Neural Network Models for Object Recognition

Individual Presentation

Machine Learning Module - April 2024

M. Birkan Durak

Introduction

CIFAR-10 Dataset

Description

The CIFAR-10 dataset is a widely used benchmark dataset for image classification tasks.

Dataset Size
Total Images: 60,000

Training Images: 50,000

Test Images: 10,000

Image Characteristics:

Dimensions
32x32 pixels

Color Channels: 3 (RGB)

Resolution
Low resolution, which makes it computationally efficient for training and testing models.

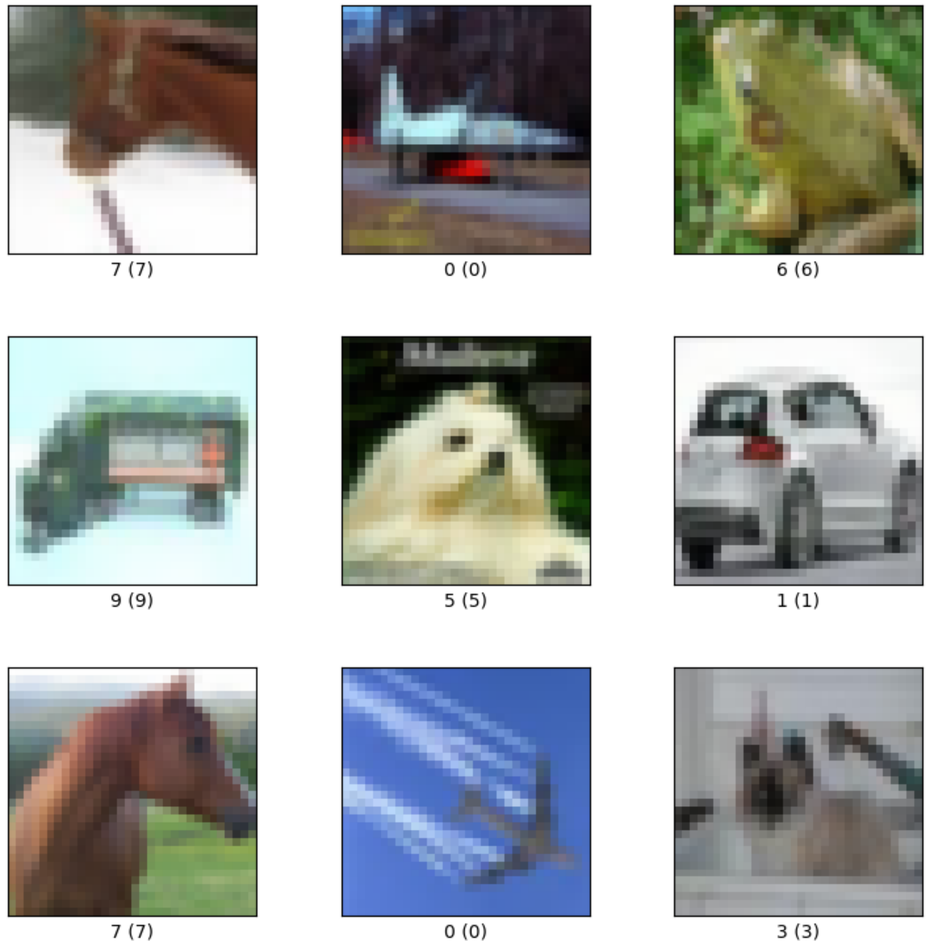


Figure 1. CIFAR-10 dataset images (Krizhevsky & Hinton (2009))

Introduction

CIFAR-10 Dataset

Classes: 10 mutually exclusive classes:

1.Airplane

2.Automobile

3.Bird

4. Cat

5. Deer

6. Dog

7. Frog

8. Horse

9. Ship

10. Truck

Class Distribution:

Training Set: Approximately 5,000 images per class

Test Set: Approximately 1,000 images per class

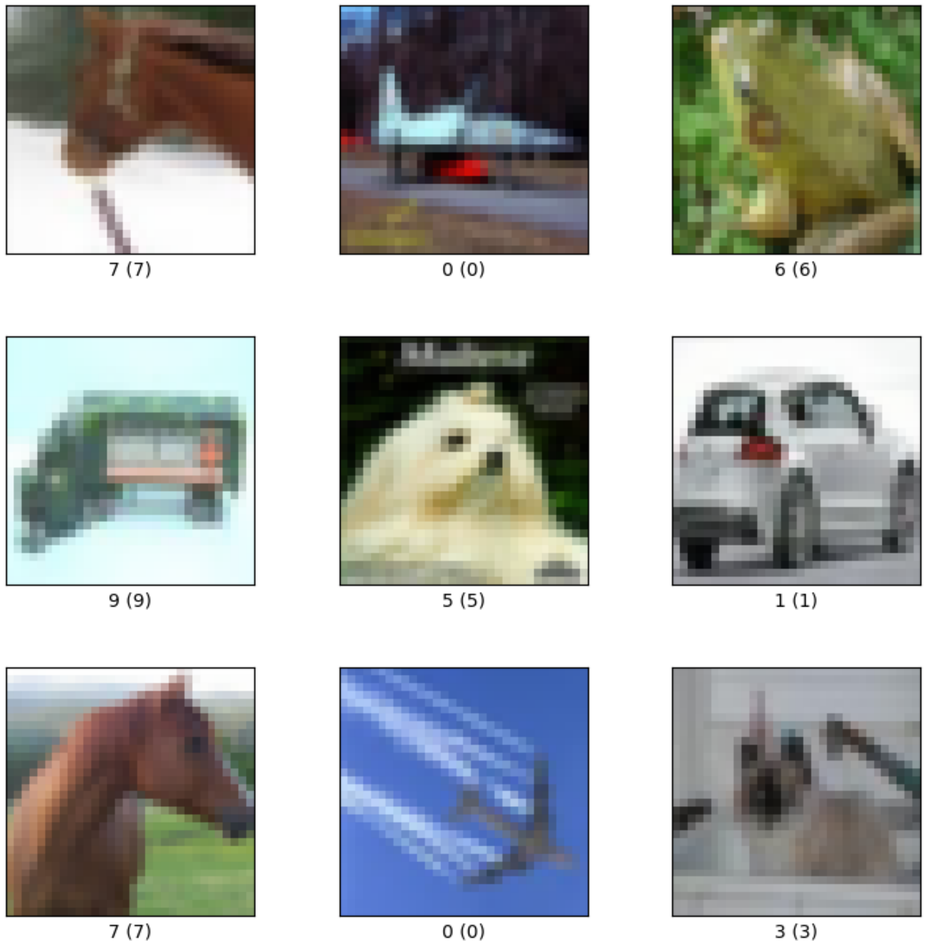


Figure 1. CIFAR-10 dataset images (Krizhevsky & Hinton (2009))

Data Partitioning

- I loaded the CIFAR-10 dataset using Keras, which provides 60,000 32x32 color images in 10 different classes, with 6,000 images per class. The dataset was split into 50,000 training images and 10,000 test images.

- I further split the training set into a training set and a validation set using an 80-20 split ratio. This resulted in 40,000 images for training and 10,000 images for validation. The splitting was done using the train_test_split function from Scikit-Learn to ensure a randomized and balanced distribution

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from keras.datasets import cifar10
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from keras.optimizers import Adam
from keras.preprocessing.image import ImageDataGenerator
from keras.utils import to_categorical
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report
import pandas as pd
```

```
# Load the CIFAR-10 dataset
(X_train, y_train), (X_test, y_test) = cifar10.load_data()

# Split the training set into training and validation sets (80-20 split)
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.2, random_state=42)
```

Data Partitioning

- Loaded data was checked and here are the results;

Total number of Images in the Dataset: 50000

Number of train images: 40000

Number of test images: 10000

Shape of training dataset: (40000, 32, 32, 3)

Shape of testing dataset: (10000, 32, 32, 3)

