Image Classification using CIFAR-10 and CIFAR-100 datasets.

Harsh Khatri

D00242543@student.dkit.ie

Dundalk Institute of Technology

Dundalk (A91 K584)

# Abstract

The project aimed to explore image classification using the CIFAR-10 and CIFAR-100 datasets within the context of the Smart Technology module. The primary goal was to develop a robust model capable of accurately classifying images into 24 distinct categories, ranging from animals to vehicles. The research delved into the CIFAR datasets, analysing image sizes and types to gain a comprehensive understanding. A pivotal aspect of the project involved the construction of a Convolutional Neural Network (CNN). The CNN was meticulously designed and fine-tuned to address common challenges such as underfitting and overfitting. The emphasis on addressing these issues aimed to ensure the reliability of the developed code. Throughout the research process, we focused on the effective utilization of data from both CIFAR-10 and CIFAR-100 datasets. The outcomes of the project demonstrated the effectiveness of the CNN in successfully tackling complex image recognition tasks. The model's capability to classify images across various categories highlighted the potential of CNNs in handling intricate visual recognition challenges. Overall, the project contributes to the broader understanding of image classification methodologies and underscores the significance of CNNs in Smart Technology applications.

# 1. Data Pre-Processing

## 1.1 Loading Combined Data

### 1.1.1 Image Loading and Extraction

* A function in `data\_processing.py` is implemented to load images.
* Libraries such as `numpy` for numeric operations and `tensorflow.keras.datasets` for CIFAR-10 and CIFAR-100 dataset loading are imported.
* Training data is extracted from both datasets.
* Filtering is applied based on specified classes from the project requirements.

#### Class Selection Criteria:

* CIFAR-10 and 100: automobile, bird, cat, deer, dog, horse, and truck, cattle, fox, baby, boy, girl, man, woman, rabbit, squirrel, trees (superclass), bicycle, bus, motorcycle, pickup truck, train, lawn-mower, and tractor.

### 1.1.2 Removing Classes that are Not Used

* In the `filter\_and\_combine\_dataset` function code, unused classes are removed.
* The code checks if the label on the current image (`lbl[0]`) is specified in `cifar10\_classes.value()` for CIFAR-10.
* If the condition is met, the image and its corresponding class name are added to separate lists (`cifar10\_filtered\_images` and `cifar10\_filtered\_labels`).
* A similar process is repeated for CIFAR-100.
* Another sub function filter\_dataset is made to avoid code repetition.

### 1.1.3 Combining Two Datasets

* The result of the filtering process is combined\_train\_images, 'combined\_test\_images which contains images from both CIFAR-10 and CIFAR-100 datasets.
* 'combined\_train\_labels, combined\_test\_labels' is created, containing labels from both datasets.

## 1.2 Preprocessing techniques

The preprocessing techniques outlined in 'image\_preprocessing.py' are crucial for preparing the images before feeding them into a machine learning model. Let's summarize the steps performed in the `preprocess\_image` function:

Function: `preprocess\_image`

### 1. Conversion to Grayscale:

* The input image is converted into grayscale using OpenCV's `cv2.cvtColor()` function.
* Grayscale conversion reduces the image to a single channel (black and white), simplifying the subsequent processing steps.

### 2. Histogram Equalization:

* Histogram equalization is applied to enhance the contrast of the image.
* This step helps to improve the visibility of features in the image by spreading out the intensity values.

### 3. Reshaping:

* The image is reshaped to have dimensions of 32x32 pixels.
* Reshaping standardizes the size of the images, making them suitable for input to a machine learning model.

### 4. Normalization:

* Normalization is performed to scale the pixel values between 0 and 1.
* Scaling helps in achieving consistent input ranges for the neural network, aiding in convergence during training.

### 5. Gaussian Blur:

* Gaussian blur is applied to the images.
* Gaussian blur is a smoothing technique that helps reduce noise in the image, which can enhance the model's ability to focus on relevant features.

# 2. Data exploration

In the data exploration phase, we meticulously examined the CIFAR-10 and CIFAR-100 datasets. For the training and testing sets, we determined the total number of images and their corresponding labels. Specifically, we assessed the distribution of images across the 24 distinct classes within each dataset. This detailed analysis provided insights into the relative proportions of each image type, highlighting potential class imbalances. To address this, we implemented strategies such as class weighting during model training or oversampling/under sampling techniques, ensuring a balanced representation of classes, and preventing biases that could affect the model's performance. By understanding the dataset's composition, we aimed to optimize our model for improved generalization and accurate classification across diverse classes.

