Comparative Study of CIFAR-10 Image Classification using ANN, CNN, and Transfer Learning Models

Rhaiem Chayma

Contents

List of Figures

1 Introduction

The CIFAR-10 dataset serves as a benchmark for image classification tasks, containing 60,000 32x32 color images categorized into 10 classes. In this study, we aim to evaluate the performance of three different models: Artificial Neural Network (ANN), Convolutional Neural Network (CNN), and a transfer learning model, in classifying CIFAR-10 images. Image classification, crucial in various domains, relies on neural networks to discern intricate patterns within images.

2 Dataset Description

The CIFAR-10 dataset consists of 60,000 32×32 color images in 10 different classes, with 6,000 images per class. Each image is labeled with one of the following categories:

Class Label	Class Name
0	Airplane
1	Automobile
2	Bird
3	Cat
4	Deer
5	Dog
6	Frog
7	Horse
8	Ship
9	Truck

Table 1: CIFAR-10 Class Labels

The dataset is commonly used for training and testing computer vision algorithms.

2.1 Sample Data

Let's showcase some sample images from the CIFAR-10 dataset:

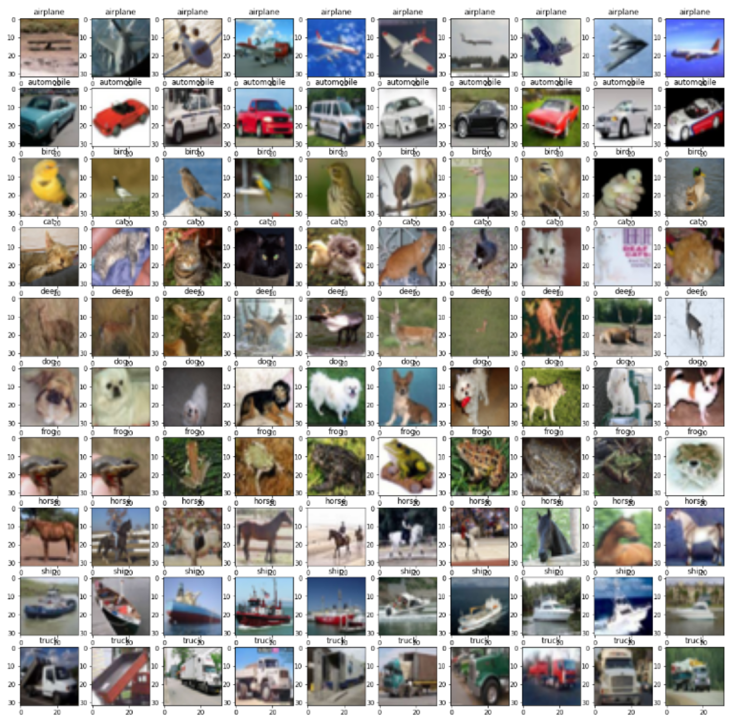


Figure 1: Sample Images from CIFAR-10 Dataset

2.2 Class Distribution

The classes in both the training and test datasets are equally distributed. This ensures that the model is trained and evaluated on a balanced dataset, preventing biases towards any particular class.

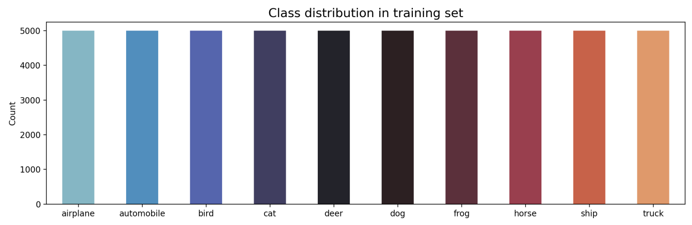


Figure 2: Class Distribution in Training Set

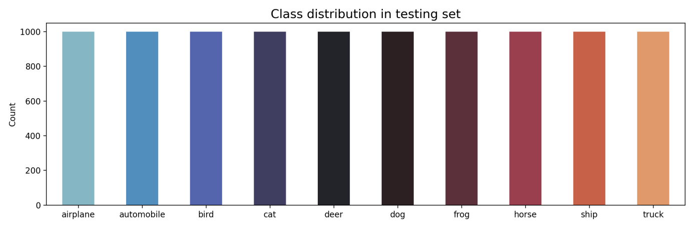


Figure 3: Class Distribution in Testing Set

2.3 Data Preparation

Working with Neural Network and Deep Learning based systems, it is generally a good idea to start with theStandardization of the data.

