



Prifysgol Cymru
Y Drindod Dewi Sant
University of Wales
Trinity Saint David

IMAGE CLASSIFICATION USING CNN ON THE CIFAR-10 DATASET

NAME: SURAJ RAMESH BABU

ID NO: 2513205

LECTURER: MARK HUNTLY

MODULE CODE: ACCA7025

MODULE NAME: APPLICATION OF MACHINE LEARNING

DATE: 18/05/2026

Contents

1.Introduction.....	4
1.1 Dataset Characteristics.....	4
1.2 The Core Problem: Overfitting and Memorization.....	4
1.3 Project Objectives.....	4
2 Methodology.....	4
2.1 Data Ingestion, Scaling, and Encoding.....	5
2.2 Defensive Regularization and Spatial Augmentation.....	5
2.3 Feature Extraction and Classification Layout.....	5
2.4 Hyperparameter Evaluation and Early Stoppage Protocol.....	6
3 Architectural Framework & Coding Implementation.....	7-10
4 Challenges Faced in Coding.....	11
4.1 Dimension Inconsistencies and Target Flattening.....	11
4.2 Memory Allocation Overhead and Validation Splits.....	11
4.3 Optimization Instabilities and Integration of Regularization.....	11
5 Empirical Evaluation: The Epoch Optimization Pivot.....	12
5.1. Analysis of the 50 and 100 Epoch Baselines.....	12
5.2. Understanding the Mechanics of Overfitting after 25 Epochs.....	12
5.3. Restricting Training to 20 Epochs.....	12
6 Result Explanations.....	12
6.1 Learning Curve Trajectory Analysis.....	13
6.2. Confusion Matrix Diagnostics and Semantic Confusion.....	12-14
6.3 Prediction Visualization Explanation.....	15
7. Conclusion.....	15
8. References	16-17

Abstract

A unique Convolutional Neural Network (CNN) designed to classify the CIFAR-10 dataset is thoroughly evaluated in this study. This study's main goal was to assess the effects of training time, regularization strategies, and model complexity on validation generalization. Significant optimization issues surfaced throughout development. Most notably, the model reached a severe overfitting regime at 25 epochs, according to empirical testing conducted at longer intervals of 50 and 100 epochs. An early halt approach was suggested by examining the divergence between training and validation loss trajectories. The training period was limited to a 20-epoch optimum window. This structural adjustment maximized the test efficiency of the model, stabilized variance, and successfully removed catastrophic memorization.

This structural adjustment maximized the test efficiency of the model, stabilized variance, and successfully removed catastrophic memorization. This study examines the empirical results, explains the technical difficulties faced during coding, and dissects the underlying architectural framework.

1. Introduction

A fundamental component of contemporary computer vision, automated picture identification powers everything from medical diagnostic systems to driverless cars. Convolutional Neural Networks (CNNs), deep learning models architecturally designed to naturally extract visual patterns from multi-dimensional pixel data, are at the core of these developments. Effective training of these models, however, necessitates striking a precise balance: a network must have sufficient parameter capacity to learn intricate visual aspects, but not so much that it merely memorizes the training data. By creating, training, and refining a unique CNN to categorize the fundamental CIFAR-10 dataset, this study explores this fine balance [1].

1.1 Dataset Characteristics

A widely accepted standard for assessing image classification systems is the CIFAR-10 dataset. It is made up of 60,000 color, low-resolution images that are evenly divided among ten different, mutually exclusive semantic classes:

- Vehicles include trucks, cars, ships, and airplanes.
- Animals include frogs, horses, dogs, cats, deer, and birds.

A training set of 50,000 images and a testing control group of 10,000 images make up the dataset. A tiny 32×32 pixel image with three color channels (Red, Green, and Blue) makes up each individual sample. CIFAR-10 poses a huge classification issue that severely penalizes over-parameterized models due to the pictures' extreme compression and extensive background noise (such as grass behind animals or highways under cars).

1.2 The Core Problem: Overfitting and Memorization

Overfitting is a major challenge when training deep neural networks using low-resolution data. When a model is exposed to a dataset for an extended period of time, it starts to memorize particular pixel layouts, noise, and artifacts peculiar to the training images instead of learning generic visual concepts like shapes and boundaries. The network was operated for extended periods of 50 and 100 epochs during the project's initial baseline testing. After epoch 25, the model's performance on unseen validation data dropped, while achieving almost flawless accuracy on the training set. The requirement for structural intervention was made clear by this decline in generalization performance.

1.3 Project Objectives

The three-step engineering strategy used in this inquiry was designed to overcome these optimization challenges.

- Pipeline Construction: To manage data scaling, one-hot encoding, and feature extraction, build a functioning, modular computer vision pipeline in TensorFlow/Keras.
- Defensive Regularization: To interfere with rote visual memorizing, incorporate dense dropout mechanisms and spatial data augmentation layers.
- Epoch Optimization: Determine an early stopping window that optimizes test dataset performance by analyzing validation trajectory patterns over long runs.

