**Diabetic Retinopathy using advanced CNNs and Transformers**

**Abstract-**

* + **Project Objectives:**

The main goal of this project is to implement a strong image classification pipeline from the DDR (Diabetic Retinopathy) dataset, comprising retinal fundus images labelled with different grades of diabetic retinopathy. Our goal is to compare and assess the performance of different Convolutional Neural Network (CNN) models and Transformer-based architectures in extracting and learning intricate features from medical images. The project aims to determine the most appropriate models that can efficiently and accurately classify, thus adding to automated healthcare diagnostic systems.

* + M**ethodology:**

The project began with preprocessing the DDR (Diabetic Retinopathy) dataset through image resizing, normalization, and augmentation (such as flipping, rotation, and brightness adjustments) to improve model generalization. Multiple CNN architectures like basic CNN and Vgg19—were trained to leverage their hierarchical feature extraction capabilities. In parallel, Vision Transformers (ViT) were also implemented to explore their strength in capturing long-range dependencies without using convolutional layers. Accuracy was the primary evaluation metric, supported by precision and F1-score.

* + **Key findings:**

1. ViT outperforms CNN and VGG19 in classification accuracy, especially on higher severity classes.
2. VGG19 shows good generalization with transfer learning.
3. CNN is lightweight but may underperform on complex retinal patterns.
4. DR severity levels are harder to distinguish at moderate to severe stages—ViT handles this better due to its global attention mechanism.

**Step wise solution approach-**

Step 1: **Data Acquisition**

* Choose the DDR dataset by *MariaHerrerot* from Kaggle. Understand the dataset thoroughly, including its folder structure, image formats and resolutions, class distribution, etc. Additionally, check for class imbalance, if any.

Step 2: **Data Preprocessing in *batches***

* Resize all images to a fixed size (e.g., 224×224 or 256×256).
* Normalize pixel values (0–255 → 0–1).
* Save in batches

Step 3: **Train-Test Split and Dataset Organization**

* To prepare the data for training, we divided the DDR dataset into training, validation, and test sets using a stratified sampling approach to ensure class balance across splits.
* We applied a stratified train-test split using scikit-learn by splitting the dataset as **80%** for training, **16%** for validation, **4%** for testing
* Each split was saved into separate directories, with images organized in subfolders named after their respective class labels (e.g., train/0/, val/2/, test/4/)

Step 4: **Data Augmentation (Training Set Only)**

* To improve the model's ability to generalize and reduce overfitting, we applied **data augmentation** to the training images. This step synthetically increases the dataset size by applying random transformations to the original images, helping the model learn from diverse variations.
* Using Keras' ImageDataGenerator, the following transformations were configured:
  1. **Rotation**: Random rotation up to ±15 degrees
  2. **Width Shift**: Horizontal translation by 10% of the image width
  3. **Height Shift**: Vertical translation by 10% of the image height
  4. **Zoom**: Random zoom within a 20% range
  5. **Horizontal Flip**: Random horizontal flipping
  6. **Fill Mode**: Pixels introduced during transformations are filled using the nearest pixel values
* We applied only to the training set (/kaggle/working/images/train). The validation and test sets remain untouched to ensure fair and consistent evaluation.

Step 5: **Merging Original and Augmented Training Sets**

* To build a richer and more diverse training dataset, we combined the original and augmented images into a single folder structure (final\_train). This involved copying images from both the original (/images/train) and augmented (/images\_augmented/train) directories into the unified training directory, with subfolders organized by class label.

Step 6: **Build and Train Models**

**A. Custom CNN Model**

* Design a sequential model:
  + Convolution → ReLU → MaxPooling
  + Dropout for regularization
  + Dense layers → Softmax output
* Compile with:
  + Loss: categorical\_crossentropy
  + Optimizer: Adam
  + Metrics: accuracy
* Train using training and validation sets.

**B. VGG19 (Transfer Learning)**

* Load pretrained VGG19 (without top layers).
* Freeze base layers initially.
* Add custom classifier (dense layers).
* Fine-tune top layers (optional, after initial training).
* Use the same compile/train settings as CNN.

**C. Vision Transformer (ViT)**

* Load pretrained ViT model (e.g., vit-base-patch16-224-in21k).
* Tokenize images into patches.
* Fine-tune on your dataset:
  + Add classification head if required.
  + Use higher compute power (GPU recommended).
* Use transformers library (transformers, timm).

**Step 7: Model Evaluation and Comparitive analysis**

* Evaluate all models on the **test set**.
* Compute metrics:
  + Accuracy
  + Precision, Recall, F1-score (macro and class-wise)
  + Confusion Matrix
  + ROC-AUC (if possible)
* Plot training/validation curves for each model (loss, accuracy).
* Class-wise accuracy.

**Reference:**

[1] P. Sharma, V. Agarwal, S. Garg, “Advanced Deep Learning Approach for Diabetic Retinopathy Detection using ResNet‑50 and Explainability Techniques,” Proc. IEEE Conf., 2023.

[2] M. Chetoui et al., “Explainable Diabetic Retinopathy Using EfficientNet,” IEEE J. Biomed. Health Informatics, 2022.

[3] A. Awais et al., “Prospective Study on DR Detection Based on Modified CNN,” Sindh Inst. Ophthalmol., 2021.

[4] O. Ronneberger, P. Fischer, T. Brox, “U‑Net: Convolutional Networks for Biomedical Image Segmentation,” MICCAI, 2015.

[5] S. Lundberg, S.-I. Lee, “A Unified Approach to Interpreting Model Predictions,” NeurIPS, 2017

Team:

1. Parnavi Sharma 2. Shivanshu Garg