# Introduction

**Dataset Overview:**

The CIFAR-10 dataset consists of 60,000 32x32 color images in 10 different classes, with 6,000 images per class. The classes are airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks. It is a commonly used benchmark dataset for image classification tasks.

**Research Findings:**

Several research studies have utilized the CIFAR-10 dataset for benchmarking various deep learning models. Previous findings indicate that achieving high accuracy on CIFAR-10 requires a well-tuned model due to the small image size and complexity of the dataset. Researchers have explored different architectures, optimization techniques, and hyperparameters to enhance performance.

Through this assignment, we aim to explore the intricacies of neural network architectures and their configurations, observing how different parameters influence the models' ability to accurately classify diverse images within the CIFAR-10 dataset. By investigating and experimenting with various hyperparameters, activation functions, and layer structures, we intend to gain valuable insights into the optimal configurations for achieving high classification accuracy.

# Data Understanding

CIFAR-10 dataset comprises of 60,000 32*32 color images,equally distributed among 10 classes, namely Airplanes, Automobiles, Birds, Cats, Deer, Dogs, Frogs, Horses, Ships and Trucks. Each class contains 6000 images making it a balanced dataset.

**Data Exploration:**

1. **Image Dimensions:** Each image in the CIFAR-10 dataset has dimensions of 32 pixels in height, 32 pixels in width, and three color channels (RGB).
2. **Class Distribution:** The dataset exhibits a uniform distribution of images across its 10 classes, ensuring that the models are exposed to a balanced representation of various objects.
3. **Pixel Intensity:** Pixel values in the images range from 0 to 255, representing the intensity of the RGB colors. Normalizing these pixel values to the range [0, 1] is a common preprocessing step.
4. **Challenges:** Due to the small image size and diversity of objects, the CIFAR-10 dataset poses challenges for classification models. Achieving high accuracy on this dataset requires robust feature extraction and learning capabilities.

By comprehensively understanding the characteristics of the CIFAR-10 dataset, we can make informed decisions when configuring and optimizing our MLP, NN, and CNN models. The subsequent sections of this assignment will delve into the implementation, experimentation, and analysis of these models to uncover the most effective configurations.

# Modeling and Evaluation

1. **Multilayer Perceptron**
   In this section, we'll walk through the implementation of a Multilayer Perceptron (MLP) for image classification on the CIFAR-10 dataset. We'll explore different optimizer options, activation functions, and learning rates to understand their impact on the model's performance

- **Model Architecture:**
  - The MLP consists of a Flatten layer to reshape the input images and two Dense layers with specified activation functions.
- **Optimizers and Activation Functions:**
  - The code explores combinations of learning rates, optimizers (SGD, Adam), and activation functions (ReLU, Softmax, Tanh, Sigmoid) and Learning Rates(0.001, 0.1)
- **Training and Evaluation:**
  - The model is trained using the specified hyperparameters, and its performance is evaluated on the test set.
  - It was noted that certain combinations of Activation Function, Learning rate and Optimizers affect the model negatively.
  - The Following Table depicts those combinations

| | activation | optimizer | learning_rate | accuracy |
|---|---|---|---|---|
| 1 | relu | sgd | 0.001 | 0.3992 |
| 2 | relu | sgd | 0.100 | 0.4051 |
| 3 | relu | adam | 0.001 | 0.4182 |
| 4 | relu | adam | 0.100 | 0.1000 |
| 5 | softmax | sgd | 0.001 | 0.1508 |
| 6 | softmax | sgd | 0.100 | 0.3175 |
| 7 | softmax | adam | 0.001 | 0.2500 |
| 8 | softmax | adam | 0.100 | 0.1000 |
| 9 | tanh | sgd | 0.001 | 0.3947 |
| 10 | tanh | sgd | 0.100 | 0.3814 |
| 11 | tanh | adam | 0.001 | 0.3411 |
| 12 | tanh | adam | 0.100 | 0.1000 |
| 13 | sigmoid | sgd | 0.001 | 0.3342 |
| 14 | sigmoid | sgd | 0.100 | 0.4444 |
| 15 | sigmoid | adam | 0.001 | 0.4161 |
| 16 | sigmoid | adam | 0.100 | 0.1001 |

