Set: Used to assess final model performance.

Data is stored in directories:

* /kaggle/input/diabetic-retinopathy-balanced/content/Diabetic\_Balanced\_Data/train
* /kaggle/input/diabetic-retinopathy-balanced/content/Diabetic\_Balanced\_Data/test
* /kaggle/input/diabetic-retinopathy-balanced/content/Diabetic\_Balanced\_Data/va

# Data Pre-Processing

The dataset contains retinal images categorized into five classes based on the severity of diabetic retinopathy.

Data Transformations

**Training Data Augmentation**:

Random resizing, cropping, horizontal flipping, rotation, and color jittering are applied to the training images to increase the diversity of the training data and prevent overfitting.

**Validation Data Transformation**:

For validation, only resizing, center cropping, and normalization are applied to ensure consistency in evaluation.

**Data Loading**

The dataset is loaded using ImageFolder, which automatically categorizes images based on their folder structure.

Data loaders are created for both training and validation datasets.

# Model Development

The InceptionV3 model is loaded with pre-trained weights, and the final fully connected layer is modified to output 5 classes (one for each severity level).

*model = models.inception\_v3(weights=models.Inception\_V3\_Weights.DEFAULT)  
model.fc = nn.Linear(2048, 5) # Adjust to 5 output classes  
model.to(device)*

To leverage the powerful feature extraction capabilities of InceptionV3, the model is loaded with pre-trained weights from ImageNet. The default fully connected layer of InceptionV3 is designed for 1,000 classes, so it is replaced with a new fully connected (FC) layer tailored for this specific task. The modified FC layer has an output of five neurons, corresponding to the five severity levels of DR, with a softmax activation function to output class probabilities.

Since deep CNN models contain millions of parameters, training all layers from scratch would require a massive amount of labeled data and computational resources. Instead, transfer learning is applied by freezing the early layers of InceptionV3 and fine-tuning only the later layers.

Freezing the initial layers helps preserve low-level features such as edges and textures, which are useful for many image recognition tasks. The deeper layers, responsible for higher-level abstractions, are fine-tuned to learn domain-specific patterns related to diabetic retinopathy.

*for param in model.parameters():*

*param.requires\_grad = False # Freeze all layers*

*for param in model.Mixed\_6.parameters():*

*param.requires\_grad = True*

*for param in model.Mixed\_7.parameters():*

*param.requires\_grad = True*

*for param in model.fc.parameters():*

*param.requires\_grad = True*

# **Training Strategy and Optimization Techniques**

To ensure stable and efficient training, several optimization techniques are employed. The loss function used is cross-entropy loss, which effectively handles multi-class classification. To prevent overfitting and improve generalization, dropout is applied in the final layers of the model. Additionally, weight decay is included in the optimizer to introduce regularization, reducing the risk of excessive reliance on specific features.

The learning rate is carefully set at a low value of 0.00005, combined with momentum, allowing the model to converge steadily without drastic fluctuations. Data augmentation techniques such as random cropping, horizontal flipping, rotation, and color jittering enhance the model's ability to recognize patterns under varying conditions. The validation set, on the other hand, undergoes only normalization to ensure consistency in evaluation.

To improve model efficiency, early stopping is implemented, monitoring validation accuracy and stopping training when no further improvements are observed for a specified number of epochs. The best-performing model is saved automatically whenever a higher validation accuracy is achieved, ensuring that the final model selected is the most effective one.

