Train and Deploy a CNN Model Using TensorFlow Serving

Objective:
Build a CNN model using distributed training to detect diabetic retinopathy and deploy it using TensorFlow Serving.

Problem Statement:
You are a Computer Vision Engineer at health.ai. Your company is developing a deep learning application to automate the detection of diabetic retinopathy. The company is collecting high-resolution retina image data from various clinical partners. However, the dataset is expected to be enormous and cannot be stored on a central system. You have been assigned the task of building a proof of concept using the Kaggle retinopathy dataset. Your goal is to train a CNN model with the Mirrored Strategy and deploy it with TensorFlow Serving.

Dataset Details:
The dataset comprises a large collection of high-resolution retina images captured under various imaging conditions. For each subject, both left and right fields are provided. The images are labeled with a subject ID and designated as either left or right. A clinician has rated the presence of diabetic retinopathy in each image on a scale of 0 to 4. Like any real-world dataset, there may be noise in both the images and labels. The images might contain artifacts, be out of focus, underexposed, or overexposed.

Note: Please download the dataset **Sample** from the Reference Materials section.

Please follow the steps below:

1. Download the dataset and preprocess it to correct for noise, underexposure, and overexposure.

2. Augment the dataset by applying various image transformations and then split it into training and test sets.

3. Define the distributed training strategy, such as the Mirrored Strategy, to train the model using multiple devices or machines.

4. Specify the number of shared instances for the distributed training.

5. Design a CNN architecture that can effectively extract features from the input data.

6. Set parameters such as the loss function, optimizer, number of epochs, learning rate, and evaluation metric for training the model.

7. Implement checkpoints to save the model's weights during training, allowing for easy recovery in case of interruptions.

8. Train the model using the defined parameters until you achieve an accuracy of at least 80%.

9. Save the trained model for future use.

10. Deploy the saved model using TensorFlow Serving to make it available for inference and prediction tasks.