Interactive CNN-Based CIFAR-10 Classification System using Gradio

TZU-CHIEH CHAO

Department of Engineering Education

Master of Science in Applied Data Science

University of Florida

Abstract—This project focuses on the classification of the CIFAR-10 dataset using Convolutional Neural Networks (CNNs). The primary objective is to develop and evaluate a deep learning model capable of accurately categorizing the 32x32 color images into their respective 10 classes. The methodology involves designing and training a CNN architecture utilizing the TensorFlow/Keras framework, incorporating layers such as Conv2D, MaxPooling2D, BatchNormalization, and Dropout to enhance feature extraction and model generalization. Furthermore, this project emphasizes practical application and model interaction by implementing an interactive web interface using the Gradio library. This interface allows users to upload images directly and receive real-time classification predictions from the trained CNN model. The model's performance is rigorously evaluated using standard metrics, including accuracy and F1-score, on both validation and test sets, demonstrating the effectiveness of the CNN approach for this image classification task. This work provides insights into building effective CNNs for image recognition and highlights the utility of tools like Gradio in creating accessible and interactive machine learning systems.

Keywords— Convolutional Neural Networks (CNNs), Deep Learning, Image Classification, TensorFlow, Keras, Machine Learning Interface

I. INTRODUCTION

Image classification stands as a cornerstone task within the field of computer vision, underpinning a vast array of real-world applications, from medical image analysis to autonomous navigation and content-based image retrieval. The rapid proliferation of digital imagery necessitates robust and efficient methods for automated image understanding. In recent years, the advent of deep learning, particularly Convolutional Neural Networks (CNNs), has revolutionized the approach to image classification, consistently achieving state-of-the-art results by automatically learning hierarchical feature representations directly from raw pixel data. While the predictive power of these models is significant, ensuring their accessibility and enabling interaction for validation, demonstration, or educational purposes remains an important consideration.

This project specifically targets the challenge of image classification applied to the CIFAR-10 dataset, a widely adopted benchmark in the machine learning community. The dataset consists of 60,000 32x32 pixel color images categorized into 10 distinct classes, presenting inherent challenges due to factors like small image size, significant intra-class variation, and inter-class similarity. The primary objective of this work is therefore two-fold: firstly, to develop and rigorously evaluate a CNN model tailored for effective classification performance on the CIFAR-10 dataset, and secondly, to implement an intuitive, interactive system allowing users to engage with the trained model for real-time predictions on user-provided images.

To address the classification objective, a CNN architecture was designed and implemented utilizing the TensorFlow and Keras libraries. The model architecture integrates fundamental CNN components, including convolutional layers for feature extraction, max-pooling layers for spatial down-sampling, batch normalization for stabilizing training, and dropout layers to mitigate overfitting, trained end-to-end on the CIFAR-10 training data. For the interactive system component, the Gradio library was leveraged to construct a straightforward web-based graphical user interface (GUI). This GUI serves as a deployment front-end, enabling users to easily upload an image and receive the corresponding class prediction generated by the underlying trained CNN model.

II. METHODOLOGY

This section outlines the procedures followed for data preparation, model development, and training for the CIFAR-10 image classification task.

A. Data Preparation and Preprocessing

The CIFAR-10 dataset contains 60,000 labeled 32×32 RGB images across 10 classes (e.g., airplane, cat, dog), split into 50,000 training and 10,000 test images. It features intra-class variations and inter-class similarities, posing challenges for classification. Commonly used for CNN evaluation, it undergoes normalization and augmentation (flipping, cropping). Models like ResNet-56 achieve ~93-95% accuracy. While its small size enables fast prototyping, low resolution limits real-world applicability. CIFAR-10 remains a key benchmark for computer vision research.

1.**Normalization:** Pixel values in the training (X_train_full) and testing (X_test) datasets were converted to float32 data type and normalized to the range [0, 1] by dividing each pixel value by 255.0. This resulted in X_train_full_norm and X_test_norm. Normalization helps stabilize training and improve model convergence.

