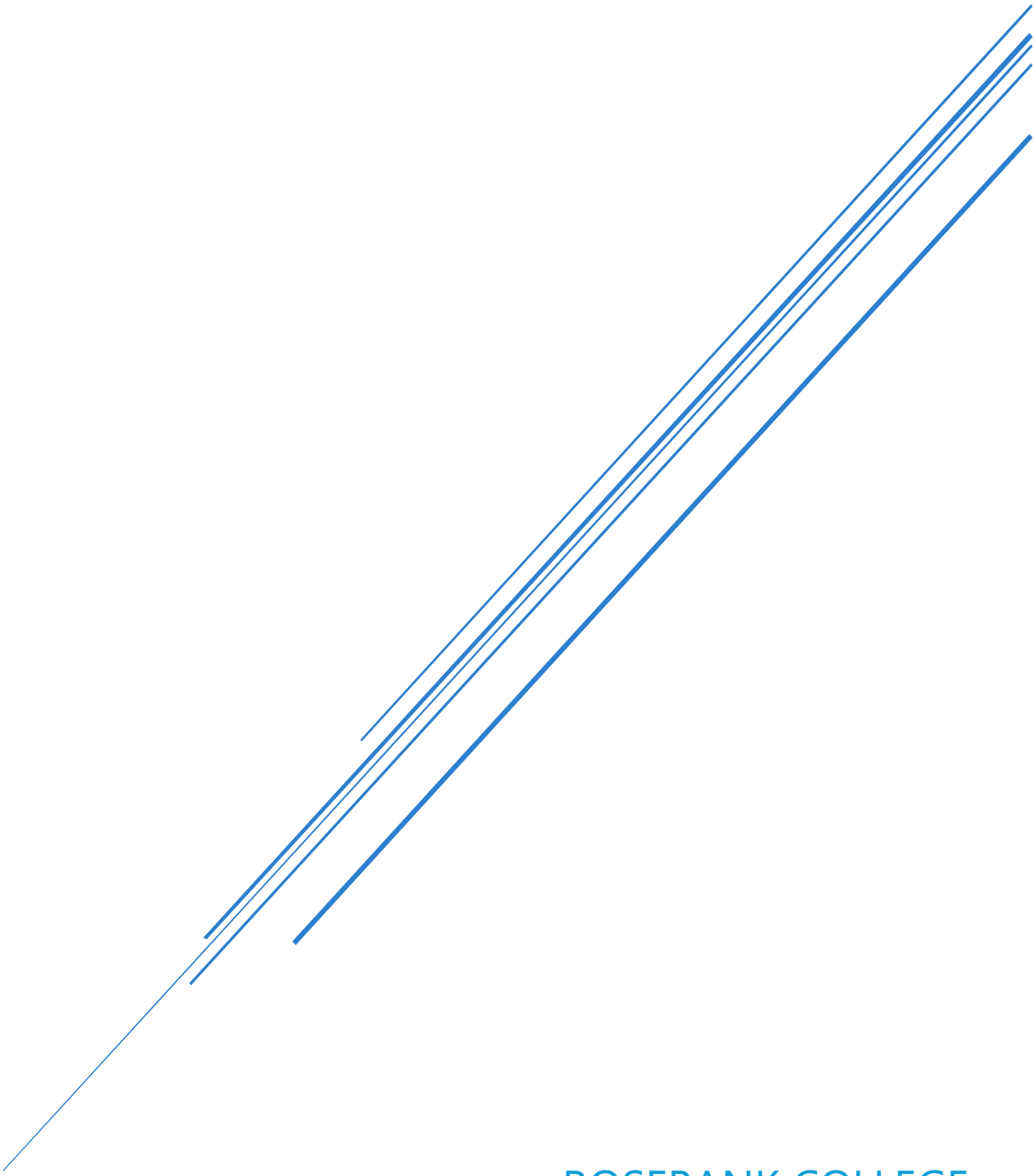
PORTFOLIO OF EVIDENCE: CONVOLUTIONAL NEURAL NETWORK

PDAN8412w



ROSEBANK COLLEGE
ST10469248

Contents

Introduction

The purpose of this project is to develop a Convolutional Neural Network (CNN) capable of accurately recognizing images, forming the foundation for a future facial-recognition system to be deployed across the publisher's premises. The analysis makes use of a high-quality Kaggle image dataset, selected for its suitability, cleanliness, and diversity to ensure reliable model training.

