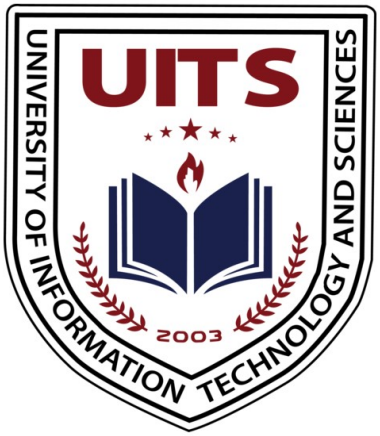


Image Classification using CNN (Convolutional Neural Network)

Course code: **CSE 432**
Course name: **Machine Learning Lab**

Submitted by,

Musber Ahmed Sadman

ID- 2215151038

Sec-7A2

Semester: Spring

Batch: 51

Submitted to:

Md. Yousuf Ali, Lecturer, CSE, UITS

Date of Submission: 07-07-2025

1. Introduction:

One of the most fundamental task in computer vision that involves the categorization images over predefined classes is named Image Classification. In this project, we are going to implement and evaluate a CNN for image classification. We are using the dataset named CIFAR-10 dataset. CNNs are a class of deep neural networks. It is mainly designed for processing visual data and automatically learning hierarchical features from raw pixel data. CNN are inspired by the human visual system. More specifically based on how the brain processes visual information. Unlike traditional neural networks where every neuron in one layer connects to every neuron in the next (fully connected), CNNs leverage a hierarchical structure to learn features directly from the raw input data

2. Objectives:

The primary objective of this project is to:

- Load and Preprocess an image.
- Design and build simple CNN model.
- Train the CNN model on the dataset.
- Evaluate the model's performance.
- Visualize the training progress to understand model behavior.

3. Dataset Description (CIFAR-10):

The CIFAR-10 is a broadly used dataset in the field of machine learning and computer vision. It is a subset of the 80 million tiny images dataset and was collected by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton.

Key Characteristics:

- Total Images: 60,000 color images.
- Image Dimensions: Each image is 32x32 pixels with 3 color channels (RGB).
- Classes: The dataset is divided into 10 distinct classes.
- Distribution: There are 6,000 images per class, ensuring a balanced dataset.
- Splits: It is pre-divided into a training set of 50,000 images and a test set of 10,000 images.

The 10 classes are:

- airplane
- automobile
- bird
- cat
- deer
- dog
- frog
- horse
- ship
- truck

Dataset Link: The CIFAR-10 dataset is conveniently available directly through TensorFlow's Keras API, eliminating the need for manual download. It can be loaded using tf.keras.datasets.cifar10.load_data().

Data Preprocessing: Before feeding the images to the neural network, pixel values (which range from 0 to 255) are normalized to a scale of 0 to 1. This normalization helps in faster convergence of the model during training and improves overall performance.

Simple Data Visualization: To understand the nature of the images in the CIFAR-10 dataset, a sample of 25 training images is visualized below. Each image is displayed with its corresponding class label.

4. Code Screenshots:

Importing the libraries:

```
[1]  import tensorflow as tf
     from tensorflow.keras import datasets, layers, models
     import matplotlib.pyplot as plt
     import numpy as np
```

Loading and Preparing the CIFAR-10 Dataset:

```
[2]  #Load and Prepare the CIFAR-10 Dataset
     print("Loading CIFAR-10 dataset...")
     (train_images, train_labels), (test_images, test_labels) = datasets.cifar10.load_data()
```

Normalizing and preparing for visualization:

```
[3]  #Normalize pixel values to be between 0 and 1
     train_images, test_images = train_images / 255.0, test_images / 255.0

     #Define class names for visualization
     class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',
                    'dog', 'frog', 'horse', 'ship', 'truck']

     print(f"Train images shape: {train_images.shape}")
     print(f"Train labels shape: {train_labels.shape}")
     print(f"Test images shape: {test_images.shape}")
     print(f"Test labels shape: {test_labels.shape}")
     print(f"Number of classes: {len(class_names)}")
```