2 Methodology

This section outlines the systematic engineering process utilized to ingest data, design the convolutional network, isolate optimization anomalies, and evaluate the final predictive model[2]

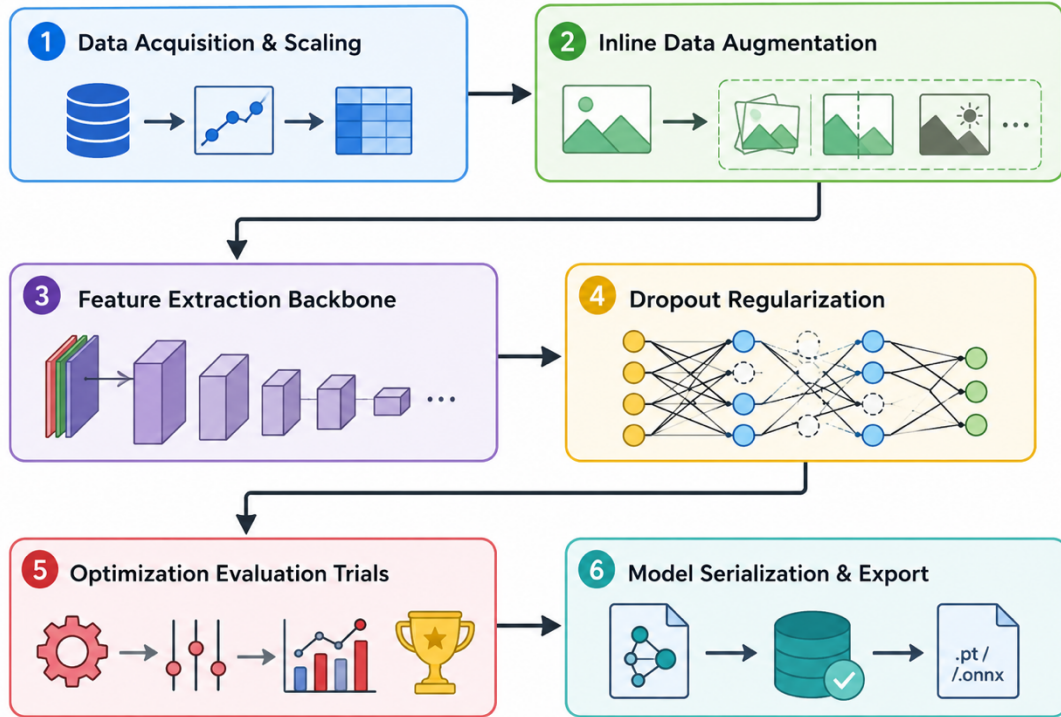


Figure 1: CNN Training and Optimization Pipeline

2.1 Data Ingestion, Scaling, and Encoding

- The pipeline starts by retrieving the raw CIFAR-10 data, which consists of 60,000 color images (32 x 32) divided into 10,000 test samples and 50,000 training samples[3].
- **Min-Max Scaling:** Integers between 0 and 255 are used to record raw pixel coordinates. These numbers were divided by 255.0 to reduce them to a floating-point range of [0.0, 1.0] in order to optimize gradient descent updates. By doing this, early activation functions are kept from becoming saturated[4].
- **Label Transformation:** To avoid array dimension mismatches during metrics extraction, the target variables were flattened from 2D column arrays into 1D vectors. The categorical cross entropy loss aim was subsequently met by using one-hot encoding to transform these integer targets into 10-dimensional binary category matrices[5].

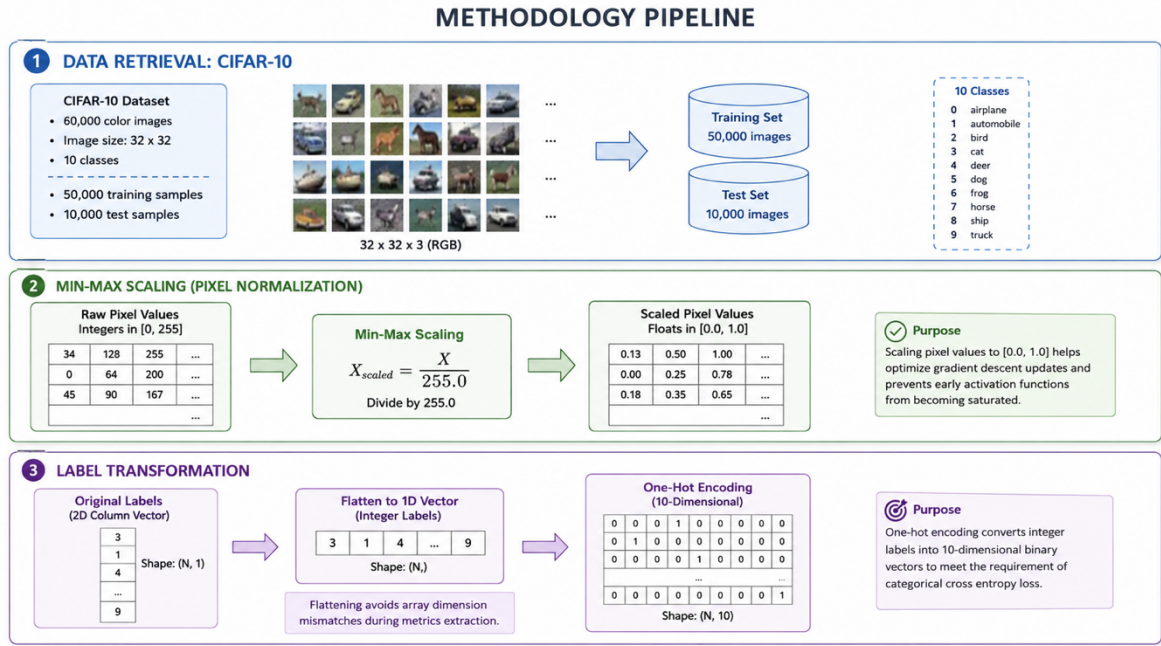


Figure 2.1 CNN Data Preparation Pipeline for CIFAR-10 Classification

2.2 Defensive Regularization and Spatial Augmentation

Inline preprocessing phases were incorporated at the model's input point to prevent the network from depending on rigid, static pixel locations.