```
# Checking loaded data
print('Total number of Images in the Dataset:', len(X_train) + len(X_test))
print('Number of train images:', len(X_train))
print('Number of test images:', len(X_test))
print('Shape of training dataset:',X_train.shape)
print('Shape of testing dataset:',X_test.shape)
```

Why maintaining a separate validation set is important?

- **Model Tuning:** It allows us to tune hyperparameters and validate the model's performance during training, preventing overfitting to the training data.

- **Performance Monitoring:** By using a validation set, we can monitor the model's performance on unseen data and make adjustments as needed.

- **Generalization:** Ensures that the model generalizes well to new, unseen data. Without a validation set, we risk creating a model that performs well on training data but poorly on real-world data.

Model Architecture

- **Input Layer:** Accepts 32x32x3 images.
- **Convolutional Layers:** I used three convolutional blocks with increasing filter sizes of 32, 64, and 128. These layers apply convolution operations to detect features such as edges, textures, and more complex patterns.
- **MaxPooling Layers:** Each convolutional block is followed by a MaxPooling layer to reduce spatial dimensions and computational load, making the model more efficient.

```
model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(MaxPooling2D((2, 2)))
model.add(Dropout(0.25))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Dropout(0.25))
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))
```

Model Architecture

- **Dropout Layers:** Added after each convolutional block to prevent overfitting by randomly dropping neurons during training.
- **Fully Connected Layers:** After flattening the output from the convolutional layers, I pass it through a dense layer with 128 units, which helps in combining all the features learned by the convolutional layers.
- **Output Layer:** A dense output layer with 10 units and a softmax activation function to classify the images into one of the 10 classes.

Activation function

- ReLU (Rectified Linear Unit) activation function was chosen for my convolutional and fully connected layers.