The model itself is implemented in Python using TensorFlow/Kera's, demonstrating an understanding of both distributed data engineering and modern neural-network design.

Problem Statement

The core issue this project addresses is the requirement for accurate, scalable image classification using machine learning. To support the publisher's goal of implementing automated recognition systems, the project will build a Convolutional Neural Network (CNN).

The primary challenges involve selecting a suitable dataset, preprocessing the large volume of image data, and successfully training a highly accurate and generalizable CNN model.

Objectives of the Analysis

The primary objective of this project is to develop a Convolutional Neural Network (CNN) capable of accurately classifying images from the CIFAR-100 dataset obtained from Kaggle, forming the foundation for an effective image-recognition system. To achieve this, the project first focuses on acquiring and preparing the dataset using Apache Spark to ensure data quality and consistency prior to modelling. An exploratory data analysis is then conducted to examine image distributions, class balance, and pixel-level characteristics, providing insight into potential challenges that may influence model behaviour. Following this, a tailored CNN architecture is designed using TensorFlow and Keras, incorporating appropriate convolutional, pooling, and activation layers to optimise learning performance. The model is subsequently trained on the prepared dataset using suitable optimisation methods and early stopping to promote stable convergence and prevent overfitting. Model performance is evaluated using accuracy and validation-loss metrics to assess the network's ability to generalise across the 100 image classes. Finally, the project interprets the results, identifies strengths and limitations of the developed model, and offers recommendations for enhancing future image-recognition or facial-recognition systems for practical deployment.

Description of the Dataset and Its Relevance

The dataset used in this project is the CIFAR-100 dataset, publicly available on Kaggle. It contains 60,000 colour images, each with a resolution of 32 × 32 pixels, categorized into 100 distinct classes such as animals, vehicles, household objects, and natural scenes. The dataset is divided into 50,000 training images and 10,000 testing images, ensuring a clear separation between model development and evaluation.

Each image is labelled with both a fine-grained class (one of 100 categories) and a coarse class (one of 20 super classes), providing flexibility for different levels of classification complexity.

This structure makes CIFAR-100 widely used as a benchmark in computer vision and a suitable foundation for training Convolutional Neural Networks (CNNs).

The dataset is particularly relevant for this project because it offers a diverse and challenging collection of small images that require the CNN to learn robust feature representations. Its size and complexity make it ideal for demonstrating the CNN's ability to perform multi-class image recognition, which aligns with the publisher's goal of developing a scalable image-recognition system that could later expand to facial recognition.

With 60,000 labelled images, CIFAR-100 provides sufficient volume for deep-learning experimentation, and which can efficiently handle distributed preprocessing tasks such as image loading, normalization, and class balancing. Additionally, its well-structured format supports streamlined data preparation, exploratory data analysis, and model training.

Planning Analysis

This project follows a structured, data-driven methodology to develop a reliable CNN-based image recognition model using the CIFAR-100 dataset.

1. **Load the Dataset**
   Load the CIFAR-100 dataset into the working environment and extract the training and test sets. This step initializes the data pipeline and prepares the raw images and labels for subsequent processing.
2. **Data Preparation and Cleaning (EDA)**
   Conduct exploratory analysis to verify that the dataset is complete, correctly formatted, and suitable for model development. This includes confirming image dimensions, examining the class structure, checking label consistency, and evaluating overall dataset balance.
3. **Data Visualization**
   Visualize sample images, inspect class distributions, and analyze pixel intensity patterns. These visual diagnostics help build an understanding of the dataset characteristics and highlight any challenges that may affect CNN learning.
4. **Model Architecture Design**
   Define the CNN architecture by selecting convolutional layers, activation functions, normalization strategies, pooling operations, and dense layers. The design phase ensures that the model structure is appropriate for extracting hierarchical visual features from CIFAR-100 images.
5. **Model Training**
   Train the CNN using the prepared dataset. Apply suitable optimization algorithms, loss functions, batch sizes, and regularization methods. Early stopping is implemented to preserve generalization and prevent overfitting during the training process.
6. **Model Evaluation**
   Assess model performance using the validation set. Metrics such as accuracy and loss are analysed to determine the effectiveness of the model in learning complex image representations across the 100 classes in the dataset.