Normalization

For normalizing the pixel data (Image) we can simply divide the whole pixel values with 255 since pixel values ranges from 0-255. So if we divide them with 255 we automatically normalize the data between 0-1.

One Hot Encoding

CIFAR 10 has 10 categories, in general we should label the categorical data using the one hot encoding.

3 Data Augmentation

Data augmentation is performed to artificially increase the size of the training dataset and improve the model's generalization and robustness. Common augmentation techniques include rotation, translation, scaling, flipping, and zooming of images.

- Rotation Range: 20 degrees
- Width Shift Range: 10
- Height Shift Range: 10
- Zoom Range: 10
- Horizontal Flip: Enabled
- Vertical Flip: Disabled

These parameters are applied during the training process to generate augmented images, providing variations in rotation, shifting, zooming, and flipping to improve the model's ability to generalize to unseen data.

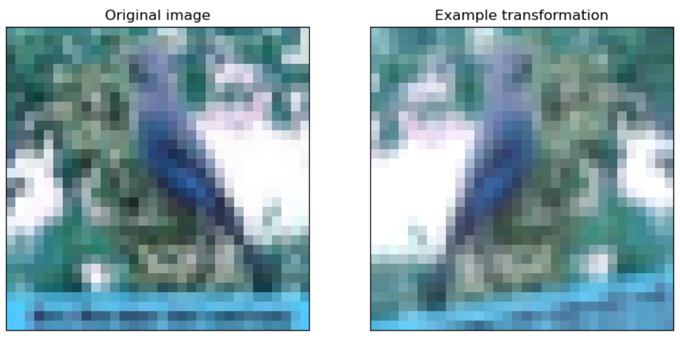


Figure 4: Example of Data Augmentation

4 Artificial Neural Network (ANN)

4.1 Model Architecture

The architecture of the Artificial Neural Network (ANN) model used for classifying CIFAR-10 images is as follows:

```
# Define the ANN model
model = models.Sequential([
    layers.Flatten(input_shape=(32, 32, 3)),
    layers.Dense(128, activation='relu'),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax')
])

# Compile the model
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

4.1.1 Hyperparameters

4.1.2 Choice of Hyperparameters

The following hyperparameters were chosen based on empirical evidence and best practices in deep learning:

- Number of hidden layers: Two hidden layers were chosen to allow the model to capture complex patterns in the data without overfitting.
- Number of neurons per hidden layer: 128 neurons were chosen for the first hidden layer and 64 neurons for the second hidden layer to strike a balance between model complexity and computational efficiency.

Hyperparameter	Value
Number of hidden layers	2
Number of neurons per hidden layer	128 for the first hidden layer, 64 for the second hidden layer
Activation functions	ReLU for all hidden layers, softmax for the output layer
Dropout rate	0.0 (no dropout used in this model)

Table 2: Hyperparameters of the ANN Model

- Activation functions: ReLU activation function was chosen for all hidden layers due to its simplicity and effectiveness in preventing vanishing gradients. Softmax activation function was chosen for the output layer to output probability distributions over the classes.
- Dropout rate: A dropout rate of 0.0 (no dropout) was chosen for this model to simplify the architecture and prevent underfitting on the CIFAR10 dataset, which contains a relatively small number of images.

4.2 Results

The model was trained for 100 epochs with early stopping set to 3 (the model stopped at the 18th epoch). The accuracy achieved on the test set was 0.477, which is relatively low.

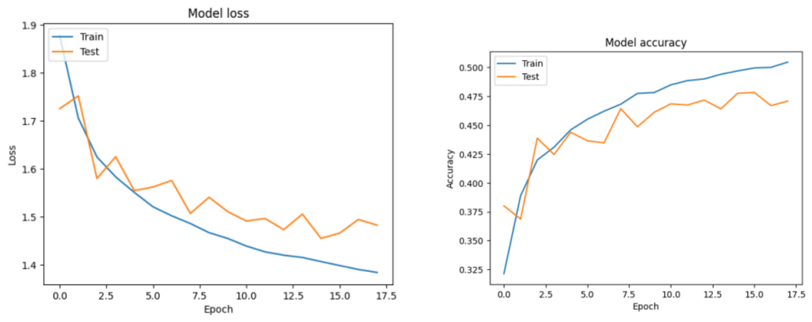


Figure 5: Model Loss and Accuracy of ANN

4.3 Convolutional Neural Network (CNN)

Defining the Model (Convolutional Neural Network)

The 2D convolution is a fairly simple operation at heart: you start with a kernel, which is simply a small matrix of weights. This kernel “slides” over the 2D input data, performing an element-wise multiplication with the part of the input it is currently on, and then summing up the results into a single output pixel.

4.4 Model Architecture