Visualizing the sample data:

```
[4]  #Visualize Sample Data
     plt.figure(figsize=(10,10))
     for i in range(25):
         plt.subplot(5,5,i+1)
         plt.xticks([])
         plt.yticks([])
         plt.grid(False)
         plt.imshow(train_images[i])
         # The labels are arrays, so we need to flatten them for class_names indexing
         plt.xlabel(class_names[train_labels[i][0]])
     plt.suptitle("Sample CIFAR-10 Images", y=1.02, fontsize=16)
     plt.show()
```

Building CNN Model and Flattening the output:

```
[5]  #Build the Convolutional Neural Network (CNN) Model
     model = models.Sequential()
```

```
[6]  #Convolutional Block 1
     model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
     model.add(layers.MaxPooling2D((2, 2)))

     #Convolutional Block 2
     model.add(layers.Conv2D(64, (3, 3), activation='relu'))
     model.add(layers.MaxPooling2D((2, 2)))

     #Convolutional Block 3
     model.add(layers.Conv2D(64, (3, 3), activation='relu'))

     #Flatten the output to feed into Dense layers
     model.add(layers.Flatten())
```

Dense Layers:

```
[7]  #Dense layers
     model.add(layers.Dense(64, activation='relu'))
     model.add(layers.Dense(10, activation='softmax')) #Because there are 10 classes
```

Compile the model and print the model summary:

```
[8]  #Compile the Model
     model.compile(optimizer='adam',
                   loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
                   metrics=['accuracy'])

     #Print model summary
     model.summary()
```

Train the model and then evaluating it:

```
[9]  #Train the Model
     print("\nTraining the model...")
     history = model.fit(train_images, train_labels, epochs=10,
                         validation_data=(test_images, test_labels))

     #Evaluate the Model
     print("\nEvaluating the model...")
     test_loss, test_acc = model.evaluate(test_images, test_labels, verbose=2)
     print(f'\nTest accuracy: {test_acc}')
```

Showing the plot of training and testing accuracy of the CNN Model:

```
[10] #Plot Training History
     plt.figure(figsize=(12, 5))

     # Plot accuracy
     plt.subplot(1, 2, 1)
     plt.plot(history.history['accuracy'], label='Training Accuracy')
     # In this specific CIFAR-10 setup, 'val_accuracy' is the accuracy on the test set per epoch
     plt.plot(history.history['val_accuracy'], label = 'Test Accuracy (per epoch)')
     # Plot the final test accuracy as a horizontal line
     plt.axhline(y=test_acc, color='r', linestyle='--', label=f'Final Test Accuracy ({test_acc:.4f})'
     plt.xlabel('Epoch')
     plt.ylabel('Accuracy')
     plt.ylim([0, 1])
     plt.legend(loc='lower right')
     plt.title("Training and Test Accuracy")

     plt.show()
```

5. Outputs:

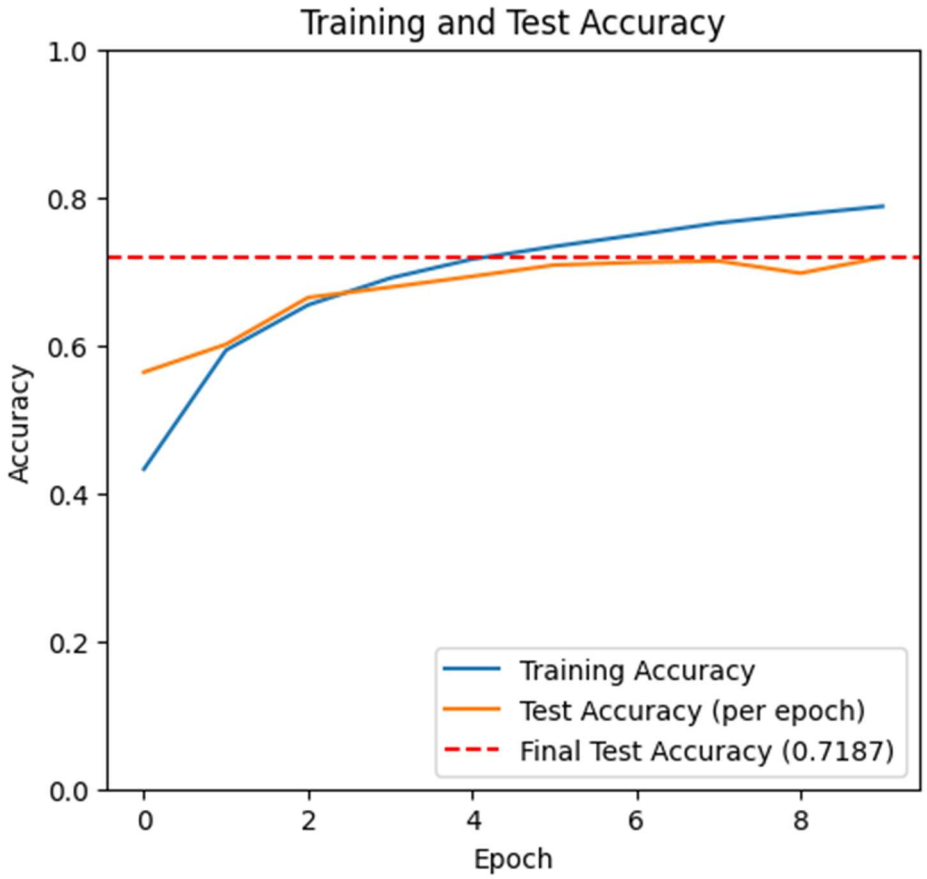
After evaluating the model, the following results show up:

The test accuracy is shown:

```
Evaluating the model...
313/313 - 1s - 2ms/step - accuracy: 0.7187 - loss: 0.8670

Test accuracy: 0.7186999917030334
```

The plot of Training Vs Test Accuracy:



6. Methodology:

To tackle the image classification task, we used a Convolutional Neural Network (CNN) — a type of deep learning model that's especially effective for working with image data. CNNs are powerful because they can automatically learn and identify patterns, shapes, and structures in images by using layers that mimic the way we visually process the world.

Here's a descriptive discussion of how the model is structured:

- Convolutional Layers (Conv2D): These are the core building blocks of the network. They use small filters to scan over the image and detect features like edges, textures, or colors. As the image moves through these layers, the network learns more complex and abstract patterns. We use the ReLU (Rectified Linear Unit) activation function here to add non-linearity, helping the model handle complex data.
- Max Pooling Layers (MaxPooling2D): After detecting features, these layers reduce the size of the data. Think of it as zooming out — we get a simpler version of the feature map that still holds the important information. This helps the model run faster and become more resilient to small changes in the image.
- Flatten Layer: Once the image has passed through all the convolution and pooling layers, we flatten the data into a single vector. This step gets it ready for the final classification part of the model.
- Dense Layers: These fully connected layers are where the actual decision-making happens. Each neuron is connected to all neurons in the previous layer, allowing the model to learn overall patterns and relationships in the data.
- Output Layer: Finally, the last dense layer has 10 neurons, each representing one of the 10 categories in the CIFAR-10 dataset. We use a softmax activation function here, which turns the output into probabilities. This helps the model decide which class the image most likely belongs to.

Model Design:

The specific model that we have designed in this project is a sequential model with three convolutional blocks that are followed by dense layers. The following model consists of-

- Block 1: Conv2D (32 filters, 3x3 kernel, relu) -> MaxPooling2D (2x2)
- Block 2: Conv2D (64 filters, 3x3 kernel, relu) -> MaxPooling2D (2x2)
- Block 3: Conv2D (64 filters, 3x3 kernel, relu)
- Flatten Layer
- Dense Layer: Dense (64 neurons, relu)
- Output Layer: Dense (10 neurons, softmax)

This model is compiled using the 'adam optimizer' which is an efficient stochastic gradient descent algorithm.

7. Results:

This model was trained for 10 epochs using model.fit() method. The accuracy of values and loss was reported during the draining correspond to the performance on the test set at the end of each epoch which was showed in the output as well. We took 50000 train images and for testing we took 10000 test images.

The Test Accuracy has been concluded at = **0.7187**.

The Plotting was shown in the output.

8. Conclusion:

The Project was successfully used for a Convolutional Neural Network (CNN) to classify images from the CIFAR-10 dataset. The model was able to reach the test accuracy around 71.87% which is quite a good result considering the simplicity of the network and the complexity of the dataset.

By analyzing the training history graph, we can visualize how the model learned over time. Moreover, these plots gave us some beneficial clues about the areas where the model might have started to overfit which assisted us better to understand its overall pefromance.