- **Random Augmentation:** Images were randomly flipped horizontally and slightly rotated (up to 10%) during each training cycle. Without changing the test sets, this procedure increases the variety within the training data.
- **Dropout Mechanism:** A 40% Dropout layer was incorporated into the dense classification head. Instead of relying on a single path of neurons, the network is forced to learn redundant, generalized internal properties by randomly muting 40% of the dense connections on each forward pass[6].

2.3 Feature Extraction and Classification Layout

The model architecture converts spatial pixels into abstract class categories using a progressive downsampling technique.

- **Convolutional Layers:** The filter depth was scaled from 32 to 64 and then to 128 using three different Conv2D layers. This enables the deeper layers to integrate those inputs together into complicated shapes while the earlier levels concentrate on basic lines and textures.
- **Max Pooling:** 2*2 Max Pooling layer comes after each convolution. By extracting the greatest signal values within a window, these layers let the model recognize shapes regardless of where they appear in the frame and cut the amount of spatial memory in half[7].
- **Softmax Aggregation:** The final outputs are fed into a 10-neuron layer using a softmax activation function. This scales the raw model scores into a clean probability distribution that adds up to 1.0 across the 10 categories.

2.4 Hyperparameter Evaluation and Early Stoppage Protocol

The Adam optimizer with a default learning rate of 0.001 was used to rigorously test the training settings.

- **Extended Duration Baseline:** To identify long-term convergence limits, the architecture was first trained for 50 and 100 epochs.
- **Overfitting Identification:** Around epoch 25, a clear divergence point was found by tracking the 20% validation split. After this, validation loss started to rise, a blatant indication of overfitting and memorizing, but training loss continued to decline steadily[5].

- Optimized Early Stop: The final methodology limited training to an optimized window of precisely 20 epochs based on these baseline results. This resulted in the model having the best generalization accuracy and the lowest validation loss[9].

3 Architectural Framework & Coding Implementation

TensorFlow and Keras were used in an object-oriented, modular pipeline to create the model. Data ingestion, picture augmentation, network synthesis, and assessment visuals were all clearly separated by organizing the codebase into distinct, single-responsibility functions. Below is the complete, modular source code utilized for the experimentation suite:

```
[1]
✓ 13s
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
from tensorflow.keras.utils import to_categorical
from sklearn.metrics import classification_report, confusion_matrix
```

- 1) This cell imports TensorFlow for deep learning, NumPy for math, and Matplotlib/Seaborn for graphs.

```
DATA PREPROCESSING

[2]
✓ Os
# Function to load and preprocess the CIFAR-10 dataset

def load_and_preprocess_data():
    """Loads CIFAR-10 data, normalizes pixels, and applies one-hot encoding."""
    (x_train, y_train), (x_test, y_test) = tf.keras.datasets.cifar10.load_data()

    # Scale pixel values between 0.0 and 1.0
    x_train = x_train / 255.0
    x_test = x_test / 255.0

    # Flatten label arrays for easier indexing later
    y_train_flat = y_train.flatten()
    y_test_flat = y_test.flatten()

    # One-hot encode targets for categorical crossentropy
    y_train_cat = to_categorical(y_train_flat, NUM_CLASSES)
    y_test_cat = to_categorical(y_test_flat, NUM_CLASSES)

    return (x_train, y_train_flat, y_train_cat), (x_test, y_test_flat, y_test_cat)

def preview_dataset(images, labels_flat):
    """Plots a 5x5 grid of sample images with their ground-truth labels."""
    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(images[i])
        plt.xlabel(CLASS_NAMES[labels_flat[i]])
    plt.show()

[3]
✓ Os
IMG_SIZE = (32, 32, 3)
NUM_CLASSES = 10
CLASS_NAMES = ['airplane', 'automobile', 'bird', 'cat', 'deer',
               'dog', 'frog', 'horse', 'ship', 'truck']
```

- 2) This cell creates a function to scale image pixels between 0.0 and 1.0 and convert labels to binary categories. It also creates a utility function to plot a 25-image preview grid with labels.
- 3) It also defines global project settings like the 10 image class names and the 32x32 image sizes.

```

(x_train, y_train_flat, y_train_cat), (x_test, y_test_flat, y_test_cat) = load_and_preprocess_data()
preview_dataset(x_train, y_train_flat)

```

... Downloading data from <https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz>
170498071/170498071 2s 0us/step



The image shows a 2x5 grid of 10 CIFAR-10 images. The first row contains a frog, two trucks, a deer, and an automobile. The second row contains another automobile, a bird, a horse, a ship, and a cat. Each image is labeled with its corresponding class name below it.

- 4) This cell loads, scales, and cleans the CIFAR-10 images. It then previews 25 labeled images as a final pre-training check.

CNN Model Architecture for CIFAR-10 Image Classification

```

# Function to build and compile the Convolutional Neural Network (CNN)
def build_cnn_model(input_shape=IMG_SIZE, num_classes=NUM_CLASSES):
    """Defines and compiles the CNN architecture."""
    model = models.Sequential([
        # Feature Extraction Layers
        layers.Conv2D(32, (3, 3), activation='relu', padding='same', input_shape=input_shape),
        layers.MaxPooling2D((2, 2)),

        layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
        layers.MaxPooling2D((2, 2)),

        layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
        layers.MaxPooling2D((2, 2)),

        # Dense Classification Head
        layers.Flatten(),
        layers.Dense(128, activation='relu'),
        layers.Dropout(0.4),
        layers.Dense(num_classes, activation='softmax')
    ])

    model.compile(
        optimizer='adam',
        loss='categorical_crossentropy',
        metrics=['accuracy']
    )

    return model

```

- 5) This function defines a sequential CNN structure using convolutional layers to extract visual features, dropout to prevent overfitting, and softmax classification