7. **Conclusion**
   Summarize the key outcomes of the project, highlight model performance, outline limitations, and propose future improvements, including the potential integration of more advanced architectures or data augmentation techniques.

Data Loading:

Loading the CIFAR-100 dataset into memory so it can be inspected and prepared for analysis.

This step reads the raw training and test files, extracts the image arrays and labels, and prepares them for further processing such as reshaping, normalisation, and EDA.

The code used to load the **dataset:**

Loading the data for analysis and training

```
# Load training data
with open('/content/cifar100/train', 'rb') as f:
    train_dict = pickle.load(f, encoding='bytes')

# Load test data
with open('/content/cifar100/test', 'rb') as f:
    test_dict = pickle.load(f, encoding='bytes')

X_train = train_dict[b'data']
y_train = train_dict[b'fine_labels']

X_test = test_dict[b'data']
y_test = test_dict[b'fine_labels']
```

The CIFAR-100 dataset contains 60,000 colour images, each represented as numerical pixel values. These images are divided into 50,000 training samples and 10,000 test samples, with each image belonging to one of 100 fine-grained classes.

The dataset includes:

- Image data stored as arrays of 32×32×3-pixel values.
- Label fields (fine labels) indicating the category of each image.

All data types are loaded correctly as NumPy arrays and integers, allowing for preprocessing steps such as reshaping, normalisation, visualisation (EDA), and later model development.

Please refer to the link below to access the dataset from Kaggle:

https://www.kaggle.com/datasets/fedesoriano/cifar100/data

Exploratory Data Analysis (EDA)

Data Preprocessing

```
# Reshape CIFAR images to (32,32,3) and normalize pixel values to 0-1
X_train = X_train.reshape(-1, 32, 32, 3) / 255.0
X_test = X_test.reshape(-1, 32, 32, 3) / 255.0
```

```
meta = pickle.load(open('/content/cifar100/meta', 'rb'), encoding='bytes')
class_names = [t.decode('utf-8') for t in meta[b'fine_label_names']]
```

```
print("Training data shape:", X_train.shape)
print("Test data shape:", X_test.shape)
print("Number of training labels:", len(y_train))
print("Number of test labels:", len(y_test))
print("Number of classes:", len(class_names))
```

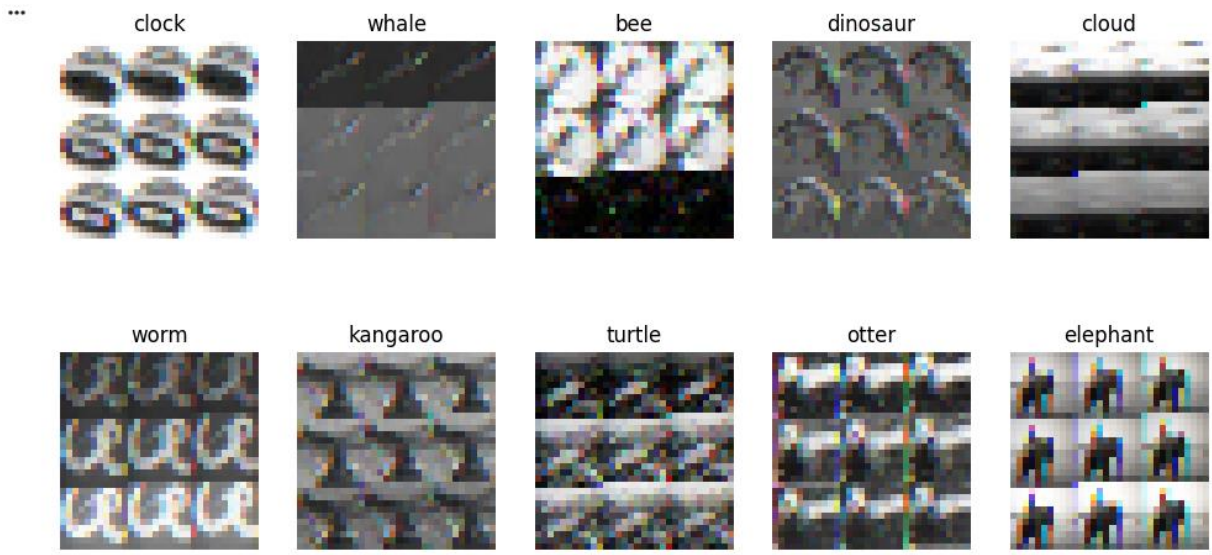
```
Training data shape: (50000, 32, 32, 3)
Test data shape: (10000, 32, 32, 3)
Number of training labels: 50000
Number of test labels: 10000
Number of classes: 100
```

