**Week 1**

**Problem statement**: Develop a CNN-based model to detect and classify plant diseases from leaf images of crops like apple, cherry, grape, and corn. The model should identify healthy and diseased leaves and predict the disease type, supporting precision agriculture through early detection and better disease management.

**Aim**: Design and implement a CNN model to classify plant leaf images into healthy or diseased categories, aiding early diagnosis and improving crop management practices.

**Pipeline:**

1. **Data Collection and Loading**: The dataset used in this project consists of images of plant leaves that belong to various crop types, including both healthy and diseased specimens. These images are crucial for training a deep learning model capable of detecting and classifying plant diseases with high accuracy. To ensure proper training and evaluation, the dataset is well-organized into three main subsets: train, test, and validation. The training set is used to teach the model, the validation set is used to tune model parameters and avoid overfitting, and the test set is used to evaluate the final performance of the model on unseen data. All these subsets are placed inside a single folder, which is then compressed into a ZIP file for ease of storage and transfer. This ZIP file is uploaded to Google Drive, which acts as cloud storage for accessing the data from any location. In Google Colab, which provides a cloud-based development environment with free GPU access, the Google Drive is mounted using the drive.mount() function from the google.colab library. Once mounted, the ZIP file is located within the drive, and standard shell commands are used to unzip and extract its contents. The images are then ready for preprocessing and model training.
2. **Upload**: To train a CNN model for plant disease detection, the dataset containing images of healthy and diseased leaves needs to be uploaded and accessed in the development environment. For this project, Google Drive is used as a storage solution due to its seamless integration with Google Colab, which provides free computational resources, including GPU support for deep learning tasks. The dataset is first structured into three folders:

(i) train/ – for training the model

(ii) test/ – for final evaluation

(iii) valid/ – for tuning the model during training

These folders are collectively zipped into a single .zip file to make uploading easier and reduce transfer time. This .zip file is then uploaded to your personal Google Drive.

1. **Image processing**: Image processing is a crucial step in preparing raw image data for training a deep learning model. It helps improve the quality of input data and ensures consistency in image size, format, and pixel intensity, which enhances model performance and generalization. For this project, the dataset contains images of plant leaves that vary in resolution, lighting conditions, and orientation. To standardize the input and improve the learning process, the following image processing steps are applied:

(i) Resizing: All images are resized to a fixed dimension (e.g., 224x224 or 128x128 pixels) to ensure uniform input shape for the CNN model.

(ii) Normalization: Pixel values are scaled to a range of [0, 1] by dividing each value by 255. This helps the model converge faster and improves numerical stability during training.

(iii) Data Augmentation: To artificially increase the dataset size and reduce overfitting, random transformations are applied, such as:

(i) Horizontal and vertical flipping

(ii) Rotation (e.g., ±15 degrees)

(iii) Zooming in/out

(iv) Brightness and contrast adjustments

These augmentations generate new variations of the original images, making the model more robust to unseen data. Image processing is typically done using libraries like TensorFlow/Keras, OpenCV, or PIL.

1. **CNN Model**: A Convolutional Neural Network (CNN) is a deep learning architecture commonly used for image classification tasks. In this project, a CNN model is built to automatically detect and classify plant diseases from leaf images. The model takes preprocessed images as input and passes them through several layers to extract features and make predictions. The structure of the CNN model typically includes:

(i) Convolutional Layers: These layers apply filters (kernels) to the input image to extract important features such as edges, textures, and patterns. Each filter captures different spatial features from the image.

(ii) Activation Function (ReLU): The ReLU (Rectified Linear Unit) function introduces non-linearity into the model, helping it learn complex patterns.

(iii) Pooling Layers: Pooling (usually Max Pooling) reduces the spatial dimensions of feature maps, lowering computation and helping the model become invariant to small translations.

(iv) Flatten Layer: Converts the 2D feature maps into a 1D vector to be passed into fully connected layers.

(v) Fully Connected (Dense) Layers: These layers perform the final classification by mapping features to output classes (e.g., healthy, apple\_scab, corn\_rust, etc.).

(vi) Output Layer: Uses a softmax activation function to produce probability scores for each class.

The model is compiled using a loss function like categorical\_crossentropy and optimized using algorithms like Adam. It is then trained on the training dataset and validated on unseen data to evaluate performance.

1. **Training and Evaluation**: Once the CNN model is built, the next step is to train it using the labeled dataset and then evaluate its performance. Training allows the model to learn patterns and features from the input images, while evaluation tests its ability to generalize to unseen data.

**Training Phase**

(i) Input: The model takes preprocessed and augmented images from the train folder, each labeled with its respective class.

(ii) Process: During training, the model adjusts its weights using backpropagation and the chosen optimizer (e.g., Adam).

(iii) Epochs and Batches: The training is conducted over several epochs (complete passes through the dataset), and the data is divided into batches for efficient processing.

(iv) Loss Function: Categorical Crossentropy is used to calculate the error between the predicted and actual labels.

(v) Validation: After each epoch, the model is evaluated on the validation dataset to monitor overfitting and fine-tune hyperparameters like learning rate or dropout.

**Evaluation Phase**

(i) Testing: Once training is complete, the model is tested on the test dataset, which contains unseen data.

(ii) Metrics: Performance is evaluated using metrics like accuracy, precision, recall, and (iii) F1-score: A confusion matrix can also be used to visualize prediction results across all classes.

This process ensures the model is reliable and performs well in real-world scenarios.