The network architecture comprises a sequence of convolutional, pooling, dropout, and dense layers. This design aims to learn hierarchical features from CIFAR-10 images while employing regularization techniques to prevent overfitting.

The network initiates with a pair of Conv2D layers, each featuring 32 filters sized 3x3. Subsequently, a Batch Normalization layer is applied to accelerate training and introduce regularization. The subsequent MaxPooling2D layer decreases spatial dimensions, enhancing translation invariance and reducing computational complexity. A Dropout layer follows, randomly deactivating a fraction of input units (initially set to 0.2) during training to mitigate overfitting.

This pattern repeats thrice, with each repetition doubling the number of filters in the Conv2D layers (32, 64, 128, and 256). Simultaneously, the dropout rate increases from 0.2 to 0.5. Post convolutional and pooling layers, a Flatten layer converts 2D outputs to a 1D vector. Lastly, a Dense layer with 10 units, each representing a CIFAR-10 class, and a softmax activation function concludes the classification process.

This architecture leverages the strengths of deep CNNs while employing regularization techniques like L2 regularization, Dropout, and Batch Normalization to mitigate overfitting. Inspired by VGG16, this model prioritizes efficiency and simplicity over advanced features.

5 Training the CNN Model

The neural network model is trained using a batch size of 64, with a maximum of 250 epochs or until the early stopping condition is met. Evaluation on the validation data occurs after each epoch.

5.1 Hyperparameters

Below is a summary table of the hyperparameters used in training the CNN model:

Hyperparameter	Value
Batch Size	64
Maximum Epochs	300
Initial Learning Rate	0.005
Dropout Rates	0.2 - 0.5

Table 3: Hyperparameters of the CNN Model

5.2 Training the CNN Model

Now, let's train the neural network model. The training utilizes a batch size of 64 and runs for a maximum of 300 epochs. However, we've incorporated early stopping, which halts training if there's no improvement in the validation loss for 40 epochs. During training, the model's performance is evaluated on the validation data after each epoch.

Hyperparameter	**Value**
Batch Size	64
Epochs	300
Optimizer	Adam
Learning Rate	0.0005

Table 4: Hyperparameters for Training the CNN Model

5.2.1 Callback Functions

To enhance the training process, we've incorporated two callback functions:

1. **ReduceLROnPlateau:** This callback reduces the learning rate by half (factor=0.5) if the validation loss does not improve for 10 consecutive epochs. Adjusting the learning rate dynamically helps the model converge closer to the global minimum of the loss function, particularly when progress has plateaued.

2. **EarlyStopping:** This callback monitors the validation loss and halts the training process if there hasn't been any improvement for 40 epochs. By preventing unnecessary resource consumption and time wastage, this callback ensures that the model concludes with the optimal configuration.

5.2.2 Code Snippet

```
# Set the batch size for training
batch_size = 64

# Set the maximum number of epochs for training
epochs = 300

# Define the optimizer (Adam)
optimizer = Adam(learning_rate=0.0005)

# Compile the model with the defined optimizer, loss function, and metrics
model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['accuracy'])

# Add ReduceLROnPlateau callback
```

```
reduce_lr = ReduceLROnPlateau(monitor='val_loss',
factor=0.5, patience=10, min_lr=0.00001)

# Add EarlyStopping callback
early_stopping = EarlyStopping(monitor='val_loss', patience=40,
                               restore_best_weights=True, verbose=1)

