**Assignment No: - 3**

**Image Classification using CNNs**

**Problem Statement:**

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

classification.

**Objective:**

* To understand the architecture and working of Convolutional Neural Networks.
* To learn how to preprocess image data for training CNNs.
* To implement a CNN model using Keras and TensorFlow for multiclass classification.
* To evaluate model performance using validation data.
* To visualize training accuracy and loss over epochs.

**S/W Packages and H/W apparatus used:**

* **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 used:**

* **TensorFlow**
* **Keras**
* **NumPy**
* **Matplotlib**

**Theory:**

A Convolutional Neural Network (CNN) is a deep learning algorithm primarily used for processing structured grid data, such as images. CNNs automatically detect features and patterns in images, making them highly effective for image classification tasks.

**Structure:**

Input Layer: Receives input images.

Convolutional Layers: Apply convolution operations to extract features from images. These layers contain filters that learn to recognize patterns.

Pooling Layers: Reduce the spatial dimensions of feature maps, retaining the most essential information while reducing computation.

Fully Connected Layers: After flattening the pooled feature maps, these layers connect every neuron in one layer to every neuron in the next layer, leading to the output layer.

Output Layer: Produces class probabilities for the input images.

**Activation Functions:**

Common activation functions used in CNNs include ReLU (Rectified Linear Unit) and SoftMax, which introduce non-linearity into the model and help in classifying multiple classes.

**Backpropagation:**

The backpropagation algorithm is employed for training CNNs, where gradients are calculated and used to update the weights of the network to minimize the loss function.

**Methodology:**

1. **Data Acquisition:**

* Load the CIFAR-10 dataset, which contains 60,000 images across 10 classes.

1. **Data Preparation:**

* Normalize pixel values to a range between 0 and 1 to facilitate faster convergence.

1. **Model Architecture:**

* Create a sequential model using Keras.
* Add convolutional and pooling layers:
  + First convolutional layer with 32 filters and a kernel size of 3x3, followed by max pooling.
  + Second convolutional layer with 64 filters and another max pooling layer.
  + Additional convolutional layers as needed.
* Flatten the output and add dense layers:
  + Fully connected layer with 64 units and ReLU activation.
  + Output layer with 10 units for classification.

1. **Model Compilation:**

* Compile the model using the Adam optimizer and Sparse Categorical Crossentropy as the loss function.

1. **Model Training:**

* Fit the model on the training dataset while validating on the test dataset.
* Track accuracy and loss over epochs.

1. **Model Evaluation:**

* Evaluate the model on the test dataset to measure performance.

1. **Loss Visualization:**

* Plot training and validation accuracy and loss over epochs to assess model performance.

**Advantages:**

* **Feature Extraction:** CNNs automatically learn to extract relevant features from images, reducing the need for manual feature engineering.
* **Translation Invariance:** They are robust to shifts and distortions in images, enabling better generalization to new data.
* **Reduced Parameters:** The use of convolutional layers decreases the number of parameters compared to fully connected networks, making training faster and less prone to overfitting.
* **Hierarchical Feature Learning:** CNNs learn features at multiple levels of abstraction, from simple edges to complex shapes.

**Limitations:**

* **Data Requirements:** CNNs typically require large amounts of labelled data for effective training.
* **Computational Cost:** Training deep CNNs can be computationally expensive and time-consuming.
* **Overfitting Risk:** If the model is too complex, it may overfit the training data, leading to poor performance on unseen data.
* **Hyperparameter Sensitivity:** The model's performance can be highly sensitive to the choice of hyperparameters, requiring careful tuning.

**Applications:**

* **Image Classification:** CNNs are widely used for classifying images in various domains, including medical imaging, object detection, and facial recognition.
* **Image Segmentation:** They are employed in tasks where the goal is to classify each pixel in an image, such as in autonomous driving.
* **Video Analysis:** CNNs can be applied to video data for actions recognition and tracking.

**Working / Algorithm:**

Step 1: Import required libraries (TensorFlow, Keras, NumPy, Matplotlib).  
Step 2: Load dataset (e.g., CIFAR-10 / MNIST).  
Step 3: Normalize images by dividing pixel values by 255.  
Step 4: Build CNN model:

* Conv2D → MaxPooling → Conv2D → MaxPooling → Flatten → Dense → Output.  
  Step 5: Compile model with optimizer (Adam) and loss (categorical\_crossentropy).  
  Step 6: Train model with training dataset (epochs ~10).  
  Step 7: Evaluate on test set and compute accuracy.  
  Step 8: Visualize accuracy/loss using Matplotlib plots.

**Diagram:**



**Conclusion:**

In conclusion, Convolutional Neural Networks (CNNs) are a powerful and efficient approach for image classification tasks, particularly for multiclass classification. By leveraging their ability to automatically extract features from images and learn hierarchical representations, CNNs have achieved state-of-the-art performance across various applications. However, practitioners should be mindful of the data requirements, computational costs, and the potential for overfitting. With proper management and optimization, CNNs can be effectively applied to solve complex image classification problems.