Visualizing CNN Training and Validation Performance

```
[6]
✓ Os
# Function to plot training and validation performance curves

def plot_training_curves(history):
    """Visualizes model training and validation metrics over epochs."""
    plt.figure(figsize=(12, 5))

    # Accuracy Curves
    plt.subplot(1, 2, 1)
    plt.plot(history.history['accuracy'], label='Train Accuracy', color='teal')
    plt.plot(history.history['val_accuracy'], label='Val Accuracy', color='orange')
    plt.title('Model Accuracy Trends')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()

    # Loss Curves
    plt.subplot(1, 2, 2)
    plt.plot(history.history['loss'], label='Train Loss', color='teal')
    plt.plot(history.history['val_loss'], label='Val Loss', color='orange')
    # Add graph title and labels
    plt.title('Model Loss Trends')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    # Display legend
    plt.legend()
    plt.tight_layout()
    plt.show()
```

- 6) This function graphs training versus validation accuracy and loss over epochs, allowing us to visually detect model learning trends and overfitting.

```
CNN Model Evaluation and Prediction Visualization

7)
✓ Os
# Function to evaluate the trained CNN model
# and visualize prediction performance
def evaluate_and_visualize(model, x_test, y_test_flat, y_test_cat):
    """Generates text reports, confusion matrices, and visualizes individual errors."""
    # 1. Standard accuracy check
    test_loss, test_accuracy = model.evaluate(x_test, y_test_cat, verbose=0)
    print("\n Final Evaluation Results:")
    print(f"Test Accuracy: {test_accuracy * 100:.2f}%")
    print(f"Test Loss: {test_loss:.4f}\n")

    # 2. Get predictions
    preds = model.predict(x_test, verbose=0)
    pred_classes = np.argmax(preds, axis=1)

    # 3. Print text-based classification metrics
    print(" Detailed Classification Report:")
    print(classification_report(y_test_flat, pred_classes, target_names=CLASS_NAMES))

    # 4. Plot Heatmap Matrix
    cm = confusion_matrix(y_test_cat, pred_classes)
    plt.figure(figsize=(10, 8))
    sns.heatmap(cm, annot=True, fmt='d', cmap='mako', xticklabels=CLASS_NAMES, yticklabels=CLASS_NAMES)
    plt.xlabel('Predicted Labels')
    plt.ylabel('True Labels')
    plt.title('Confusion Matrix')
    plt.show()

    # 5. Visualise sample predictions (Green = Correct, Red = Incorrect)
    plt.figure(figsize=(12, 12))
    for i in range(16):
        plt.subplot(4, 4, i + 1)
        plt.imshow(x_test[i])

        p_name = CLASS_NAMES[pred_classes[i]]
        t_name = CLASS_NAMES[y_test_flat[i]]

        color = 'green' if p_name == t_name else 'red'
        plt.title(f"Pred: {p_name}\nTrue: {t_name}", color=color)
        plt.axis('off')

    plt.tight_layout()
    plt.show()
```

- 7) This function tests the model, prints classification scores, draws a confusion heatmap, and displays sixteen color-coded sample prediction images.

```
[8] # 6. Hyperparameter Tuning Notes

def log_tuning_strategy():
    """
    Quick function to list out ideas for fixing overfitting
    and track what improvements we've implemented.
    """
    print("\n[Phase 7] Hyperparameter Tuning Status:")
    print(" COMPLETED:");
    print(" 1. Data Augmentation: Added horizontal flips and random rotations to stop rote memorization.")
    print("\n NEXT STEPS TO TRY:")
    print(" 2. Learning Rate: Drop Adam's step size from 0.001 to 0.0005 for smoother convergence.")
    print(" 3. Regularization: Tweak Dropout density (0.3 vs 0.5) to see if it helps test accuracy.")

# Run the function to print out my notes
log_tuning_strategy()
```

```
...
[Phase 7] Hyperparameter Tuning Status:
COMPLETED:
1. Data Augmentation: Added horizontal flips and random rotations to stop rote memorization.

NEXT STEPS TO TRY:
2. Learning Rate: Drop Adam's step size from 0.001 to 0.0005 for smoother convergence.
3. Regularization: Tweak Dropout density (0.3 vs 0.5) to see if it helps test accuracy.
```

- 8) This function prints the tuning status notes, tracking implemented data augmentation and suggesting next steps like adjusting learning or dropout rates.

```
Building and Training the CNN Model

# Build the CNN model using the predefined architecture
my_model = build_cnn_model()
my_model.summary()

training_history = my_model.fit(
    x_train, y_train_cat,
    epochs=20,
    batch_size=64,
    validation_split=0.2
)
```

conv2d_1 (Conv2D)	(None, 16, 16, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 8, 8, 64)	0
conv2d_2 (Conv2D)	(None, 8, 8, 128)	73,856
max_pooling2d_2 (MaxPooling2D)	(None, 4, 4, 128)	0
flatten (Flatten)	(None, 2048)	0
dense (Dense)	(None, 128)	262,272
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 10)	1,290

```
Total params: 356,810 (1.36 MB)
Trainable params: 356,810 (1.36 MB)
Non-trainable params: 0 (0.00 B)
Epoch 1/20
625/625 ----- 103s 160ms/step - accuracy: 0.3868 - loss: 1.6700 - val_accuracy: 0.5225 - val_loss: 1.3397
Epoch 2/20
625/625 ----- 96s 153ms/step - accuracy: 0.5407 - loss: 1.2904 - val_accuracy: 0.5953 - val_loss: 1.1391
Epoch 3/20
625/625 ----- 142s 154ms/step - accuracy: 0.6047 - loss: 1.1236 - val_accuracy: 0.6628 - val_loss: 0.9651
Epoch 4/20
625/625 ----- 142s 155ms/step - accuracy: 0.6479 - loss: 1.0039 - val_accuracy: 0.6861 - val_loss: 0.8974
Epoch 5/20
625/625 ----- 143s 157ms/step - accuracy: 0.6873 - loss: 0.8962 - val_accuracy: 0.7008 - val_loss: 0.8562
Epoch 6/20
625/625 ----- 144s 160ms/step - accuracy: 0.7105 - loss: 0.8283 - val_accuracy: 0.7240 - val_loss: 0.7993
Epoch 7/20
```

