**Assignment No: 3**

**Image Classification using CNNs**

1. **Problem Statement**

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

1. **Objective**

* Understand the architecture and functioning of Convolutional Neural Networks.
* Preprocess image data for efficient CNN training.
* Build and implement a CNN model using Keras and TensorFlow for multiclass classification.
* Evaluate model performance on validation data.
* Visualize training accuracy and loss trends over epochs.

1. **Software and Hardware Requirements**

**Operating System:** Windows / Linux / MacOS  
**Programming Environment:** Python 3.x, Jupyter Notebook, Anaconda, or Google Colab  
**Hardware:** Minimum 4GB RAM (CPU); optional GPU for faster training

**Libraries:**

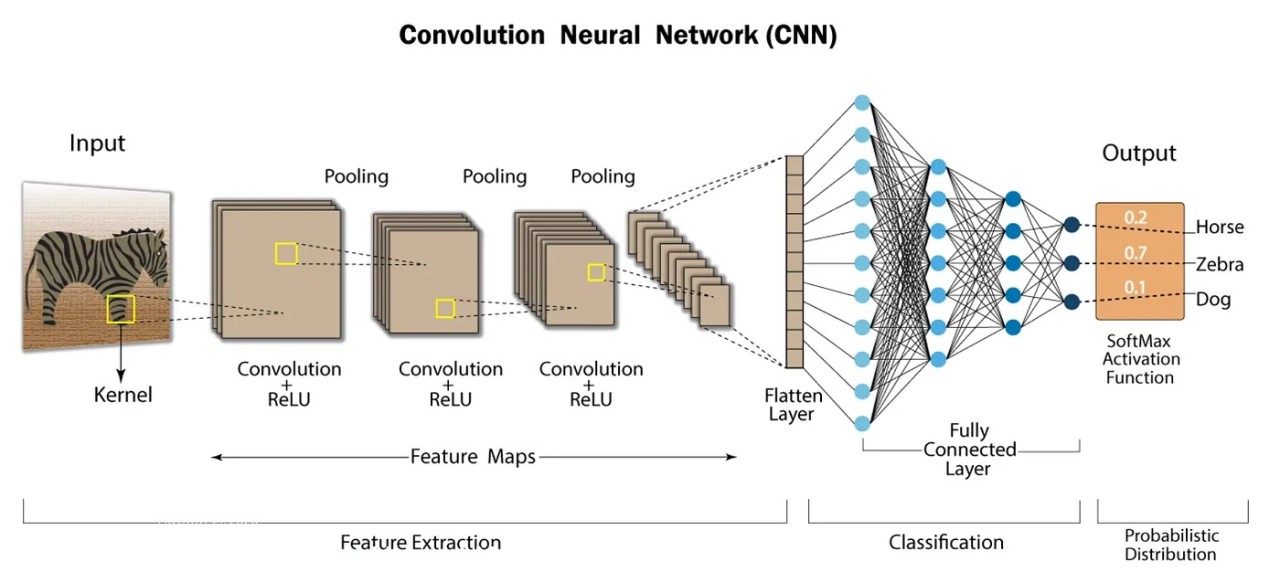
* TensorFlow
* Keras
* NumPy
* Matplotlib

1. **Theory**

**Definition:**  
Convolutional Neural Networks (CNNs) are deep learning models designed for processing structured grid data, such as images. CNNs automatically detect patterns and features, making them highly effective for image classification.

**Structure:**

1. **Input Layer:** Receives the images.
2. **Convolutional Layers:** Apply filters to extract feature maps and detect patterns.
3. **Pooling Layers:** Reduce spatial dimensions while preserving important features.
4. **Fully Connected Layers:** Flatten feature maps and connect neurons densely for decision-making.
5. **Output Layer:** Produces class probabilities.

**Fig. 1. Layers**

**Activation Functions:**

* **ReLU:** Introduces non-linearity and accelerates convergence.
* **SoftMax:** Converts outputs to class probabilities for multiclass classification.

**Backpropagation:**  
Used to update weights by calculating gradients and minimizing the loss function.

1. **Methodology**
2. **Data Acquisition:**
   * Load the CIFAR-10 dataset containing 60,000 32x32 color images across 10 classes.
3. **Data Preparation:**
   * Normalize pixel values to the range [0, 1] to improve training efficiency.
4. **Model Architecture:**
   * Build a Sequential CNN model in Keras:
     + **Conv Layer 1:** 32 filters, 3x3 kernel, ReLU activation → MaxPooling
     + **Conv Layer 2:** 64 filters, 3x3 kernel, ReLU activation → MaxPooling
     + **Conv Layer 3:** 64 filters, 3x3 kernel, ReLU activation
     + **Flatten Layer** → Dense Layer with 64 units, ReLU activation
     + **Output Layer:** 10 units (for 10 classes), SoftMax activation
5. **Model Compilation:**
   * Optimizer: Adam
   * Loss: Sparse Categorical Crossentropy
   * Metric: Accuracy
6. **Model Training:**
   * Train the model for 10 epochs with a batch size of 128.
   * Use test set for validation to monitor performance during training.
7. **Model Evaluation:**
   * Evaluate the trained model on the test data to obtain final loss and accuracy.
8. **Visualization:**
   * Plot training and validation accuracy and loss over epochs to analyze model performance.
9. **Advantages**

* **Automatic Feature Extraction:** CNNs learn relevant features without manual engineering.
* **Translation Invariance:** Robust to shifts, distortions, and minor changes in images.
* **Reduced Parameters:** Convolutional layers decrease the number of parameters compared to fully connected networks.
* **Hierarchical Learning:** Learn features at multiple levels, from edges to complex shapes.

