**Assignment No: - 3**

**Image Classification using CNNs**

**Problem Statement:**

Implement Image classification using convolutional neural networks (CNNs) for multiclass

classification.

**Objective:**

* Understand the architecture and working of Convolutional Neural Networks.
* Learn how to preprocess image data for training CNNs.
* Implement a CNN model using Keras and TensorFlow for multiclass classification.
* Evaluate model performance using validation data.
* Visualize training accuracy and loss over epochs.

**S/W Packages and H/W Apparatus:**

* **Operating System**: Windows/Linux/MacOS
* **Kernel**: Python 3.x
* **Tools**: Jupyter Notebook, Anaconda, or Google Colab
* **Hardware**: CPU with minimum 4GB RAM (optional GPU for faster processing)

**Libraries and Packages:**

* TensorFlow
* Keras
* NumPy
* Matplotlib

**Theory:**

**Convolutional Neural Networks (CNNs)** are deep learning algorithms primarily used for image-related tasks, including classification. CNNs learn to detect and understand different features in images, such as edges, shapes, and textures, through convolutional and pooling layers. CNN models are widely used in tasks such as object detection, recognition, and segmentation.

**Structure of CNNs:**

1. **Input Layer**: Receives input images.
2. **Convolutional Layers**: Extracts features by applying filters.
3. **Pooling Layers**: Reduces spatial dimensions, retains important information.
4. **Fully Connected Layers**: Connects all neurons in one layer to every neuron in the next.
5. **Output Layer**: Produces class probabilities.

**Activation Functions**:

* **ReLU (Rectified Linear Unit)**: Adds non-linearity to the network, preventing the model from being limited to linear transformations.
* **Softmax**: Typically used in the output layer for multiclass classification.

**Backpropagation**: Used to update the weights during training by minimizing the loss using gradient descent.

**Methodology:**

1. **Data Acquisition**:
   * Load the CIFAR-10 dataset, which includes 60,000 images from 10 categories (e.g., airplane, automobile, bird, etc.).
2. **Data Preparation**:
   * Normalize image pixel values between 0 and 1 for faster training.
   * Optionally augment the dataset with transformations such as flips or rotations to increase data diversity.
3. **Model Architecture**:
   * Use the Sequential API from Keras to define the model.
   * Include the following layers:
     + First convolutional layer with 32 filters (3x3), followed by max-pooling (2x2).
     + Additional convolutional layers with increased filters (64, 128) and max-pooling.
     + Flatten layer to transition from 2D to 1D data.
     + Dense layers with ReLU activation.
     + Output layer with 10 units (one for each class) and Softmax activation for multiclass classification.
4. **Model Compilation**:
   * Compile using the **Adam** optimizer.
   * Loss function: **Sparse Categorical Crossentropy**.
   * Metrics: **Accuracy**.
5. **Model Training**:
   * Train the CNN for 10 epochs using a training-validation split.
   * Monitor training and validation accuracy during each epoch.
6. **Model Evaluation**:
   * After training, evaluate the model on the test dataset.
   * Print the test accuracy.
7. **Loss Visualization**:
   * Plot training and validation accuracy/loss over epochs to visualize model performance.

**Advantages:**

* **Automatic Feature Extraction**: CNNs can automatically learn and detect features like edges, textures, and shapes, eliminating the need for manual feature engineering.
* **Reduced Parameters**: Due to shared weights in convolutional layers, CNNs require fewer parameters compared to fully connected networks.
* **Robustness**: CNNs are translation-invariant and can generalize well even when the image is shifted or rotated.

**Limitations:**

* **Data Dependency**: CNNs need large datasets to perform well.
* **Computational Power**: Training CNNs on large datasets can be computationally expensive.
* **Overfitting**: Without proper regularization, CNNs may overfit the training data, especially with small datasets.

**Applications:**

* **Image Classification**: CNNs are used to classify images in various domains like medical imaging, object detection, etc.
* **Object Detection**: CNNs can be used to detect and localize objects within images.
* **Autonomous Driving**: Used for recognizing traffic signs, pedestrians, and other vehicles.

**Working / Algorithm:**

**Step 1**: Load Dataset

* Load the CIFAR-10 dataset using TensorFlow/Keras API.

**Step 2**: Preprocess Data

* Normalize pixel values by dividing by 255 (range 0–1).

**Step 3**: Visualize Data

* Display a sample of the dataset using Matplotlib.

**Step 4**: Define CNN Architecture

* Define the CNN using Keras with multiple convolutional, pooling, and dense layers.

**Step 5**: Compile the Model

* Use Adam optimizer, Sparse Categorical Crossentropy as loss, and accuracy as the performance metric.

**Step 6**: Train the Model

* Train for 10 epochs, validating after each epoch.

**Step 7**: Evaluate the Model

* Calculate and print the test accuracy.

**Step 8**: Visualize Training History

* Plot training and validation accuracy/loss over epochs.

**Step 9**: Print Test Accuracy

* Evaluate the test accuracy to assess the model’s performance.

**Diagram:**



**Conclusion:**

Convolutional Neural Networks (CNNs) offer an effective solution for image classification tasks, providing automatic feature extraction and excellent generalization performance. With the right amount of data and computational resources, CNNs can achieve high accuracy across various applications. However, it's important to carefully handle data preprocessing, model architecture design, and evaluation to avoid overfitting and ensure robust performance.