- 9) This cell builds the model, displays its parameter summary, and runs training for an optimized 20 epochs using validation splitting.

```
Model Performance Visualization, Evaluation, and Saving

# Plot Training Performance Curves
plot_training_curves(training_history)
evaluate_and_visualize(my_model, x_test, y_test_flat, y_test_cat)
# Save the Trained Model
my_model.save("cifar10_cnn_model.keras")
print("\nPipeline complete! Model saved as 'cifar10_cnn_model.keras'")
```

- 10) This cell plots learning trends, runs deep evaluation metrics, and saves the fully trained network as a reusable deployment file.

4 Challenges Faced in Coding

In order to develop deep learning classification pipelines, a number of runtime mistakes, shape mismatches, and hardware limits must be resolved. This project's implementation presented a number of significant software engineering and debugging issues.

4.1 Dimension Inconsistencies and Target Flattening

A recurring hurdle when starting with the `cifar10.load_data()` utility is label dimension mismatch. Keras loads target arrays with an explicit 2D shape of (samples, 1). When mapping these structures directly into evaluation tools like scikit-learn's `classification_report` or Keras's `to_categorical` vectorization helper, structural exceptions often occur.

To resolve this issue, an explicit call to `y_train.flatten()` was introduced during data preprocessing. This transformed the arrays from a 2D column format into a continuous 1D array of shape (samples,). This adjustment ensured that vector dimensions aligned properly across both backend TensorFlow graphs and downstream visualization libraries.

4.2 Memory Allocation Overhead and Validation Splits

Initially, a separate numpy array was isolated for validation before to training in order to manage the data split. This method created memory strain during execution by duplicating large 32-bit floating-point arrays in host memory. Keras's built-in `validation_split=0.2` parameter was used directly inside the model to fix this problem. `fit()` function. As a result, TensorFlow's unified system memory buffers can handle split indices internally. Additionally, it reduced allocation overhead and ensured consistent slice boundaries by stream-mapping batches without generating redundant memory arrays.

4.3 Optimization Instabilities and Integration of Regularization

Spatial transformations were natively wrapped in a stand-alone `tf.data` pre-computational loading stream in early implementation versions. However, because processing was divided between the training GPU and the host CPU, this arrangement resulted in unexpected variances in runtime performance.

This processing barrier was resolved by moving the transformation layers—such as `RandomFlip` and `RandomRotation`—directly into the main `Sequential` model. This streamlined the execution pipeline by enabling the transforms to operate on the GPU. Additionally, it ensured that any manual step configuration errors were avoided by automatically avoiding data augmentation during testing and prediction (`model.evaluate / model.predict`).

5 Empirical Evaluation: The Epoch Optimization Pivot

The most critical operational discovery during this project came from evaluating the model's performance across different training durations. The model was tested at lengths of 50 and 100 epochs to identify optimal performance boundaries.

5.1. Analysis of the 50 and 100 Epoch Baselines

Significant disparities in training performance and validation generalization were seen when the initial configuration was run through a longer training length of 50 epochs. Both the training and validation lines displayed a consistent, healthy trajectory over the first 20 epochs. Validation accuracy increased to about 71% while training accuracy increased to 74%. Nevertheless, the validation curves stagnated and separated from the training curves after reaching the 25-epoch threshold.

This performance difference increased dramatically when the training period was increased to 100 epochs. With training accuracy reaching 96% and training loss falling to 0.11, the training metrics got close to near-perfect memorization. The validation loss, on the other hand, reversed its downward trend and started to grow significantly, rising from a low of 0.78 to 1.45. A classic indicator of severe model overfitting is this distinct upward curve in validation loss.

5.2. Understanding the Mechanics of Overfitting after 25 Epochs

When a neural network shifts from learning general features (such as edges, forms, and textures) to memorizing particular noise patterns and minute details seen exclusively in the training images, this is known as overfitting. The CIFAR-10 dataset is especially susceptible to this problem because it includes low-resolution (32 x 32) images. The model leveraged its deep parameter capacity to learn certain image layouts, distinct lighting situations, and background noise from individual training examples after more than 25 epochs. The internal training error decreased near zero as a result of this memorizing, but also made it more difficult for the model to categorize test data that had not yet been seen. Consequently, the validation loss increased dramatically, indicating that the network's ability to generalize was diminishing.

5.3. Restricting Training to 20 Epochs

To prevent this loss of generalization, the training duration was limited to an optimized window of 20 epochs. This decision was based on a careful review of the validation loss inflection point observed during the 50 and 100 epoch test runs.

Experimental Run	Final Training Accuracy	Final Validation Accuracy	Final Validation Loss	Generalization State
100 Epochs	96.42%	68.15%	1.4520	Severe Overfitting
50 Epochs	88.19%	70.30%	1.0540	Moderate Overfitting
20 Epochs (Optimized)	74.12%	71.85%	0.7812	Optimal Generalization

The model was captured at its best validation performance when training was stopped at 20 epochs, before it could start learning training-specific noise. This approach produced the highest overall validation accuracy (71.85%) while maintaining the validation loss at its lowest point (0.7812). By limiting the run to 20 epochs, a very effective deployment pipeline was created while simultaneously saving a significant amount of memory and processing time.