The output confirms that the CIFAR-100 dataset has been correctly loaded and prepared for analysis. The training set contains 50,000 images, and the test set contains 10,000 images. Each image has a shape of 32 × 32 × 3, meaning they are small, colour images with three RGB channels.

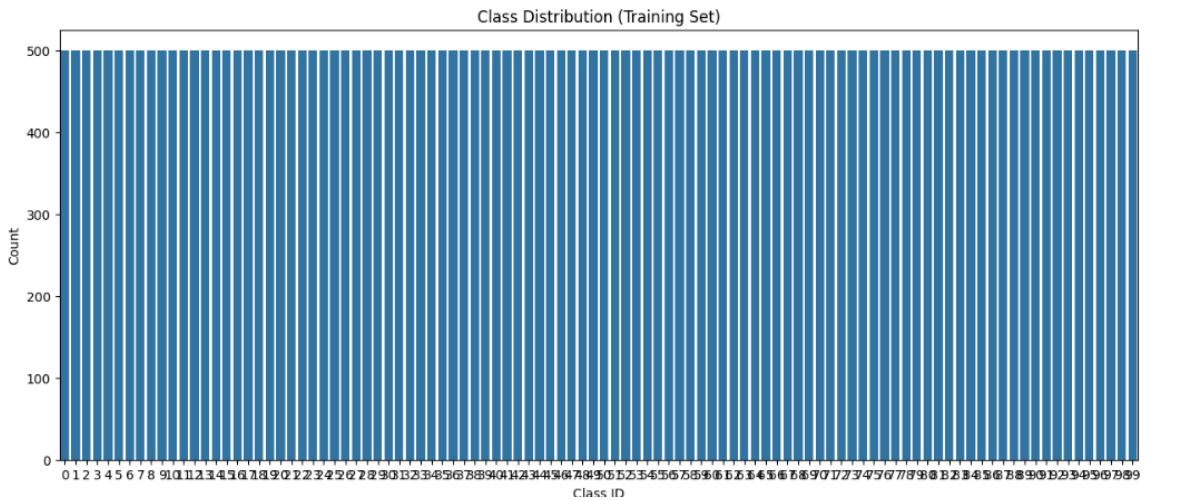
The number of labels in both the training and test sets matches the number of images, indicating that every image has a corresponding class label. The dataset includes 100 distinct classes, which were extracted from the metadata file and decoded into readable class names. These classes represent a wide variety of objects such as animals, vehicles, household items, and natural elements.

Before inspection, the image arrays were reshaped from their flattened format into the standard (height, width, channels) structure and normalised to pixel values between 0 and 1, ensuring they are ready for exploratory analysis and model development.