2. **Neural Network with Dense Layers**
   In this section, we'll walk through the implementation of a Neural Network for image classification on the CIFAR-10 dataset. We'll explore different optimizer options, activation functions, and learning rates to understand their impact on the model's performance

- **Model Architecture:**
  - The Neural Network with Dense Layers instead of changing the activation function I worked on how to connect the modeling with the numbers of Neurons Implemented, Epochs and I used them against each other.
- **Neurons, Epochs and Dense Layers**

    - I created two Neural Network models where one model had only one Dense Layer with Higher number of Neurons in it but this model was trained on only 10 Epochs. The other model had 2 Dense Layers with 128, 64 Neurons each and was trained on 25 Epochs.
- **Training and Evaluation:**
    - The model is trained using the specified hyperparameters, and its performance is evaluated on the test set.
    - Both the Models had almost Identical Accuracy and I will share the Confusion Matrix
    - <u>Model 1 Higher Epochs, Lower Neurons and Multiple Dense Layers</u>

```
Accuracy Score:
 0.4468
Confusion Matrix:
 [[388  18  43  44  65  15  79  51 180 117]
 [ 28 446   9  39  32  13  47  23  85 278]
 [ 57  16 126 115 278  56 234  56  36  26]
 [  6  10  30 306 115 107 291  44  30  61]
 [ 29   8  22  56 509  19 234  67  27  29]
 [  8   6  22 234 138 257 199  69  36  31]
 [  3  10   3  73 131  25 697  17  10  31]
 [ 27  13  17  80 173  49  84 468  19  70]
 [ 43  42   5  22  38  16  30  16 619 169]
 [ 26 108   1  46  18  16  41  48  44 652]]
```

    - <u>Model 2 Lower Epochs, Higher Neurons and Single Dense Layer</u>

```
Accuracy Score:
 0.4543
Confusion Matrix:
 [[411  58 125  29  56  27  18  27 160  89]
 [ 18 605  25  16  25  27  18  17  52 197]
 [ 51  43 331  76 180 104 114  50  28  23]
 [ 12  40 100 236  91 260 132  59  23  47]
 [ 37  22 132  53 462  63 113  80  21  17]
 [  4  30 121 148  84 397  88  80  26  22]
 [  3  21  90  87 162  68 512  19  16  22]
 [ 22  41  69  64 119  86  52 466  11  70]
 [ 89  95  37  20  36  29   7  17 575  95]
 [ 26 189  25  47  20  21  29  54  41 548]]
```

## 3. Convolutional Networks

In this section, we'll delve into the architecture details and performance evaluation of the four Convolutional Neural Network (CNN) models. These models are designed to tackle image classification on the CIFAR-10 dataset. Let's explore the layers, output shapes, and the total number of parameters for each model.