# Fit the model on the training data, using the defined
batch size and number of epochs
# The validation data is used to evaluate the model's performance during training
# stopping training early if no improvement is observed
model.fit(data_generator.flow(X_train, y_train, batch_size=batch_size),
          epochs=epochs,
          validation_data=(X_valid, y_valid),
          callbacks=[reduce_lr, early_stopping],
          verbose=2)
```

5.3 Results

The model was trained for 100 epochs with early stopping set to 3 (the model stopped at the 18th epoch). The accuracy achieved on the test set was 0.477, which is relatively low.

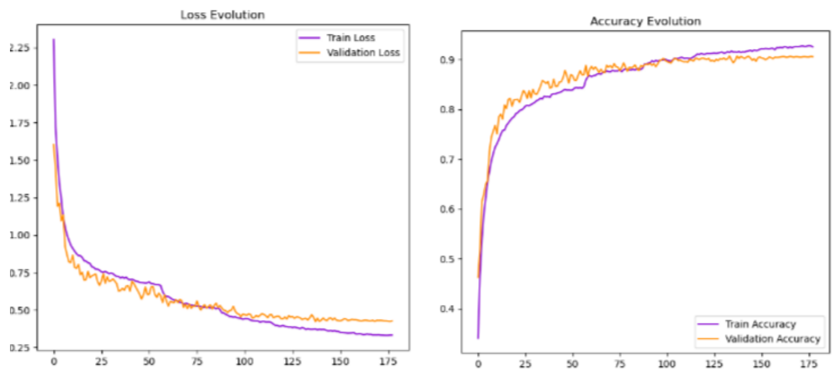


Figure 6: Model Loss and Accuracy of CNN

Based on the visualizations above, it's evident that the model is performing well without signs of overfitting. This conclusion is supported by the close alignment of training and validation accuracy and loss values throughout the training process. The gap between training and validation accuracy remains minimal, indicating that the model generalizes well to unseen data. Similarly, the model's loss on validation data closely follows the training loss, reinforcing the assertion of good generalization. Therefore, the model appears to be well regularized and not overfitting to the training data.

6 Transfer Learning

Transfer learning is a popular technique in deep learning where a pre-trained model is used as the starting point for a new task. In this experiment, we explore the use of two pre-trained models, VGG19 and ResNet-50, for classifying images in the CIFAR-10 dataset. We aim to compare the performance of these models and analyze their effectiveness in image classification.

6.1 Model Selection

We selected VGG19 and ResNet-50 for this experiment due to their popularity and proven performance in various image classification tasks. VGG19 consists of 19 layers, while ResNet-50 incorporates 50 layers of residual units, allowing for deeper architectures without degradation in accuracy.

6.2 Data Preparation

We divided the CIFAR-10 dataset into training, validation, and test sets. The training set comprises 35,000 samples, the validation set contains 15,000 samples, and the test set contains 10,000 samples.

6.3 Data Augmentation

We applied data augmentation techniques using the `ImageDataGenerator` class from Keras. Augmentation includes rotation, horizontal flipping, and zooming, which helps in increasing the diversity of the training dataset.

6.4 Model Building: VGG19

We start by building the VGG19 model, which consists of a base convolutional neural network (CNN) followed by fully connected layers. The base model is loaded with pre-trained weights from the ImageNet dataset.

```
base_model_1 = VGG19(include_top=False, weights='imagenet',
input_shape=(32, 32, 3), classes=y_train.shape[1])
```

Adding Dense Layers to VGG19

We add dense layers on top of the VGG19 base model to perform classification. The output of the base model is flattened before passing through dense layers with ReLU activation functions. The final layer uses softmax activation for multi-class classification.

```
    model_1 = Sequential()
model_1.add(base_model_1)
model_1.add(Flatten())
```

```
model_1.add(Dense(1024, activation='relu', input_dim=512))
model_1.add(Dense(512, activation='relu'))
model_1.add(Dense(256, activation='relu'))
model_1.add(Dense(128, activation='relu'))
model_1.add(Dense(10, activation='softmax'))
```

Compiling and Training VGG19 Model

We compile the VGG19 model using the SGD optimizer and categorical cross-entropy loss function. Then, we train the model using the fit_generator function, which allows us to use data generators for augmentation.

6.5 Model Building: ResNet-50

Similarly to VGG19,Similarly, we build the ResNet-50 model by loading the pre-trained weights and adding dense layers for classification on top of the base model.