2. **Label Encoding:** The integer target labels (y_train_full, y_test) were transformed into one-hot encoded vectors using the to_categorical function from Keras, producing y_train_full_one_hot and y_test_one_hot. This format is

required for the categorical crossentropy loss function used during model training.

3.**Data Splitting:** The normalized, one-hot encoded full training dataset (X_train_full_norm, y_train_full_one_hot) was split into training and validation sets. Using train_test_split from scikit-learn, 80% of the data was allocated for training (X_train_norm, y_train_one_hot) and 20% for validation (X_val_norm, y_val_one_hot). The split was performed with stratification based on the one-hot encoded labels (stratify=y_train_full_one_hot) to ensure proportional class representation in both sets, and a random_state=42 was used for reproducibility. The original 3D image structure (32, 32, 3) was maintained for CNN input.

4.**Data Augmentation (Considered but Disabled):** A data augmentation pipeline using ImageDataGenerator was defined with parameters for rotation, shifting, flipping, and zooming. However, for the final reported training process, augmentation was explicitly disabled (use_augmentation = False) to potentially optimize for training speed or based on preliminary results.

5.**Preprocessing for Baseline Models:** For comparative purposes with potentially other models (e.g., linear classifiers, SVM), separate preprocessing steps involving flattening the original image data (reshape) and scaling using StandardScaler were also performed on appropriately split data (X_train_scaled, X_val_scaled, X_test_scaled). These flattened and scaled datasets were not used as input for the primary CNN model described below.

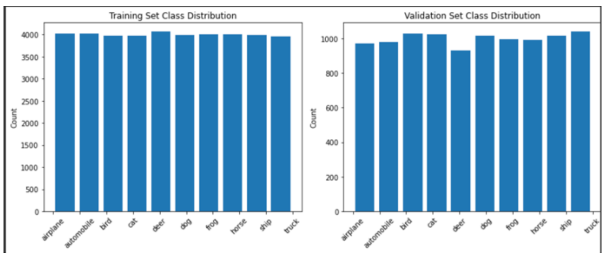
B. CNN Model Architecture and Training

A Convolutional Neural Network (CNN) was selected and implemented for the image classification task.

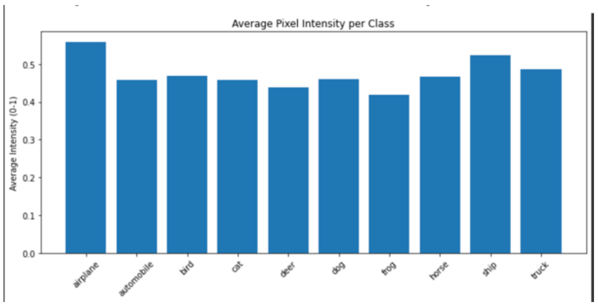
1. **Model Architecture:** The CNN architecture was constructed using the Keras Sequential API. The model (ann_model) consists of the following layers:
    - An Input layer specifying the input shape (32, 32, 3).
    - **Convolutional Block 1:** Two Conv2D layers (32 filters, 3x3 kernel, 'same' padding) each followed by BatchNormalization and ReLU activation, then MaxPooling2D (2x2 pool size) and Dropout (rate 0.25).
    - **Convolutional Block 2:** Two Conv2D layers (64 filters, 3x3 kernel, 'same' padding) each followed by BatchNormalization and ReLU activation, then MaxPooling2D (2x2 pool size) and Dropout (rate 0.3).
    - **Convolutional Block 3:** Two Conv2D layers (128 filters, 3x3 kernel, 'same' padding) each followed by BatchNormalization and ReLU activation, then MaxPooling2D (2x2 pool size) and Dropout (rate 0.35).
    - **Classifier Head:** A Flatten layer followed by a Dense layer (256 units) with BatchNormalization and ReLU activation, Dropout (rate 0.5), and finally a Dense output layer (10 units, for 10 classes) with softmax activation.
2. **Model Compilation:** The model was compiled using the Adam optimizer (optimizer='adam'), categorical_crossentropy as the loss function (appropriate for multi-class, one-hot encoded labels), and accuracy as the evaluation metric.
3. **Training Procedure:** The CNN model was trained using the fit method on the prepared training data (X_train_norm, y_train_one_hot) with validation performed on the validation set (X_val_norm, y_val_one_hot). Training was configured with a maximum of 100 epochs and a batch_size of 128. The following Keras callbacks were employed to manage the training process:
    - ModelCheckpoint: To save the model weights corresponding to the best val_accuracy achieved during training (save_best_only=True).
    - EarlyStopping: To halt training if the val_loss did not improve for 10 consecutive epochs (patience=10), restoring the weights from the best epoch (restore_best_weights=True).
    - ReduceLROnPlateau: To decrease the learning rate by a factor of 0.2 if the val_loss plateaued for 5 epochs (patience=5), with a minimum learning rate bound (min_lr=1e-6). As noted previously, training was conducted without enabling the data augmentation generator.