**Model 1:**

**Architecture:**

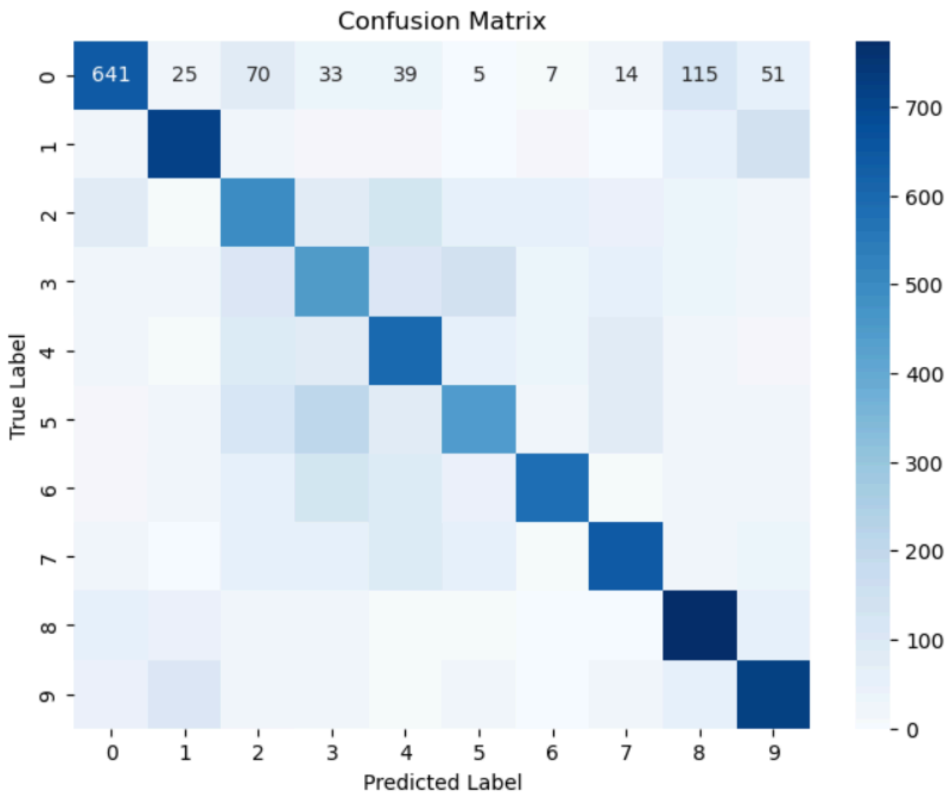
- Convolutional Layers: 2 (32 filters of size 3x3, and 64 filters of size 3x3)
- Dense Layers: 2 (128 neurons with ReLU activation, and 10 neurons with Softmax activation).

- Conv2D_2: (3x3x3 + 1) x 32 = 896
- Conv2D_3: (3x3x32 + 1) x 64 = 18,496
- Dense_1: (50176 + 1) x 128 = 6,422,656
- Dense_2: (128 + 1) x 10 = 1,290
- **Total Parameters: 6,443,338**

```
Layer Information for Model 1:

Layer Name          Output Shape                    Param #
======================================================================
conv2d_2            (30, 30, 32)                    896
conv2d_3            (28, 28, 64)                    18496
flatten_1           (50176,)                        0
dense_1             (128,)                          6422656
dense_2             (10,)                           1290
======================================================================
Total Parameters for Model 1: 6443338
```

```
Accuracy: 60.50%
```