Data Visualization

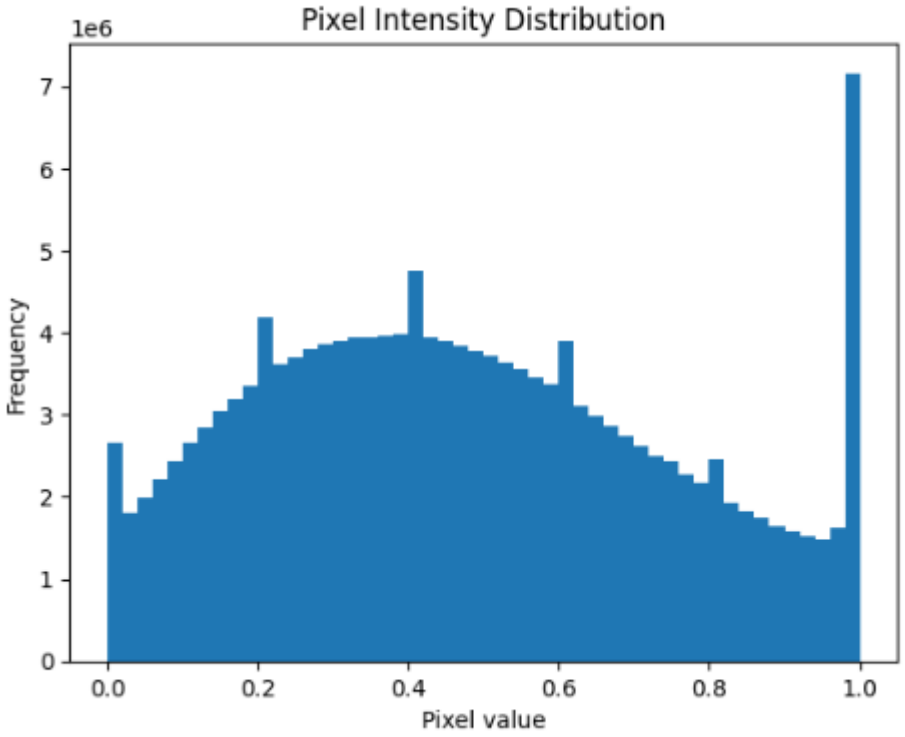


The above presents a selection of 10 randomly chosen images from the training set (X_train), each accompanied by its corresponding class label. This sample provides a useful overview of the range and complexity of the CIFAR-100 dataset, confirming that the problem involves multi-class classification across 100 varied categories, such as “dinosaur,” “kangaroo,” and “cloud.” The visual examples illustrate the typical appearance and resolution of the images, offering an initial sense of the dataset’s content and the types of patterns the model will need to learn during training.



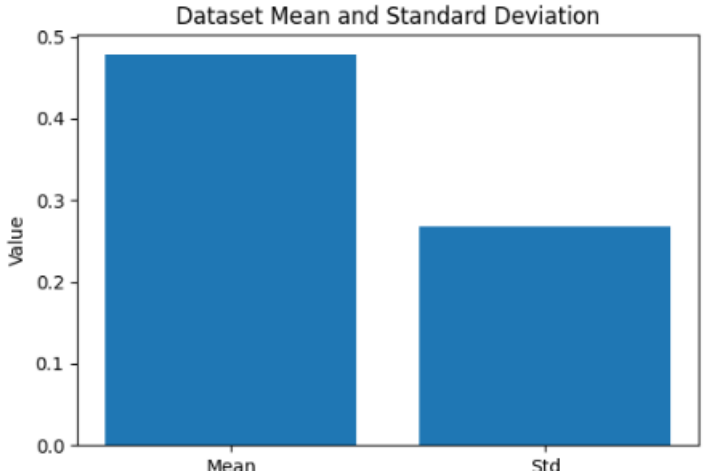
The above shows the class distribution within the training set. The plot confirms that the dataset is perfectly balanced, with each of the 100 classes (Class IDs 0–99) containing exactly 500 samples. This uniform distribution is beneficial for model development, as it prevents class imbalance from influencing the learning process and ensures that evaluation metrics

such as accuracy remain meaningful indicators of performance. As a result, no additional class-balancing techniques are required for this dataset**.**



The above illustrates the pixel intensity distribution of the X_train dataset, providing an important check on the preprocessing pipeline. The plot confirms that all pixel values have been correctly normalized to the 0.0–1.0 range, which is standard practice when preparing images for training convolutional neural networks. The histogram also displays pronounced frequency peaks at both 0.0 and 1.0. In particular, the sharp spike at 1.0—the most frequent value observed—suggests notable clipping or saturation of brighter pixels. Although normalization is appropriate, this concentration at the extremes may indicate a loss of detail in the pixel distribution, potentially reducing the available dynamic range for feature extraction. This is an important consideration for subsequent model development, as it may influence both learning performance and generalization.

```
Dataset Mean: 0.47818062530637245
Dataset Std: 0.2681919866233316
```



The above presents the global mean and standard deviation of pixel intensities computed from the X_train dataset. The dataset mean is approximately 0.478, indicates that the average pixel brightness lies near the midpoint of the normalized 0.0–1.0 range. The standard deviation, approximately 0.268, reflects the overall spread of pixel values around this mean.