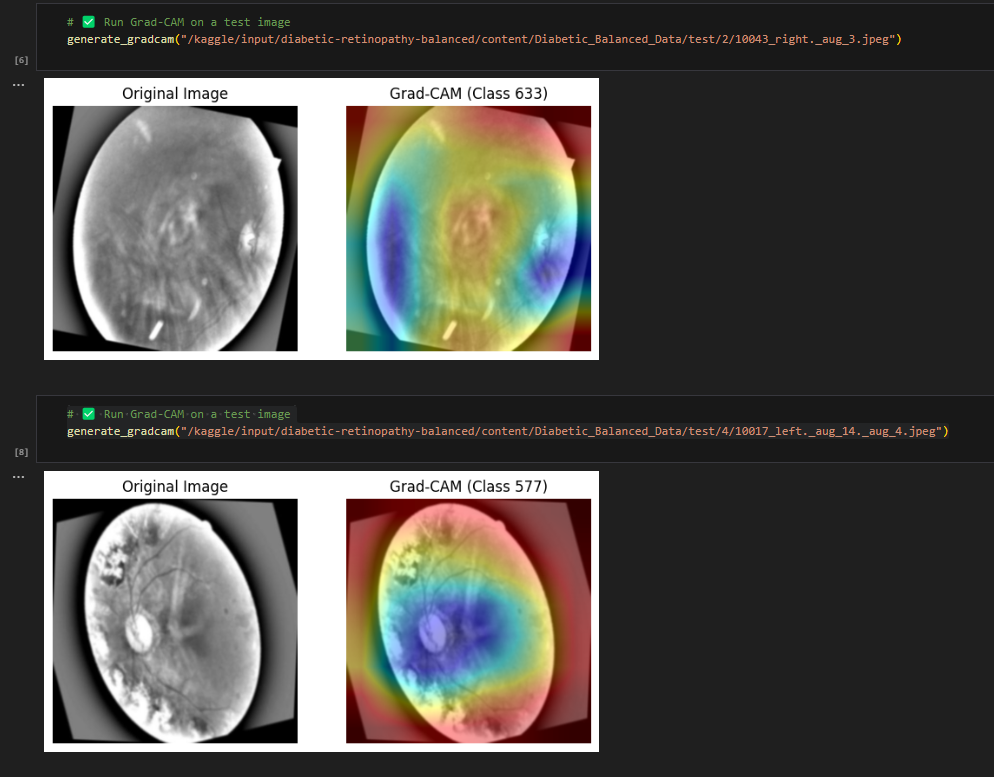
During testing, the trained model is evaluated on unseen data. The best model’s parameters are loaded, and predictions are made on the test set. Accuracy is determined by comparing predicted labels with ground truth values, providing a measure of the model’s performance in real-world scenarios. The final test accuracy of 54.23% reflects the model's ability to generalize to new data, highlighting areas for further refinement and potential improvements through techniques such as fine-tuning, additional data augmentation, or hyperparameter adjustments.

# Generating Grad-Cam

This implementation uses the InceptionV3 model to generate Grad-CAM visualizations, helping to interpret which regions of an image contribute most to a model's classification. The model is loaded with pre-trained weights and set to evaluation mode. A specific target layer, Mixed\_7c, is chosen to extract feature maps and gradients, which are essential for Grad-CAM calculations. Hooks are registered to capture both activations and gradients during the forward and backward passes.

For image preprocessing, the function resizes images to 299x299 pixels, normalizes them using standard ImageNet values, and converts them into tensors before sending them to the GPU if available. During inference, the model predicts the class of the input image, and the class index is determined based on the highest confidence score unless manually provided. The model is then backpropagated with respect to this class index to obtain gradients, which are averaged across spatial dimensions. These gradients are multiplied by the activation maps, emphasizing the most influential regions.

A heatmap is generated by averaging across the channels of the weighted activation maps and applying a ReLU operation to retain only positive influences. The heatmap is resized to match the original image dimensions and blended using OpenCV’s applyColorMap function, creating a visualization that highlights the key areas the model focused on while making its prediction. The original image and the Grad-CAM overlay are displayed side by side using Matplotlib, providing an intuitive way to understand how the model interprets the given image.



# Model Evaluation on Validation Data

The evaluate\_model function assesses the model's performance using a validation dataset. It processes images and labels in batches, runs inference, and stores predictions. If the model's output includes auxiliary logits (as seen in InceptionV3), only the primary logits are used. After obtaining predictions, precision, recall, and F1-score are calculated using sklearn.metrics. Additionally, a confusion matrix is generated and visualized using Seaborn.

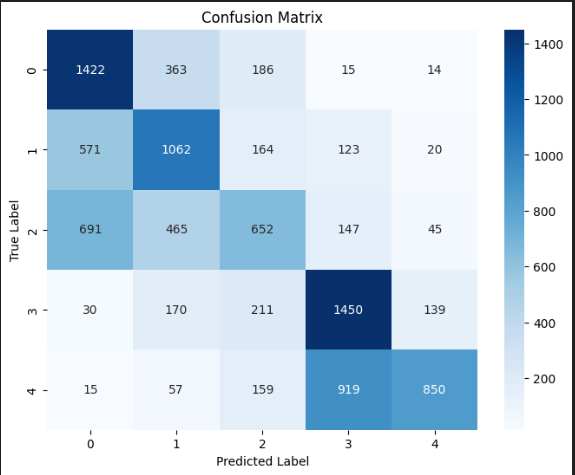
The evaluation results in:

Precision: 0.5680

Recall: 0.5469

F1-Score: 0.5378

A heatmap of the confusion matrix is plotted to visually assess the classification performance.



**Image Classification with the Trained Model**

The predict\_image function is designed to classify individual images. It first loads an image, applies resizing, normalization, and tensor conversion, and then runs it through the trained model. After inference, the predicted class index is obtained, mapped to the corresponding class name from the dataset, and displayed.

For an example input image, the model correctly predicts:

Predicted Class: 3 (3)