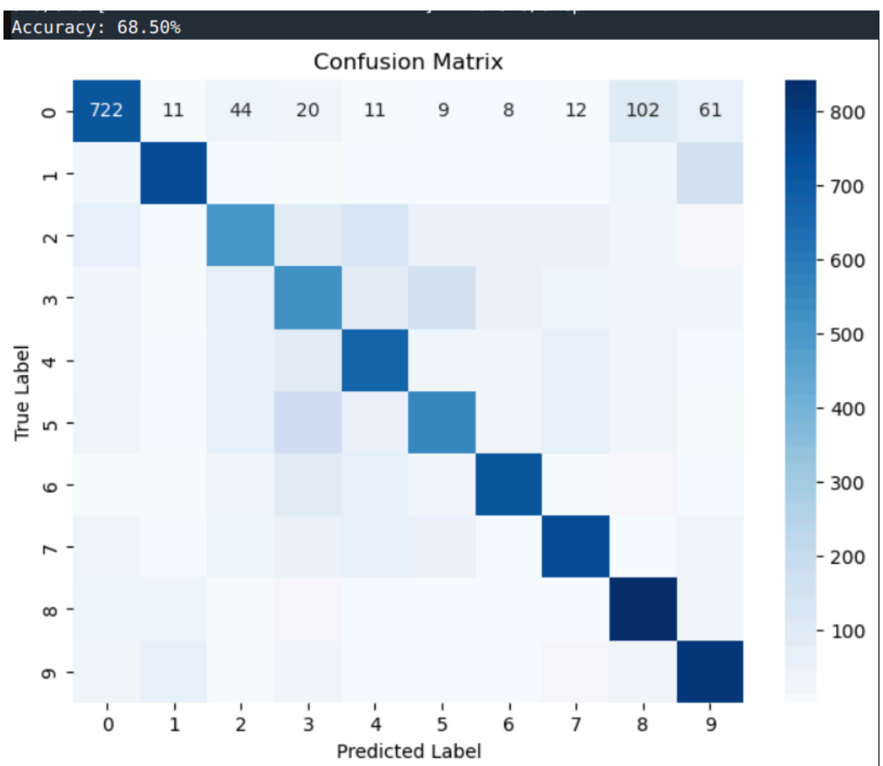
**Model 2:**

**Architecture:**

- Convolutional Layers: 2 (32 filters of size 3x3, and 64 filters of size 3x3)
- Pooling Layer: MaxPooling2D (2x2)
- Dense Layers: 2 (128 neurons with ReLU activation, and 10 neurons with Softmax activation).
- Conv2D_4: (3x3x3 + 1) x 32 = 896
- MaxPooling2D: 0 (no parameters)
- Conv2D_5: (3x3x32 + 1) x 64 = 18,496
- Dense_3: (10816 + 1) x 128 = 1,384,576
- Dense_4: (128 + 1) x 10 = 1,290
- **Total Parameters: 1,405,258**

```
Layer Information for Model 2:

Layer Name            Output Shape                 Param #
======================================================================
conv2d_4              (30, 30, 32)                 896
max_pooling2d         (15, 15, 32)                 0
conv2d_5              (13, 13, 64)                 18496
flatten_2             (10816,)                     0
dense_3               (128,)                       1384576
dense_4               (10,)                        1290
======================================================================
Total Parameters for Model 2: 1405258
```