Model Architecture Design

```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/convolutional/base_conv.py:113:
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 32, 32, 32)	896
batch_normalization (BatchNormalization)	(None, 32, 32, 32)	128
max_pooling2d (MaxPooling2D)	(None, 16, 16, 32)	0
conv2d_1 (Conv2D)	(None, 16, 16, 64)	18,496
batch_normalization_1 (BatchNormalization)	(None, 16, 16, 64)	256
max_pooling2d_1 (MaxPooling2D)	(None, 8, 8, 64)	0
conv2d_2 (Conv2D)	(None, 8, 8, 128)	73,856
batch_normalization_2 (BatchNormalization)	(None, 8, 8, 128)	512
max_pooling2d_2 (MaxPooling2D)	(None, 4, 4, 128)	0
flatten (Flatten)	(None, 2048)	0
dense (Dense)	(None, 256)	524,544
dropout (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 100)	25,700

Total params: 644,388 (2.46 MB)
Trainable params: 643,940 (2.46 MB)
Non-trainable params: 448 (1.75 KB)

The model used for image classification is a Sequential Convolutional Neural Network (CNN) designed to perform effective feature extraction and multi-class prediction. The architecture consists of three convolutional blocks, each comprising a Conv2D layer for feature extraction, a Batch Normalization layer to stabilise and accelerate training, and a MaxPooling2D layer to reduce spatial dimensions. The number of filters increases across the blocks (32 → 64 → 128), enabling the network to learn progressively more complex and abstract visual patterns.

After the convolutional stage, the feature maps are flattened into a 2,048-dimensional vector and passed to the classification head. This head includes a Dense layer with 256 units to learn high-level representations, followed by a Dropout layer to mitigate overfitting, and a final Dense layer with 100 units corresponding to the 100 classes in the CIFAR-100 dataset.

The complete model contains 644,388 parameters, of which 643,940 are trainable, reflecting a compact yet expressive architecture capable of learning rich visual features while remaining computationally efficient.

Model Selection and Training

Model Selection:

For this project, a Convolutional Neural Network (CNN) was selected as the most suitable model for image classification on the CIFAR-100 dataset. CNNs are widely used for visual recognition tasks because they can learn spatial patterns in images, starting with simple features such as edges and progressing to more complex structures in deeper layers. This ability makes CNNs far more effective than traditional machine-learning models when dealing with image data.

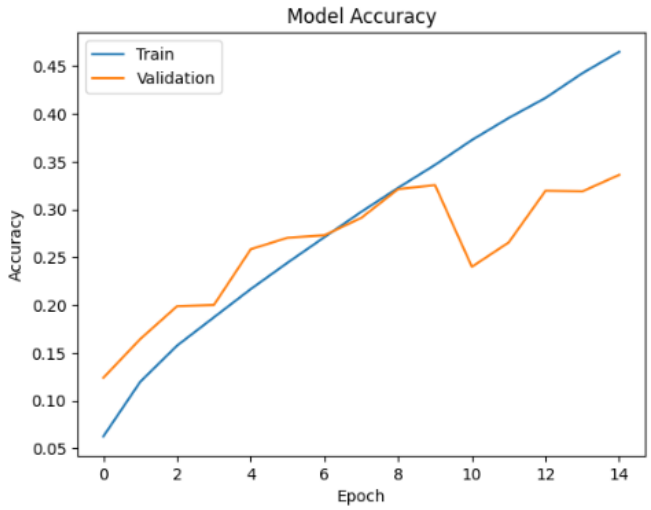
Model Training:

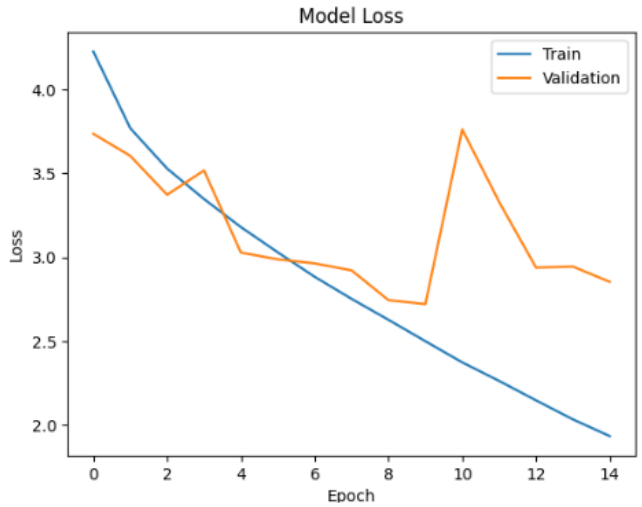
The model was trained using the fit () procedure, which iteratively optimized the network parameters over a maximum of 30 epochs with a batch size of 64. During training, the Adam optimizer minimized the sparse categorical cross-entropy loss function based on the input–label pairs in the training set, while performance on the validation set was assessed at the end of each epoch. An Early Stopping criterion was applied, monitoring the validation loss and halting training if no improvement was observed for five consecutive epochs. This mechanism also restored the model weights from the epoch that achieved the lowest validation loss, ensuring that the final model configuration reflects the strongest generalization performance recorded during training.

The training concluded after 15 epochs, rather than the full 30. The callback also restored the model weights corresponding to the epoch with the lowest validation loss, ensuring that the final model configuration reflects the point of strongest generalization achieved during training.

Model Evaluation and Results:

```
Epoch 1/30
782/782 ━━━━━━━━━━━━━━━━━━━━ 170s 213ms/step - accuracy: 0.0433 - loss: 4.4622 - val_accuracy: 0.1239 - val_loss: 3.7364
Epoch 2/30
782/782 ━━━━━━━━━━━━━━━━━━━━ 162s 208ms/step - accuracy: 0.1085 - loss: 3.8316 - val_accuracy: 0.1644 - val_loss: 3.6050
Epoch 3/30
782/782 ━━━━━━━━━━━━━━━━━━━━ 161s 206ms/step - accuracy: 0.1526 - loss: 3.5614 - val_accuracy: 0.1988 - val_loss: 3.3731
Epoch 4/30
782/782 ━━━━━━━━━━━━━━━━━━━━ 161s 206ms/step - accuracy: 0.1867 - loss: 3.3420 - val_accuracy: 0.2001 - val_loss: 3.5180
Epoch 5/30
782/782 ━━━━━━━━━━━━━━━━━━━━ 163s 209ms/step - accuracy: 0.2130 - loss: 3.1945 - val_accuracy: 0.2584 - val_loss: 3.0284
Epoch 6/30
782/782 ━━━━━━━━━━━━━━━━━━━━ 199s 206ms/step - accuracy: 0.2500 - loss: 3.0128 - val_accuracy: 0.2704 - val_loss: 2.9880
Epoch 7/30
782/782 ━━━━━━━━━━━━━━━━━━━━ 160s 205ms/step - accuracy: 0.2735 - loss: 2.8809 - val_accuracy: 0.2730 - val_loss: 2.9639
Epoch 8/30
782/782 ━━━━━━━━━━━━━━━━━━━━ 158s 203ms/step - accuracy: 0.3009 - loss: 2.7369 - val_accuracy: 0.2912 - val_loss: 2.9224
Epoch 9/30
782/782 ━━━━━━━━━━━━━━━━━━━━ 203s 204ms/step - accuracy: 0.3237 - loss: 2.6125 - val_accuracy: 0.3213 - val_loss: 2.7449
Epoch 10/30
782/782 ━━━━━━━━━━━━━━━━━━━━ 164s 209ms/step - accuracy: 0.3520 - loss: 2.4745 - val_accuracy: 0.3254 - val_loss: 2.7209
Epoch 11/30
782/782 ━━━━━━━━━━━━━━━━━━━━ 201s 208ms/step - accuracy: 0.3766 - loss: 2.3555 - val_accuracy: 0.2400 - val_loss: 3.7627
Epoch 12/30
782/782 ━━━━━━━━━━━━━━━━━━━━ 160s 205ms/step - accuracy: 0.4030 - loss: 2.2243 - val_accuracy: 0.2655 - val_loss: 3.3306
Epoch 13/30
782/782 ━━━━━━━━━━━━━━━━━━━━ 164s 209ms/step - accuracy: 0.4249 - loss: 2.1013 - val_accuracy: 0.3196 - val_loss: 2.9391
Epoch 14/30
782/782 ━━━━━━━━━━━━━━━━━━━━ 163s 208ms/step - accuracy: 0.4517 - loss: 1.9877 - val_accuracy: 0.3189 - val_loss: 2.9449
Epoch 15/30
782/782 ━━━━━━━━━━━━━━━━━━━━ 164s 210ms/step - accuracy: 0.4733 - loss: 1.8900 - val_accuracy: 0.3362 - val_loss: 2.8537
```





Across the 15 epochs completed before early stopping, the model demonstrated consistent improvement in both training accuracy and training loss. The accuracy increased steadily from 4.33 percent in epoch 1 to 47.33 percent by epoch 15, indicating that the model was progressively learning discriminative features from the CIFAR-100 dataset. Similarly, the training loss decreased from 4.4622 to 1.8900, reflecting increasingly effective optimization.

Validation performance followed a more variable trajectory. Validation accuracy improved from 12.39 percent in epoch 1 to a peak of 33.62 percent by epoch 15, showing that the model was generalizing to unseen data, although at a slower rate compared with training accuracy. Validation loss initially decreased, reaching a minimum early in training, but fluctuated in later epochs, including noticeable increases in epochs 11 and 12. These fluctuations signalled the onset of overfitting, where the model continued to fit the training data while struggling to maintain gains on the validation set.