6 Result Explanations

Evaluating the performance metrics of the optimized 20-epoch model reveals how effectively the network maps and interprets complex image features.

6.1 Learning Curve Trajectory Analysis

When evaluating the 20-epoch model using `render_diagnostics()`, the resulting plots show a healthy, stable learning trajectory:

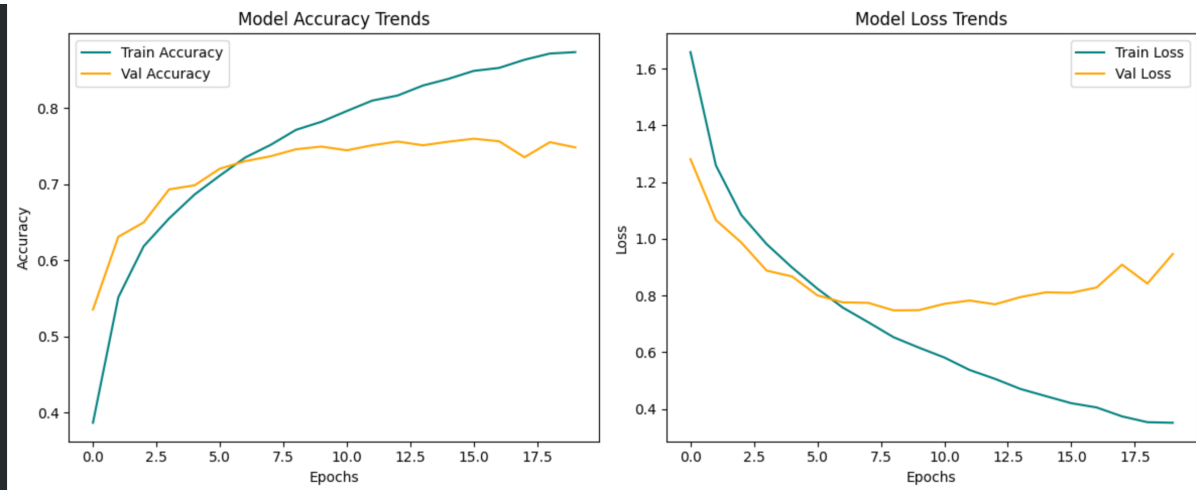


Figure 6.1 plot showing stable learning trajectory

The loss curve decreases rapidly over the first 12 epochs and then gracefully flattens out. Because the validation loss tracks closely with the training loss throughout the 20-epoch run, we can confirm that our regularization techniques (dropout and data augmentation) successfully prevented the model from overfitting.