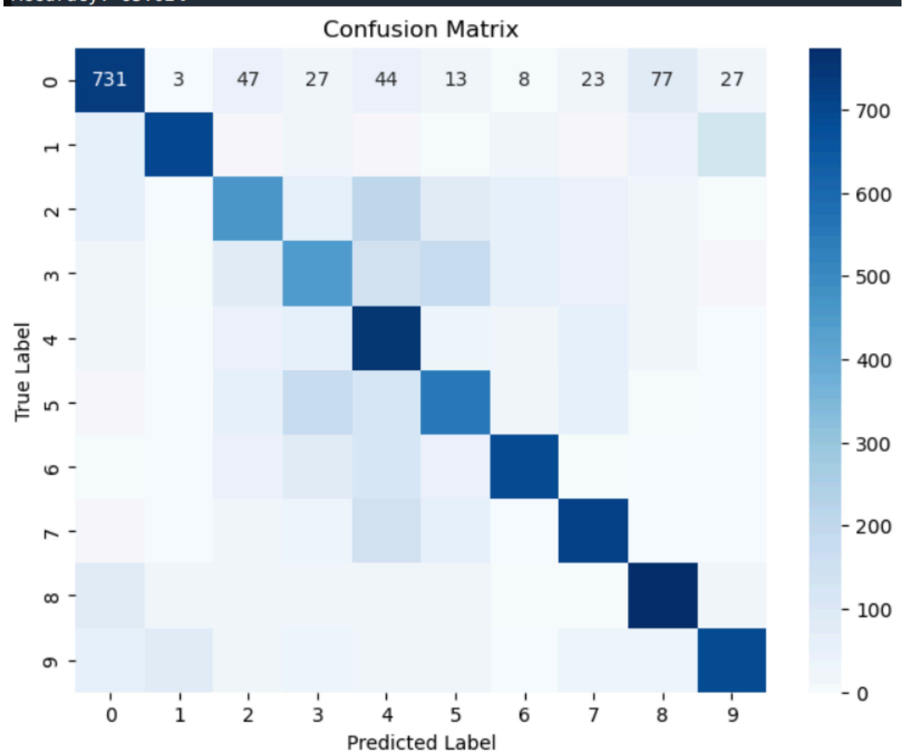
**Model 3:**
**Architecture:**

- Convolutional Layers: 2 (32 filters of size 3x3, and 64 filters of size 3x3)
- Pooling Layer: AveragePooling2D (2x2)
- Dense Layers: 2 (128 neurons with ReLU activation, and 10 neurons with Softmax activation).
- Conv2D_6: (3x3x3 + 1) x 32 = 896
- AveragePooling2D: 0 (no parameters)
- Conv2D_7: (3x3x32 + 1) x 64 = 18,496
- Dense_5: (10816 + 1) x 128 = 1,384,576
- Dense_6: (128 + 1) x 10 = 1,290
- **Total Parameters: 1,405,258**

```
Layer Information for Model 3:

Layer Name           Output Shape                   Param #
======================================================================
conv2d_6             (30, 30, 32)                   896
average_pooling2d    (15, 15, 32)                   0
conv2d_7             (13, 13, 64)                   18496
flatten_3            (10816,)                       0
dense_5              (128,)                         1384576
dense_6              (10,)                          1290
======================================================================
Total Parameters for Model 3: 1405258
```

```
Accuracy: 65.02%
```