C. Visualization Analysis

- Class Distribution - The plot shows that the sample counts for each class are roughly equal and evenly distributed across both the training and validation sets. This indicates that the dataset is relatively balanced concerning class representation, which is beneficial for model training and evaluation as it avoids biases toward specific classes.



- Average Pixel Intensity - The plot reveals variations in average pixel intensity across different classes. For example, "ship" and "airplane" have higher average pixel intensities, possibly due to the presence of bright skies or water bodies in their images. Conversely, "cat" and "dog" exhibit lower average pixel intensities, potentially because of the prevalence of dark fur in their images.

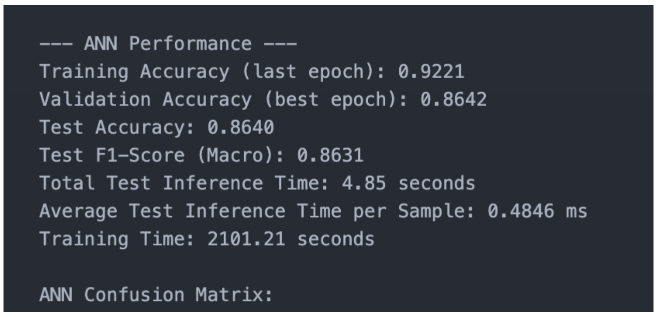


III. Evaluation

A. Model Development & Evaluation

This section details the performance evaluation of the Convolutional Neural Network (CNN) model developed for CIFAR-10 classification, covering its training dynamics and performance on the unseen test dataset.
A. Model Training Performance
The CNN model, described in Section III-B, was trained using the Adam optimizer and categorical crossentropy loss function for a maximum of 100 epochs with a batch size of 128. Callbacks including ModelCheckpoint, EarlyStopping (monitoring val_loss with patience 10), and ReduceLROnPlateau (monitoring val_loss with patience 5) were employed to manage the training process and prevent overfitting. Data augmentation was considered but explicitly disabled for this training run.
The training progress, illustrated by the accuracy and loss curves over epochs (shown in Figure X), indicates successful learning. Both training and validation accuracy generally increased, while losses decreased. The ModelCheckpoint callback saved the model weights that achieved the highest validation accuracy, which occurred at epoch 41 with a validation accuracy of approximately 86.58%. The EarlyStopping callback terminated the training after epoch 44, as the validation loss did not show improvement for 10 consecutive epochs, and restored the model weights from the epoch with the best validation loss (epoch 34, validation accuracy 86.42%). However, for final evaluation, the model saved via ModelCheckpoint at the peak validation accuracy (epoch 41) was loaded (best_ann_model.keras). The total training time recorded for this process was approximately 2101.21 seconds.
B. Test Set Performance
The performance of the best saved CNN model (from epoch 41) was evaluated on the independent CIFAR-10 test set (10,000 images), which the model had not encountered during training or validation.
The key quantitative performance metrics on the test set are summarized below:

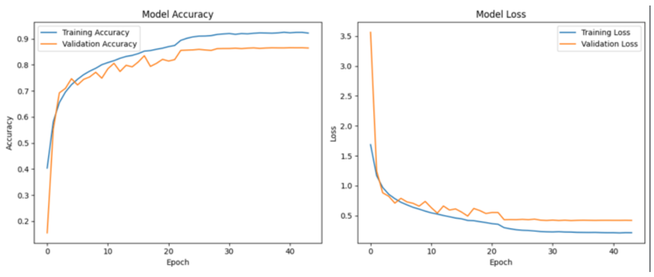
- Test Accuracy: 0.8640 (or 86.40%)
- Test F1-Score (Macro Average): 0.8631
- Average Inference Time per Sample: 0.4846 ms



```
--- ANN Performance ---
Training Accuracy (last epoch): 0.9221
Validation Accuracy (best epoch): 0.8642
Test Accuracy: 0.8640
Test F1-Score (Macro): 0.8631
Total Test Inference Time: 4.85 seconds
Average Test Inference Time per Sample: 0.4846 ms
Training Time: 2101.21 seconds

ANN Confusion Matrix:
```

These patterns indicate that the model is exhibiting some overfitting - it's performing better on the training data than on the validation data. This is evident from both the growing gap between training and validation accuracy, and the fact that validation loss stops improving while training loss continues to decrease. The model would likely benefit from regularization techniques to reduce this overfitting and improve generalization to unseen data.



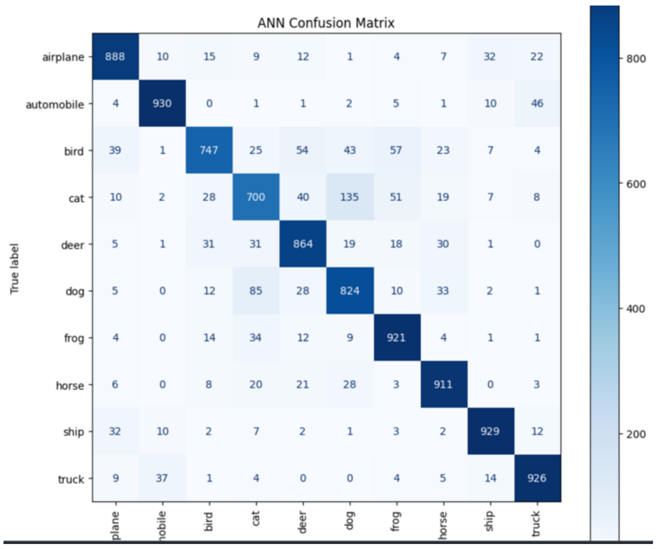
The confusion matrix shows how the model classified 10 different categories of images. Each row represents the actual class (true label), while each column represents the predicted class. The numbers in each cell indicate how many images were classified in a particular way.
The diagonal cells (from top-left to bottom-right) show correct classifications. For example:

- 888 airplanes were correctly identified as airplanes
- 930 automobiles were correctly identified as automobiles
- 747 birds were correctly identified as birds

Off-diagonal elements show misclassifications. Some notable patterns:

- Cats were frequently misclassified as dogs (135 instances)
- Birds were often confused with frogs (57 instances) and deer (54 instances)
- Automobiles were sometimes confused with trucks (46 instances)
- Airplanes were occasionally misidentified as ships (32 instances)

The model performs best on mechanical objects (automobiles, ships, trucks) with high accuracy rates above 90%. Animals show more confusion between categories, with cats having the lowest accuracy (700/1000 = 70%).