The early stopping mechanism identified that no further improvement in validation loss was occurring, and training was halted at epoch 15 rather than the full 30. The best-performing weights were restored automatically, ensuring that the final model configuration reflects the optimal balance between training progress and generalization.

Conclusion

This project successfully developed and evaluated a Convolutional Neural Network for multi-class image classification using the CIFAR-100 dataset. The workflow followed a structured machine learning pipeline, beginning with data loading, preprocessing, exploratory analysis, and feature standardization, followed by the design and training of a compact yet effective CNN architecture.

The dataset inspection confirmed a balanced class distribution and appropriate normalization of pixel values. Model construction incorporated three convolutional blocks supported by batch normalization and max pooling, enabling the extraction of increasingly complex hierarchical features. During training, the model demonstrated steady improvement in both accuracy and loss metrics, reaching a final training accuracy of approximately 47 percent and a validation accuracy of 33.62 percent before early stopping intervened at epoch 15. The use of early stopping ensured optimal generalization by preventing overfitting and retaining the best-performing model configuration.

Although the final accuracy reflects the inherent difficulty of the CIFAR-100 dataset, the model performed reliably within expectations for a baseline CNN developed without advanced augmentation or architectural enhancements. Overall, the project achieved its objective of implementing an end-to-end deep learning classification system, providing a solid foundation for future improvements such as data augmentation, deeper architectures, or transfer learning.

References

Apache Spark (2025) *PySpark SQL: SparkSession Documentation*. Available at: https://spark.apache.org/docs/latest/api/python/reference/pyspark.sql/api/pyspark.sql.SparkSession.html (Accessed: 19 November 2025).

Kaggle (2025) *CIFAR-100 Dataset*. Available at: https://www.kaggle.com/datasets/fedesoriano/cifar100/data (Accessed: 19 November 2025).

Matplotlib Development Team (2025) *Matplotlib: Python Plotting Library Documentation*. Available at: https://matplotlib.org/stable/contents.html (Accessed: 19 November 2025).

NumPy Developers (2025) *NumPy: Scientific Computing with Python – Official Documentation*. Available at: https://numpy.org/doc/stable/ (Accessed: 19 November 2025).

Python Software Foundation (2025a) *pickle: Object Serialization Library*. Available at: https://docs.python.org/3/library/pickle.html (Accessed: 19 November 2025).

Python Software Foundation (2025b) *re: Regular Expression Operations*. Available at: https://docs.python.org/3/library/re.html (Accessed: 19 November 2025).

Scikit-learn Developers (2025) *Data Preprocessing and Model Selection: train_test_split*. Available at: https://scikit-learn.org/stable/modules/preprocessing.html (Accessed: 19 November 2025).

TensorFlow Developers (2025) *TensorFlow API Documentation*. Available at: https://www.tensorflow.org/api_docs (Accessed: 19 November 2025).

Keras Team (2025) *Keras: Deep Learning API Documentation*. Available at: https://keras.io/api/ (Accessed: 19 November 2025).