**Ex 3.1: Implement CNN for image classification**

Objective:

To perform complex image **binary** classification and **multiclass** classification using CNN. Perform hyperparameter tuning and compare the results. Perform two classifications and perform all the following tasks for both the classifications.

Task 1: Dataset Selection

Step 1.1: Visit Kaggle and select a dataset suitable for image classification.

Step 1.2: Download the dataset and split it into training, validation, and test sets.

Task 2: Data Preprocessing

Step 2.1: Load the dataset.

Step 2.2: Normalize the images (e.g., rescale pixel values to a range of [0,1] or [-1, 1]).

Step 2.3: Convert categorical labels to one-hot encoding if necessary.

Task 3: Implement CNN Without Data Augmentation

Step 3.1: Design a basic CNN architecture. Start with a simple architecture like:

Input layer (e.g., 32x32x3 for CIFAR-10)

Convolutional layers (e.g., 2-3 Conv layers with ReLU activation)

MaxPooling layers

Fully connected layers

Output layer (softmax for classification)

Step 3.2: Compile the model using an appropriate optimizer (e.g., Adam), loss function (e.g., categorical crossentropy), and metrics (e.g., accuracy).

Step 3.3: Train the CNN on the original dataset (without augmentation) for a specified number of epochs.

Step 3.4: Evaluate the model on the validation and test datasets. Record the accuracy, precision, recall, F1-score, and confusion matrix.

Task 4: Data Augmentation

Step 4.1: Implement data augmentation techniques such as:

Horizontal and vertical flips

Random rotations and translations

Zooming, shearing, and brightness adjustments

Step 4.2: Use TensorFlow/Keras ImageDataGenerator or PyTorch transforms for augmenting the data.

Task 5: Implement CNN With Data Augmentation

Step 5.1: Retrain the CNN model using the augmented data.

Step 5.2: Use the same architecture as in Task 3 but apply the data augmentation during training.

Step 5.3: Evaluate the performance of the CNN on the validation and test datasets, and record the performance metrics.

Task 6: Experiment with Different CNN Architectures and Hyperparameters

Step 6.1: Modify the CNN architecture by experimenting with:

Adding more convolutional layers

Using dropout layers to prevent overfitting

Varying the number of filters in Conv layers

Adjusting the kernel size and pooling layers

Step 6.2: Experiment with different optimizers (e.g., SGD, RMSprop) and learning rates.

Step 6.3: Train the modified models and evaluate them using the same metrics as before.

Task 7: Comparison and Reporting

Step 7.1: Compare the performance of the models:

CNN without augmentation vs. CNN with augmentation

Different CNN architectures and hyperparameters

Step 7.2: Create visualizations for:

Training and validation accuracy/loss curves

Confusion matrices

Performance metrics (e.g., accuracy, F1-score) in a tabular format

Step 7.3: Write a report summarizing your findings:

Which configuration yielded the best performance?

What impact did data augmentation have on the model’s performance?

How did different architectures and hyperparameters affect the results?