```
    base_model_2 = ResNet50(include_top=False, weights='imagenet',
    input_shape=(32, 32, 3), classes=y_train.shape[1])
```

Adding Dense Layers to ResNet-50

We add dense layers on top of the ResNet-50 base model for classification, similar to the process followed for VGG19.

```
    model_2 = Sequential()
model_2.add(base_model_2)
model_2.add(Flatten())
model_2.add(Dense(1024, activation='relu', input_dim=512))
model_2.add(Dense(512, activation='relu'))
model_2.add(Dense(256, activation='relu'))
model_2.add(Dense(128, activation='relu'))
model_2.add(Dense(10, activation='softmax'))
```

Compiling and Training ResNet-50 Model

```
    model_2.compile(optimizer=adam, loss='categorical_crossentropy',
    metrics=['accuracy'])
model_2.fit_generator(train_generator.flow(x_train, y_train, batch_size=batch_size),
                      epochs=epochs,
                      steps_per_epoch=x_train.shape[0] // batch_size,
                      validation_data=val_generator.flow(x_val, y_val,
                      batch_size=batch_size),
                      validation_steps=250,
                      callbacks=[lrr], verbose=1)
```

6.6 Results

First let's start the evaluation by taking a look to a side-by-side comparison of the 2 models we've used.

VGG19 Architecture	**ResNet-50 Architecture**
Input Shape (32, 32, 3)	**Input Shape** (32, 32, 3)
Base Convolutional Layers	**Base Convolutional Layers**
Conv2D(64, (3, 3), activation='relu', padding='same', input_shape=(32, 32, 3))	Conv2D(64, (7, 7), strides=(2, 2), padding='same', input_shape=(32, 32, 3))
Conv2D(64, (3, 3), activation='relu', padding='same')	MaxPooling2D((3, 3), strides=(2, 2), padding='same')
MaxPooling2D((2, 2), strides=(2, 2))	Conv2D(64, (1, 1), strides=(1, 1), padding='same')
Fully Connected Layers	**Fully Connected Layers**
Dense(1024, activation='relu', input_dim=512)	Dense(1024, activation='relu', input_dim=2048)
Dense(512, activation='relu')	Dense(512, activation='relu')
Dense(256, activation='relu')	Dense(256, activation='relu')
Dense(128, activation='relu')	Dense(128, activation='relu')
Dense(10, activation='softmax')	Dense(10, activation='softmax')

Table 5: Comparison of VGG19 and ResNet-50 Architectures

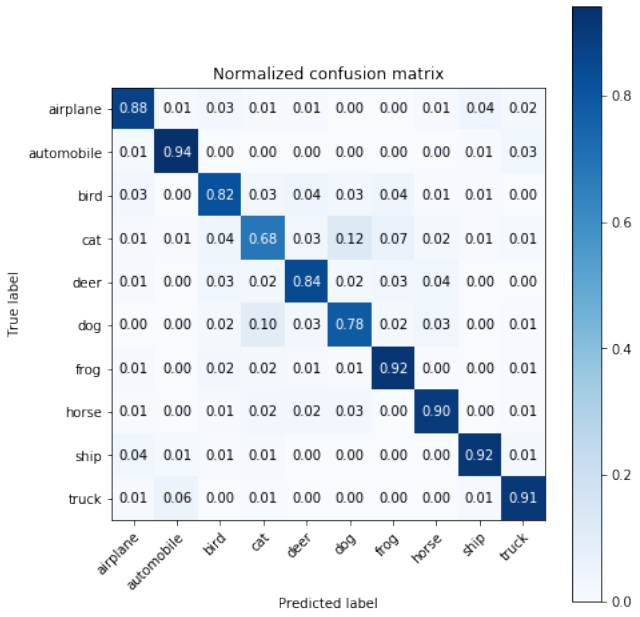
For the two models, we'll be using Normalized confusion matrices that provide a visual representation of the classification performance of each model.

VGG19

After training VGG19 for 50 epochs, we achieved a validation accuracy of 82.16%. The normalized confusion matrix for VGG19 on CIFAR-10 is shown below

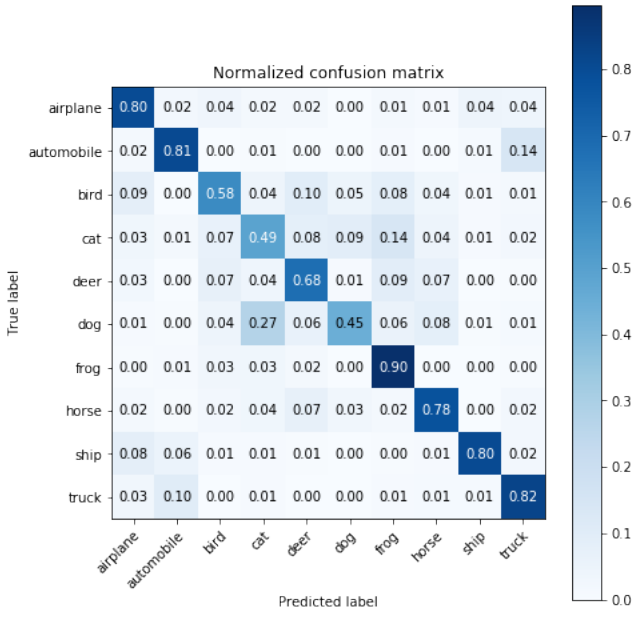