1. **Limitations**

* **Data Requirements:** CNNs need large amounts of labeled data for optimal training.
* **Computational Cost:** Training deep CNNs is resource-intensive.
* **Overfitting Risk:** Complex models may overfit small datasets.
* **Hyperparameter Sensitivity:** Performance depends on careful tuning of layers, learning rate, and other hyperparameters.

1. **Applications**

* **Image Classification:** Object recognition, medical imaging, and facial recognition.
* **Image Segmentation:** Pixel-level classification for applications like autonomous driving.
* **Video Analysis:** Action recognition, motion tracking, and activity detection.

1. **Working / Algorithm**

**Step 1: Load Dataset**

* Load CIFAR-10 dataset (50,000 training images, 10,000 test images) using Keras datasets API.

**Step 2: Preprocess Data**

* Normalize images by dividing pixel values by 255 to scale between 0 and 1.

**Step 3: Visualize Data**

* Display a sample of 25 images with corresponding class labels.

**Step 4: Define CNN Model**

* Create a Sequential CNN with three convolutional layers, max pooling, flattening, a dense layer, and a SoftMax output layer for 10 classes.

**Step 5: Compile Model**

* Use Adam optimizer, sparse categorical crossentropy loss, and accuracy as metric.

**Step 6: Train Model**

* Train for 10 epochs, monitor training and validation accuracy and loss.

**Step 7: Evaluate Model**

* Compute final test accuracy and loss on unseen test data.

**Step 8: Visualize Training History**

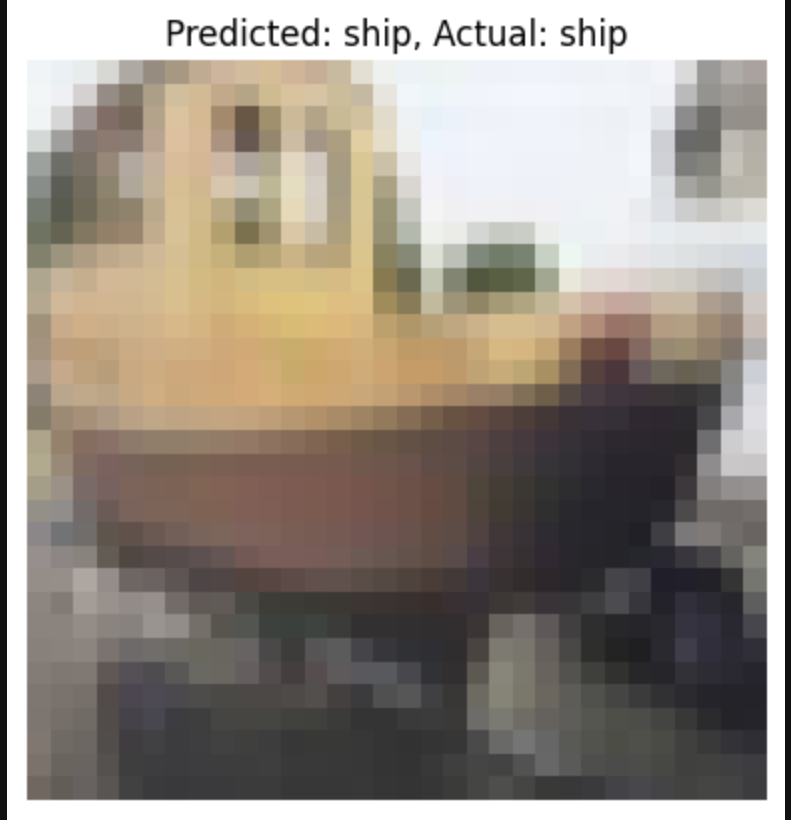
* Plot accuracy and validation accuracy to assess model performance.

**Step 9: Print Test Accuracy**

* Display the final test accuracy for evaluation on unseen data.

1. **Conclusion**

Convolutional Neural Networks (CNNs) are highly effective for multiclass image classification, thanks to their ability to automatically extract hierarchical features from images. CNNs reduce the need for manual feature engineering while maintaining robustness to translations and distortions. Despite challenges such as high computational cost, data requirements, and overfitting, CNNs provide state-of-the-art performance across various applications, from image classification to video analysis and autonomous systems. Proper training, tuning, and data management ensure that CNNs can efficiently handle complex visual recognition tasks.

1. **Output**

**Fig. 1. Output**