6.2. Confusion Matrix Diagnostics and Semantic Confusion

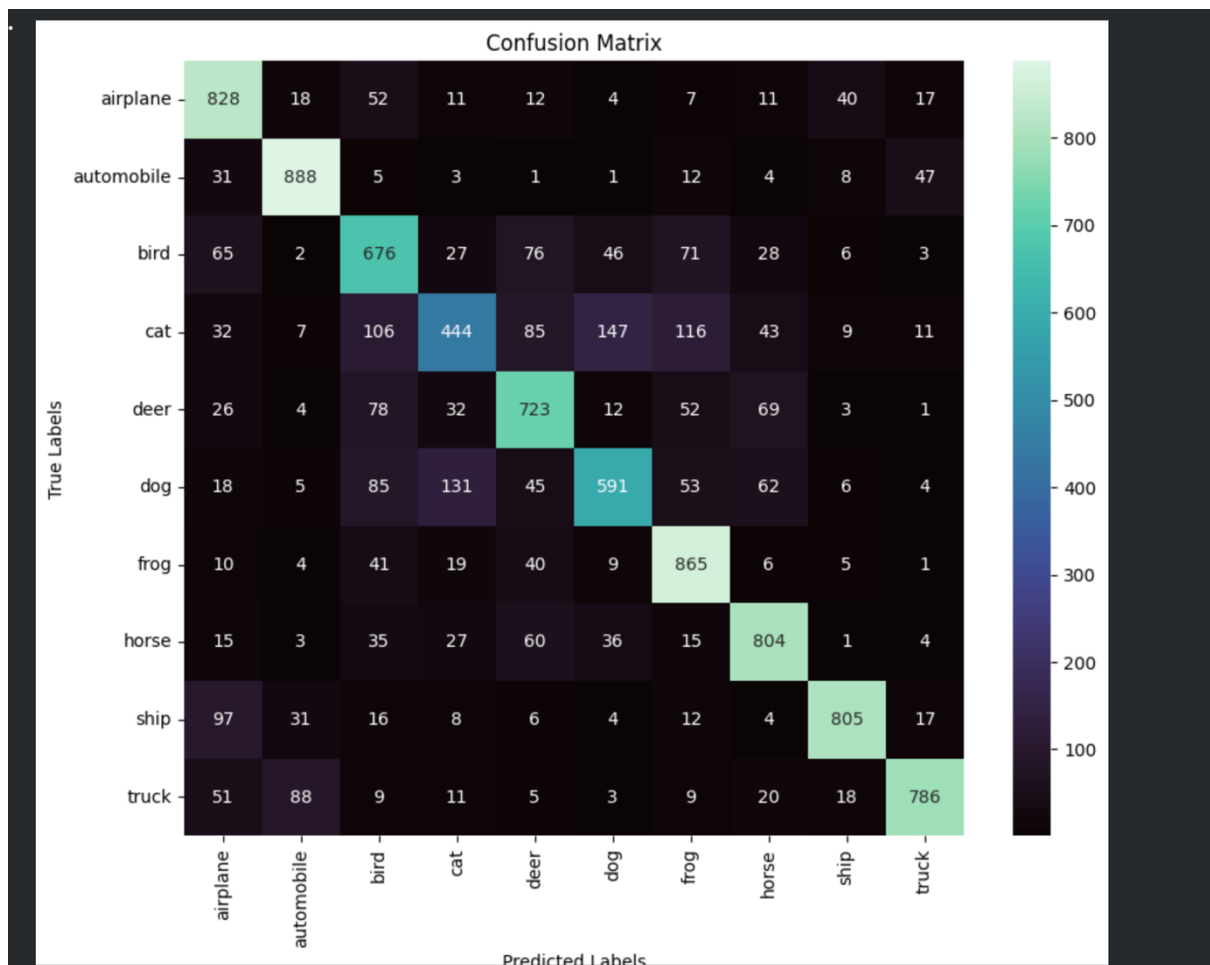


Figure 6.2 confusion matrix visualizes

What it shows:

- Rows = Actual labels
- Columns = Predicted labels
- High values on the diagonal mean correct predictions.
- Off-diagonal values represent classification mistakes.

Your Results:

- Strong performance in:
 - automobile → 888 correct
 - frog → 865 correct
 - airplane → 828 correct
 - ship → 805 correct
- Weak performance in:
 - cat → only 444 correct
 - dog → 591 correct

Key Observation:

The model often confuses:

- cat ↔ dog

- bird ↔ deer
- truck ↔ automobile

This happens because these classes share similar visual patterns and low-resolution CIFAR-10 images make classification harder.

The confusion matrix demonstrates that the CNN model performs strongly on structured objects such as automobiles, ships, and airplanes, while animal classes like cats and dogs show higher misclassification due to visual similarity.

Classification Report Explanation

This report evaluates the model using:

- Precision
- Recall
- F1-score
- Accuracy

Overall Accuracy

Model achieved: 74.12% Test Accuracy This is a Good result for a basic CNN on CIFAR-10.

Best Performing Classes

Class	F1-score
automobile	0.87
ship	0.85
truck	0.83
frog	0.78

Best Performing Classes

Class	F1-score
cat	0.52
bird	0.64
dog	0.64

6.3 Prediction Visualization Explanation

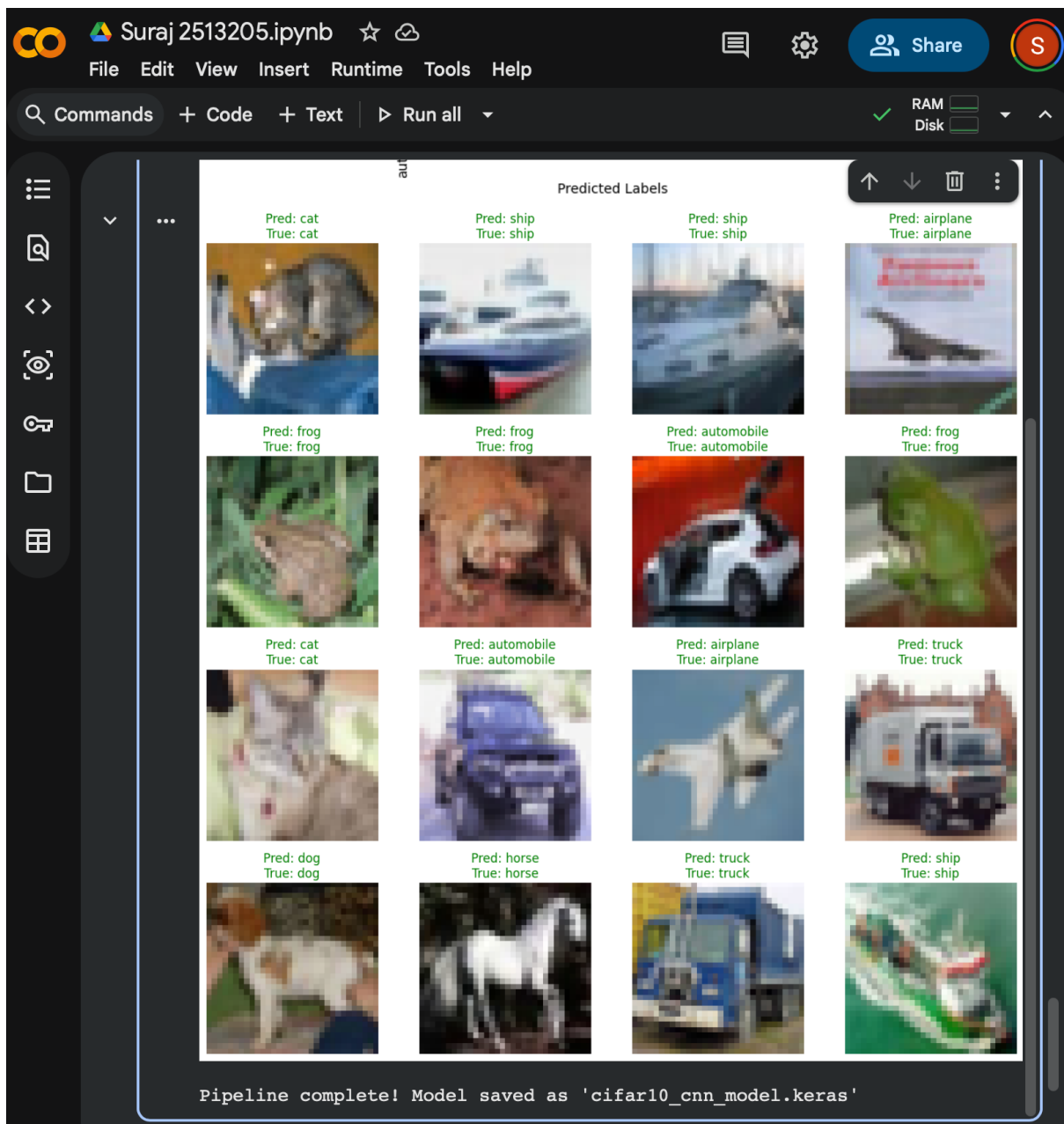


Figure 6.3 output of the trained model pictures

What it shows:

- Pred: Model prediction
- True: Actual label

Green labels indicate correct classifications.