ResNet50

ResNet50 was trained for 100 epochs, resulting in a validation accuracy of 65.58%. The normalized confusion matrix for ResNet50 on CIFAR-10 is shown below



True label \ Predicted label	airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck
airplane	0.88	0.01	0.03	0.01	0.01	0.00	0.00	0.01	0.04	0.02
automobile	0.01	0.94	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.03
bird	0.03	0.00	0.82	0.03	0.04	0.03	0.04	0.01	0.01	0.00
cat	0.01	0.01	0.04	0.68	0.03	0.12	0.07	0.02	0.01	0.01
deer	0.01	0.00	0.03	0.02	0.84	0.02	0.03	0.04	0.00	0.00
dog	0.00	0.00	0.02	0.10	0.03	0.78	0.02	0.03	0.00	0.01
frog	0.01	0.00	0.02	0.02	0.01	0.01	0.92	0.00	0.00	0.01
horse	0.01	0.00	0.01	0.02	0.02	0.03	0.00	0.90	0.00	0.01
ship	0.04	0.01	0.01	0.01	0.00	0.00	0.00	0.00	0.92	0.01
truck	0.01	0.06	0.00	0.01	0.00	0.00	0.00	0.00	0.01	0.91

(a) Normalized confusion matrix VGG19



True label \ Predicted label	airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck
airplane	0.80	0.02	0.04	0.02	0.02	0.00	0.01	0.01	0.04	0.04
automobile	0.02	0.81	0.00	0.01	0.00	0.00	0.01	0.00	0.01	0.14
bird	0.09	0.00	0.58	0.04	0.10	0.05	0.08	0.04	0.01	0.01
cat	0.03	0.01	0.07	0.49	0.08	0.09	0.14	0.04	0.01	0.02
deer	0.03	0.00	0.07	0.04	0.68	0.01	0.09	0.07	0.00	0.00
dog	0.01	0.00	0.04	0.27	0.06	0.45	0.06	0.08	0.01	0.01
frog	0.00	0.01	0.03	0.03	0.02	0.00	0.90	0.00	0.00	0.00
horse	0.02	0.00	0.02	0.04	0.07	0.03	0.02	0.78	0.00	0.02
ship	0.08	0.06	0.01	0.01	0.01	0.00	0.00	0.01	0.80	0.02
truck	0.03	0.10	0.00	0.01	0.00	0.00	0.01	0.01	0.01	0.82

(b) Normalized confusion matrix res50

Figure 7: Normalized confusion matrix Comparison

7 Discussion

From the results, we observe that VGG19 outperforms ResNet50 on the CIFAR-10 dataset in terms of validation accuracy. However, ResNet50 has the advantage of deeper architecture, which may lead to better performance on more complex datasets. Further analysis and experimentation could provide insights into optimizing the performance of these models.

8 Conclusion

In conclusion, we compared the classification results of the CIFAR-10 dataset using three different models: Artificial Neural Network (ANN), Convolutional Neural Network (CNN), and Transfer Learning with the VGG16 model. The transfer learning model outperformed both ANN and CNN models, achieving an accuracy of 0.85 on the test set. This demonstrates the effectiveness of leveraging pre-trained models for image classification tasks, especially when dealing with limited training data. Further research could explore other pre-trained models and fine-tuning strategies to improve performance even further.