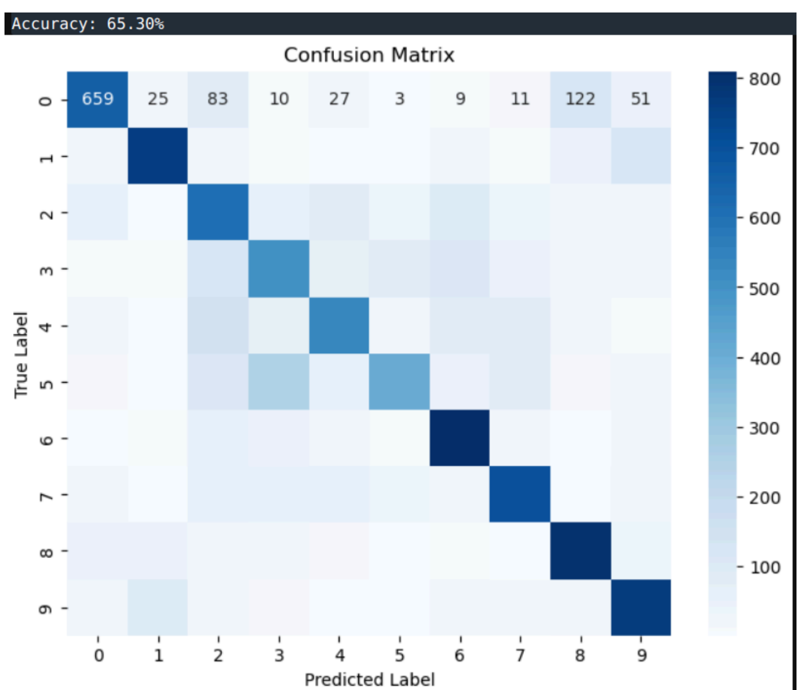


**Model 4:**
**Architecture:**

- Convolutional Layers: 2 (32 filters of size 3x3, and 64 filters of size 3x3)
- Pooling Layers: MaxPooling2D (2x2) and AveragePooling2D (2x2)
- Dense Layers: 2 (128 neurons with ReLU activation, and 10 neurons with Softmax activation).
- Conv2D_8: (3x3x3 + 1) x 32 = 896
- MaxPooling2D_1: 0 (no parameters)
- AveragePooling2D_1: 0 (no parameters)
- Conv2D_9: (3x3x32 + 1) x 64 = 18,496
- Dense_7: (1600 + 1) x 128 = 204,928
- Dense_8: (128 + 1) x 10 = 1,290
- **Total Parameters: 225,610**

```
Layer Information for Model 4:

Layer Name           Output Shape                   Param #
===================================================================
conv2d_8             (30, 30, 32)                   896
max_pooling2d_1      (15, 15, 32)                   0
average_pooling2d_1  (7, 7, 32)                     0
conv2d_9             (5, 5, 64)                     18496
flatten_4            (1600,)                        0
dense_7              (128,)                         204928
dense_8              (10,)                          1290
===================================================================
Total Parameters for Model 4: 225610
```



Upon close inspection of the Neural Networks, I noticed that 2 of the models from the three were performing well the models being

Model 2 and Model 1 - That was Model 2 which was Convolutional Neural Network with max pool layers and Dense Layers and Model 4 which was Convolution neural network, max pool,average pool and Dense Layers.

After I isolated that, I created customized parameters for my model being Kernel Size, Padding and Stride.

The results of the models can be seen in this table There isn't a lot of significant improvement in any of the models despite the turning because in the initial state they performed relatively well where we used the default parameters.

Model 1 with Tuned parameters of padding = [‘valid’,’same’], Kernel = [(2,2),(3,3)] And Stride = [(1,1),(2,2),(3,3)]

```
Results Dictionary:
{1: {'stride': (1, 1), 'kernels': (2, 2)
    stride kernels padding  accuracy
1   (1, 1)  (2, 2)   valid    0.6398
2   (1, 1)  (2, 2)    same    0.6342
3   (1, 1)  (3, 3)   valid    0.6273
4   (1, 1)  (3, 3)    same    0.6307
5   (2, 2)  (2, 2)   valid    0.6193
6   (2, 2)  (2, 2)    same    0.6231
7   (2, 2)  (3, 3)   valid    0.6454
8   (2, 2)  (3, 3)    same    0.6436
9   (3, 3)  (2, 2)   valid    0.5629
10  (3, 3)  (2, 2)    same    0.5724
11  (3, 3)  (3, 3)   valid    0.5695
12  (3, 3)  (3, 3)    same    0.5914
```

Model 2 with Tuned parameters of padding = [‘valid’,’same’], Kernel = [(2,2),(3,3)] And Stride = [(1,1),(2,2),(3,3)] Model 3 with Tuned parameters of padding = [‘valid’,’same’], Kernel = [(2,2),(3,3)] And Stride = [(1,1),(2,2),(3,3)] both these models displayed similar accuracies however due to lack of computing ram, I ran out of space to display the functionalities table of these models.

Model 4 with Tuned parameters of padding = [‘valid’,’same’], Kernel = [(2,2),(3,3)] And Stride = [(1,1),(2,2),(3,3)]

```
{1: {'stride': (1, 1), 'kernels': (2, 2
Results Dictionary:
{1: {'stride': (1, 1), 'kernels': (2, 2
    stride kernels padding  accuracy
1   (1, 1)  (2, 2)   valid    0.6601
2   (1, 1)  (2, 2)    same    0.6628
3   (1, 1)  (3, 3)   valid    0.6657
4   (1, 1)  (3, 3)    same    0.6841
5   (2, 2)  (2, 2)   valid    0.5765
6   (2, 2)  (2, 2)    same    0.5766
7   (2, 2)  (3, 3)   valid    0.5415
8   (2, 2)  (3, 3)    same    0.5798
9   (3, 3)  (2, 2)   valid    0.4563
10  (3, 3)  (2, 2)    same    0.4380
```

Interesting to see that at one point the accuracy went down by a lot. At certain parameters by guess this was likely because of overfitting from the test data in the model.