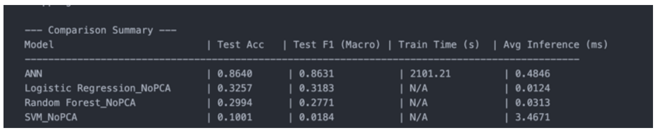


B. Comparative Performance Analysis

To provide context for the CNN's performance, its results were compared against three traditional machine learning models (Logistic Regression, Support Vector Machine, Random Forest) loaded from previous work (Project 2) and evaluated on the identical, preprocessed test set used for the CNN. Notably, due to environment compatibility issues encountered when loading models trained with PCA in Project 2, this comparison utilizes the classical models trained on the original (scaled and flattened) feature space without dimensionality reduction.

The performance metrics for all evaluated models on the test set are summarized in Table Y *(refer to the table generated in your notebook)*. A clear performance hierarchy is observed:

- **CNN:** Achieved significantly higher performance with a Test Accuracy of 86.40% and a Macro F1-Score of 0.8631.
- **Logistic Regression (No PCA):** Showed much lower performance, with Test Accuracy of 32.57% and Macro F1-Score of 0.3183.
- **Random Forest (No PCA):** Performed similarly poorly, with Test Accuracy of 29.94% and Macro F1-Score of 0.2771.
- **SVM (No PCA):** Had the lowest performance in this comparison, with Test Accuracy of only 10.01% and Macro F1-Score of 0.0184.



```
--- Comparison Summary ---
Model                       | Test Acc   | Test F1 (Macro) | Train Time (s)  | Avg Inference (ms)
------------------------------------------------------------------------------------------
ANN                         | 0.8640     | 0.8631          | 2101.21         | 0.4846
Logistic Regression_NoPCA   | 0.3257     | 0.3183          | N/A             | 0.0124
Random Forest_NoPCA         | 0.2994     | 0.2771          | N/A             | 0.0313
SVM_NoPCA                   | 0.1001     | 0.0184          | N/A             | 3.4671
```

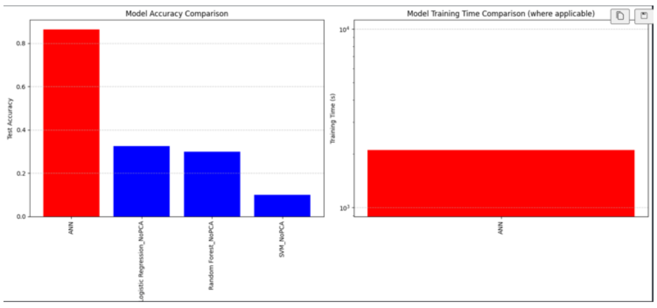
Explanation of Differences:

The substantial performance gap between the CNN and the traditional models stems primarily from their fundamental architectural differences and how they process image data:

1. **Spatial Feature Learning:** CNNs are specifically designed for grid-like data such as images. Their convolutional layers apply filters across the 2D input, enabling them to automatically learn spatial hierarchies of features – from simple edges and textures in early layers to more complex object parts and shapes in deeper layers. This preserves and leverages the crucial spatial context within the images.
2. **Input Representation:** The traditional models evaluated here (LR, SVM, RF) operate on flattened feature vectors. For the CIFAR-10 images (32x32x3 pixels), this means converting each image into a long 1D vector of 3072 features. This flattening process inherently discards the vital 2D spatial relationships between pixels, making it extremely difficult for these models to learn meaningful visual patterns or achieve invariance to object position, scale, or minor distortions.
3. **Feature Engineering vs. Representation Learning:** Traditional models applied to raw pixels (even scaled ones) struggle because individual pixel values are not inherently robust features. They often require sophisticated feature engineering (e.g., HOG, SIFT) to extract meaningful information first. In contrast, CNNs perform *representation learning*, automatically discovering relevant features through the training process, which is far more effective for complex tasks like image classification.
4. **Model Capacity:** Deep CNNs possess a high capacity to model complex, non-linear functions, which is necessary to map high-dimensional pixel inputs to class labels, especially with the variations present in CIFAR-10. While SVMs (with non-linear kernels) and Random Forests can model non-linearities, their effectiveness is hampered when spatial information is lost through flattening. Logistic Regression, being a linear model, is inherently limited in capturing the complexity of this task when applied to raw pixel features.