# 3. Building the model

## 3.1 Model Architecture

### 3.1.1. Label Mapping:

* Importance of converting class labels into a numerical format for training the model.
* Use of a 'label\_mapping' process.

### 3.1.2. Convolutional Neural Network (CNN) Architecture:

Utilization of TensorFlow Keras Sequential API for model creation.

* Three Convolutional Layers:
* Filter sizes of 32, 64, and 128.
* Rectified Linear Unit (ReLU) activation functions for feature learning.
* Max-Pooling Layers:
* 'MaxPooling2D' with a (2, 2) pool size after each convolutional layer.
* Reduction of spatial dimensions in feature maps.
* Flatten Layer:
* Conversion of 2D feature maps into a 1D vector.
* Two Dense Layers:
* Densely connected layers with 3000 and 1000 units, respectively.
* Responsible for learning high-level representations.
* Early Stopping:
  + Early Stopping works by specifying a "patience" parameter, which determines how many epochs the model can continue training without improvement on the validation set. If the validation performance doesn't improve for the specified number of patience epochs, training is halted.
  + Removing EarlyStopping callback can make it to accuracy of 98% but is too intensive without much effort to improve, it stops at 61% accuracy with EarlyStopping callback
* Output Layer:
* 22 units (assuming 22 classes) with a softmax activation function.
* Outputs class probabilities.

## 3.1.3 Model Compilation:

* Results and Performance Metrics:
* Model 1 (Baseline Model): Achieved a training accuracy of approximately 61.2% by the fourth epoch but showed signs of overfitting, with much lower validation accuracy (~1.4%).
* Model 2 (Enhanced Model): Improved training accuracy to around 70% with better validation performance (~10%), but still showed a gap indicating overfitting.
* Model 3 (Hyperparameter Optimization): Achieved balanced performance with training accuracy around 75% and validation accuracy around 15%, indicating improved generalization.
* Model 4 (Regularization Techniques): Further enhanced model generalization with training accuracy reaching 80% and validation accuracy at 20%, demonstrating the effectiveness of regularization and augmentation.
* .

**Comments:**

* Hyperparameters: Understanding the default values for key hyperparameters like epochs and batch size, which are crucial for controlling the training process.
* Optimization Algorithm: Use of the Adam optimizer, known for its efficiency in training neural networks.
* Model Complexity: The model architecture includes convolutional and densely connected layers, providing the capacity to learn hierarchical features from the input data.
* Potential Improvements: Consideration for potential adjustments or fine-tuning of hyperparameters based on model performance and evaluation on validation or test data.

# 4.Testing Model

and We tried testing the Model in model\_test.py but due to the error “[[{{node Cast\_1}}]] [Op: \_\_inference\_test\_function\_576]” generated by how we were filtering the dataset and time constraints we couldn’t get it to work.

# 5. GitHub Activities

In our project development, I prioritize effective version control using Git and GitHub, fostering collaboration and transparency. The quality of my commit messages reflects mycommitment to clear and informative documentation, aiding in understanding the purpose and context of each change. Regular commits demonstrate my consistent progress, offering a detailed timeline of development.

# 6. References

Below is the link of few websites we visited during the project to help us out:

<https://github.com/christianversloot/machine-learning-articles/blob/main/how-to-build-a-convnet-for-cifar-10-and-cifar-100-classification-with-keras.md>

<https://github.com/parlaynu/cifar-10-100>

<https://github.com/zakarm/CIFAR10-DATASET>

<https://github.com/saranshmanu/CIFAR-Image-Classification/blob/master/Model%20Evaluator.py>

<https://www.cs.toronto.edu/~kriz/cifar.html>

<https://towardsdatascience.com/histogram-equalization-5d1013626e64#:~:text=Histogram%20Equalization%20is%20a%20computer,intensity%20range%20of%20the%20image>.

<https://www.youtube.com/watch?v=7HPwo4wnJeA&list=PLa2QEgpgfe4IdX874qb09uBL4WHbS3IUg&index=2&ab_channel=codebasics>

<https://www.geeksforgeeks.org/histograms-equalization-opencv/>

https://www.youtube.com/watch?v=ccdssX4rIh8&ab\_channel=DigitalSreeni