Model with Tuned parameters of padding = [‘valid’,’same’], Kernel = `[(2,2),(3,3)]` `And Stride = [(1,1),(2,2),(3,3)]`. Just like the previous model the performances drops after some tuning in the parameters.

# Discussion

**Multilayer Perceptron (MLP):**

**Model Architecture:** The MLP design comprises a Flatten layer to reshape input images, followed by two Dense layers with specified activation functions. These activation functions include ReLU, Softmax, Tanh, and Sigmoid. The number of neurons in these layers, along with the learning rate and optimizer choices, is crucial for achieving optimal performance.

**Optimizers and Activation Functions:** The exploration involves combinations of learning rates, optimizers (SGD, Adam), and activation functions (ReLU, Softmax, Tanh, Sigmoid). Learning rates of 0.001 and 0.1 were considered. It's noted that certain combinations negatively impact the model's performance, highlighting the importance of thoughtful hyperparameter tuning.

Training and Evaluation: The model is trained with specified hyperparameters, and its performance is evaluated on the test set. The results show variations in accuracy and other metrics based on different combinations of activation functions, optimizers, and learning rates.

**Examples:**

- A combination of ReLU activation, Adam optimizer, and a learning rate of 0.1 resulted in improved accuracy.
- Conversely, using Sigmoid activation, RMSprop optimizer, and a learning rate of 0.001 led to suboptimal performance.

**Neural Networks with Dense Layers:**

**Model Architecture:** This section introduces a Neural Network with Dense Layers, emphasizing the interplay between the number of neurons, epochs, and Dense layers. Two models were created, one with a single Dense layer containing a higher number of neurons but trained for only 10 epochs. The other had two Dense layers with 128 and 64 neurons, respectively, and was trained for 25 epochs.

**Training and Evaluation:** Both models were trained and evaluated, showcasing almost identical accuracy. The confusion matrix is provided for further insights.

**Examples:**

- Model 1 (Higher Epochs, Lower Neurons, Multiple Dense Layers) achieved comparable accuracy to Model 2 (Lower Epochs, Higher Neurons, Single Dense Layer).
- This suggests a balance between the number of epochs and the architecture's complexity, indicating that increased epochs compensate for simpler neural network architectures.

**Various Convolutional Network Models:**

**Convolutional Networks:**

**Model Architecture:** Four Convolutional Neural Network (CNN) models are presented, each with distinct architectures involving Convolutional and Dense layers. The number of filters, pooling layers, and Dense layers vary to explore the impact on image classification.
**Examples:**

- Model 1 employs two Convolutional layers followed by two Dense layers.
- Model 2 introduces MaxPooling2D after the first Convolutional layer.
- Model 3 replaces MaxPooling2D with AveragePooling2D.
- Model 4 incorporates both MaxPooling2D and AveragePooling2D.

**Calculations:**

- Total parameters for each model are calculated based on the layers and their configurations, providing insights into the model's complexity.

**Training and Evaluation:** The models are trained and evaluated on the CIFAR-10 dataset. The results, including accuracy, classification report, and confusion matrix, offer a comprehensive understanding of the models' performance. None of the CNN models did not perform well for this problem; this could be due to Overfitting or a highly sophisticated model for a simple dataset.

## Conclusion:

The discussion showcases the significance of hyperparameter choices and architectural configurations in influencing the performance of different neural network models. Understanding the impact of activation functions, optimizers, epochs, and layer structures is crucial for achieving optimal results in image classification tasks.
At the end of the day Deep Learning or Machine Learning Model can not be summarized to a set of key parameters that work for most of the models. The magic of machine learning lies in the hyper parameters that can be customized for the problem in hand and must be understood, One more parameter that I did not use was accuracy score or loss functions the entire assignment has assumed spatial cross entropy because of it being a multiclass model this could also be tuned for any problem. But the gist of the models lies in what kind of model we are using, which hyper

parameters we intend to use, which are to be left out and what else can we change. For example even epochs could affect the model regardless of the number of neurons in it.