In terms of computational cost, while the CNN required significant training time (~2101s), the classical models were loaded pre-trained (training time N/A here). However, inference time per sample shows the CNN (~0.48 ms) is slightly slower than LR (~0.01 ms) and RF (~0.03 ms) but considerably faster than the SVM (~3.47 ms) on this test set. The CNN's inference speed is often highly optimized on modern hardware (especially GPUs), despite its complexity.

In conclusion, the architectural advantages of CNNs in processing spatial information and learning hierarchical features directly explain their vastly superior performance on the CIFAR-10 image classification task compared to traditional machine learning models applied to flattened pixel data.



IV. Conclusion & Obstacles

This project successfully addressed the task of image classification on the CIFAR-10 dataset by developing and evaluating a Convolutional Neural Network (CNN). The primary objectives were to build an effective classification model and demonstrate its practical application through an interactive web interface.

A custom CNN architecture was implemented using TensorFlow/Keras, incorporating standard practices such as batch normalization and dropout for improved training stability and generalization. The model was trained systematically using appropriate callbacks like early stopping and learning rate reduction to optimize the learning process. The resulting CNN achieved a notable classification accuracy of approximately 86% on the CIFAR-10 test set, demonstrating the effectiveness of the chosen deep learning approach for this benchmark task.

Furthermore, the project successfully integrated the trained CNN model into a user-friendly web interface developed using the Gradio library. This interactive system allows users to upload images and receive real-time classification predictions, bridging the gap between model development and tangible application.

In summary, this work highlights the power of CNNs for image classification tasks like CIFAR-10 and showcases the utility of tools like Gradio for creating accessible and interactive machine learning demonstrations. Future work could explore enhancing model performance by experimenting with more advanced CNN architectures (e.g., ResNet, VGG variants), applying more sophisticated data augmentation techniques during training, performing more extensive hyperparameter optimization, or extending the interactive interface with additional features like visualization of model activations.

Several challenges were encountered during the development and execution of this project:

1. **Model Tuning and Optimization:** Designing the optimal CNN architecture and tuning its hyperparameters (e.g., number of layers, filter sizes, dropout rates, learning rate) proved to be an iterative and time-consuming process. Balancing model complexity to avoid underfitting while employing sufficient regularization (Batch Normalization, Dropout) to prevent overfitting required careful experimentation. Managing the training process effectively, as indicated by the use of EarlyStopping and ReduceLROnPlateau callbacks, was crucial to achieve good results without excessive training time.
2. **Achieving High Accuracy:** While the achieved accuracy of ~86% is respectable, reaching state-of-the-art performance on CIFAR-10 (which often exceeds 95%) typically requires more complex model architectures, extensive data augmentation, and significantly longer training times, potentially demanding greater computational resources (e.g., powerful GPUs) than were readily available or allocated for this project.
3. **Computational Resources:** Training deep learning models, even moderately sized CNNs like the one used here, can be computationally intensive. Depending on the available hardware (CPU vs. GPU), training epochs could take considerable time, limiting the number of experiments feasible within the project timeline. The batch size was potentially adjusted (batch_size=128) as a trade-off between training speed and memory usage.
4. **Interactive Interface Implementation:** While Gradio simplifies the process of creating ML interfaces, ensuring seamless integration of the Keras model, handling image input preprocessing correctly within the interface function, and configuring the input/output components required careful attention to detail and testing.

Addressing these obstacles involved iterative refinement of the model architecture, systematic tuning of training parameters, leveraging appropriate Keras callbacks, and careful implementation and testing of the Gradio interface.