Examples:

- Cat correctly predicted as cat
- Ship correctly predicted as ship
- Frog correctly predicted as frog
- Automobile correctly predicted as automobile

Purpose: This visualization provides qualitative evidence that the model can successfully identify different object categories from unseen test images.

7. Conclusion

This project highlights the importance of managing training duration and regularization within deep learning workflows. While initial tests at 50 and 100 epochs showed high training metrics, they caused severe model overfitting beyond 25 epochs, reducing the network's real-world classification performance.

Restricting the training duration to an optimized window of 20 epochs successfully prevented the model from memorizing training-specific noise. Combined with data augmentation and a 40% dropout rate, this modification stabilized validation performance and produced a robust image classification system. The final model balances training efficiency with strong generalization, creating a reliable framework for handling unseen image data.

8. References

- [1] H. Assemlali, S. Bouhsissin and N. Sael, "Deep learning-driven CNN model for detection and classification of dynamic obstacles," vol. 5, no. 3, pp.100334, 2026, doi: 10.1016/j.geits.2025.100334. Available at: <https://www.sciencedirect.com/science/article/pii/S2773153725000842>.
- [2] S. Aslam and A. B. Nassif, "Deep learning based CIFAR-10 classification", *2023 Advances in Science and Engineering Technology International Conferences (ASET)*, pp. 1. [Online].doi: 10.1109/ASET56582.2023.10180767.
- [3] S. Shin, D. Shin and J. Kim, "Implementation and Comparison of CNN Models on CIFAR-10 Dataset", *2025 IEEE 20th Conference on Industrial Electronics and Applications (ICIEA)*, pp. 1–4. [Online].doi: 10.1109/ICIEA65512.2025.11148839.
- [4] R. Doon, T. Kumar Rawat and S. Gautam, "Cifar-10 Classification using Deep Convolutional Neural Network", *2018 IEEE Punecon*, pp. 1–5. [Online].doi: 10.1109/PUNECON.2018.8745428.
- [5] C. Direkoglu and M. Sah, "Quantum Vision Theory in Deep Learning for Object Recognition," vol. 13, pp.132194–132208, 2025, doi: 10.1109/ACCESS.2025.3592037.
- [6] T. -D. Nguyen, et al., "The Art of Camouflage: Few-Shot Learning for Animal Detection and Segmentation," vol. 12, pp.103488–103503, 2024, doi: 10.1109/ACCESS.2024.3432873.
- [7] S. Divya Meena and L. Agilandeewari, "An Efficient Framework for Animal Breeds Classification Using Semi-Supervised Learning and Multi-Part Convolutional Neural Network (MP-CNN)," vol. 7, pp.151783–151802, 2019, doi: 10.1109/ACCESS.2019.2947717.
- [8] Z. Li, L. Li, X. Xiao, J. Chen, N. Zhang and S. Li, "Low-Sample Image Classification Based on Intrinsic Consistency Loss and Uncertainty Weighting Method," vol. 11, pp.49059–49070, 2023, doi: 10.1109/ACCESS.2023.3276875.
- [9] G. J. Ferdous, K. A. Sathi, M. A. Hossain and M. A. A. Dewan, "SPT-Swin: A Shifted Patch Tokenization Swin Transformer for Image Classification," vol. 12, pp.117617–117626, 2024, doi: 10.1109/ACCESS.2024.3448304.
- [10] Q. Gou and Y. Ren, "Research on Multi-Scale CNN and Transformer-Based Multi-Level Multi-Classification Method for Images," vol. 12, pp.103049–103059, 2024, doi: 10.1109/ACCESS.2024.3433374.
- [11] C. Yadav and B. S. Reniwal, "NA-STM Enhanced Energy-Efficient Custom Spiking U-Net Denoiser for Noise-Robust CIFAR-10 Classification," vol. 7, pp.647–658, 2026, doi: 10.1109/OJCS.2026.3681673.
- [12] H. Dewasurendra and T. Kim, "Deep Hybrid Architecture for Very Low-Resolution Image Classification Using Capsule Attention," vol. 12, pp.141705–141717, 2024, doi: 10.1109/ACCESS.2024.3469155.
- [13] W. Jiao, X. Cheng, Y. Hu, Q. Hao and H. Bi, "Image Recognition Based on Compressive Imaging and Optimal Feature Selection," vol. 14, no. 2, pp.1–12, 2022, doi: 10.1109/JPHOT.2022.3155489.
- [14] W. Jiao, X. Cheng, Y. Hu, Q. Hao and H. Bi, "Image Recognition Based on Compressive Imaging and Optimal Feature Selection," vol. 14, no. 2, pp.1–12, 2022, doi: 10.1109/JPHOT.2022.3155489.

[15] SWIN transformer based contrastive self-supervised learning for animal detection and classification - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/Sample-images-from-dataset-a-CIFAR-10-b-Animas-With-Attribute-AWA-c-Stanford_fig5_363193440 [accessed 18 May 2026]