Name: Jiya Sharma
RollNo: 381056
Prn: 22311741

# Assignment 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.
2. Data Preparation:
- Normalize pixel values to a range between 0 and 1 to facilitate faster convergence.
3. 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.
4. Model Compilation:
- Compile the model using the Adam optimizer and Sparse Categorical Crossentropy as the loss function.
5. Model Training:
- Fit the model on the training dataset while validating on the test dataset.
- Track accuracy and loss over epochs.
6. Model Evaluation:
- Evaluate the model on the test dataset to measure performance.
7. 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: Load Dataset

The CIFAR-10 dataset, which contains 60,000 32x32 color images across 10 different classes, is loaded using the TensorFlow/Keras datasets API. The dataset is split into training and test sets, with 50,000 images for training and 10,000 images for testing.

Step 2: Preprocess Data

The images are normalized by dividing each pixel value by 255. This scales the pixel values between 0 and 1, improving the efficiency of model training.

Step 3: Visualize Data

A sample of 25 images from the training set is displayed using matplotlib. Each image is labeled with its corresponding class name (e.g., airplane, automobile, etc.).

Step 4: Define CNN Model Architecture

A Convolutional Neural Network (CNN) is created using the Sequential API in Keras:

First Convolutional Layer: 32 filters with a 3x3 kernel, ReLU activation, followed by max-pooling (2x2).

Second Convolutional Layer: 64 filters with a 3x3 kernel, ReLU activation, followed by max-pooling (2x2).

Third Convolutional Layer: 64 filters with a 3x3 kernel and ReLU activation.

Flatten Layer: Converts the 3D feature maps into 1D vectors.

Dense Layer: Fully connected layer with 64 units and ReLU activation.

Output Layer: Dense layer with 10 units (for 10 classes) and SoftMax activation for multiclass classification.

Step 5: Compile the Model

The model is compiled with the Adam optimizer, sparse categorical cross entropy loss (appropriate for multiclass classification), and accuracy as the performance metric.

Step 6: Train the Model

The model is trained for 10 epochs on the training data. During each epoch:

The model processes the training data in batches (with a batch size of 128).

The loss and accuracy are computed for both the training and validation (test) sets.

Validation data (test set) is used to monitor the model's performance after each epoch.

Step 7: Evaluate the Model

After training, the model is evaluated on the test data to compute the final test loss and accuracy.

Step 8: Visualize Training History

A plot is generated showing the model's accuracy and validation accuracy over the training epochs.

Step 9: Print Test Accuracy

- The final test accuracy is printed, indicating how well the model performs on unseen test data.

**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.