- ReLU is computationally efficient and helps to mitigate the vanishing gradient problem, allowing my model to converge faster. It introduces non-linearity into the model, which helps in learning complex patterns. The final layer uses the softmax activation function to produce a probability distribution over the 10 classes, which is essential for multi-class classification.

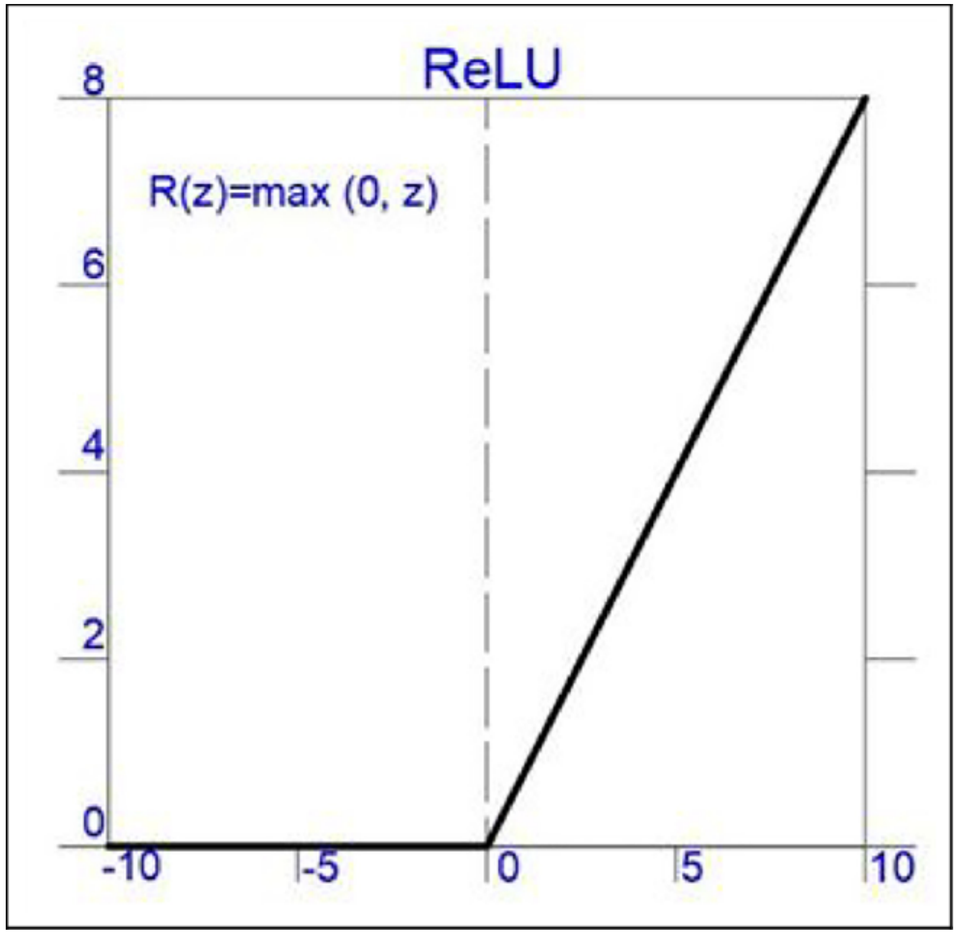


Figure 2. ReLU graph (Researchgate 2023)

Loss function

- My loss function is Categorical Crossentropy, which is well-suited for multi-class classification tasks.
- This function measures the difference between the true labels and the predicted probabilities, guiding the optimization process to minimize this difference.
- By minimizing the categorical crossentropy, it was ensured that the predicted probabilities are as close as possible to the true distribution.

Number of epochs

- I trained the model for 50 epochs.
- This number was chosen based on monitoring the model’s performance and ensuring that it was learning effectively without overfitting.
- Higher numbers were tried yet 50 was enough.
- During training, I observed that both the training and validation loss decreased over time, indicating that the model was improving. Early stopping could be considered in future iterations to prevent any potential overfitting by halting training when the validation performance stops improving.

Neural Network Design Strategy

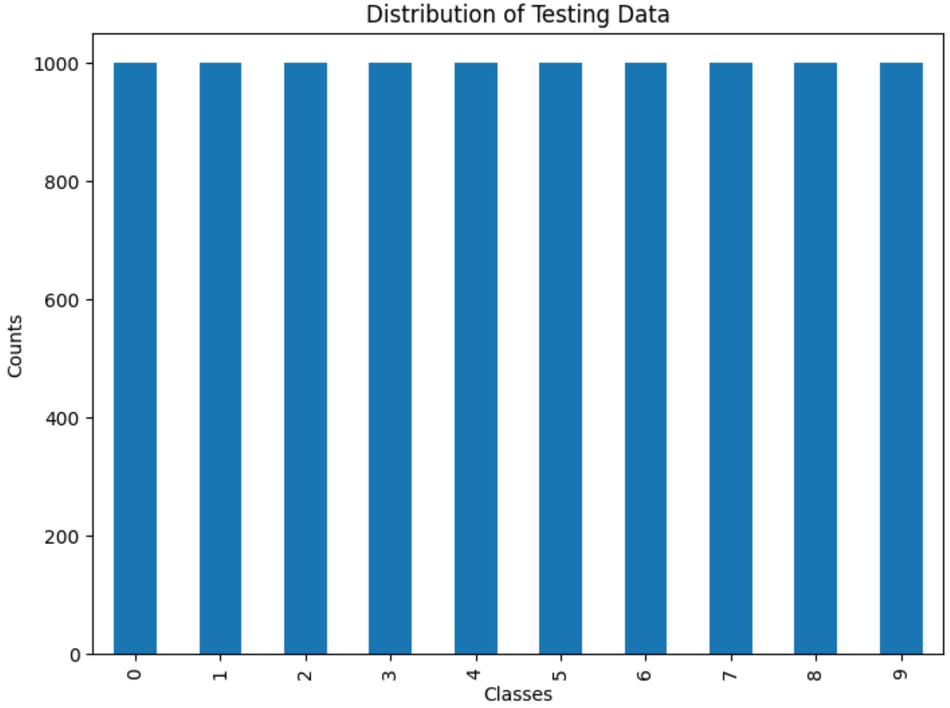
- **Regularization:** Dropout layers were used to prevent overfitting by randomly dropping neurons during training, which forces the network to learn redundant representations.
- **Data Augmentation:** I applied various transformations, such as rotation, width shift, height shift, and horizontal flip, to increase the diversity of the training data. This helps the model generalize better to unseen data.
- **Optimizer:** I used the Adam optimizer, known for its efficiency and effectiveness in handling sparse gradients.
- **Model Complexity:** I designed the model with three convolutional layers with increasing filter sizes to capture complex patterns in the images. Each layer extracts different levels of features, from simple edges to complex textures and shapes.

Results

- **Test Accuracy:** The model achieved an accuracy of 68.29% on the test set.
- **Test Loss:** The loss on the test set was 0.9502.

Results

- Graph for distribution of classes.



Results

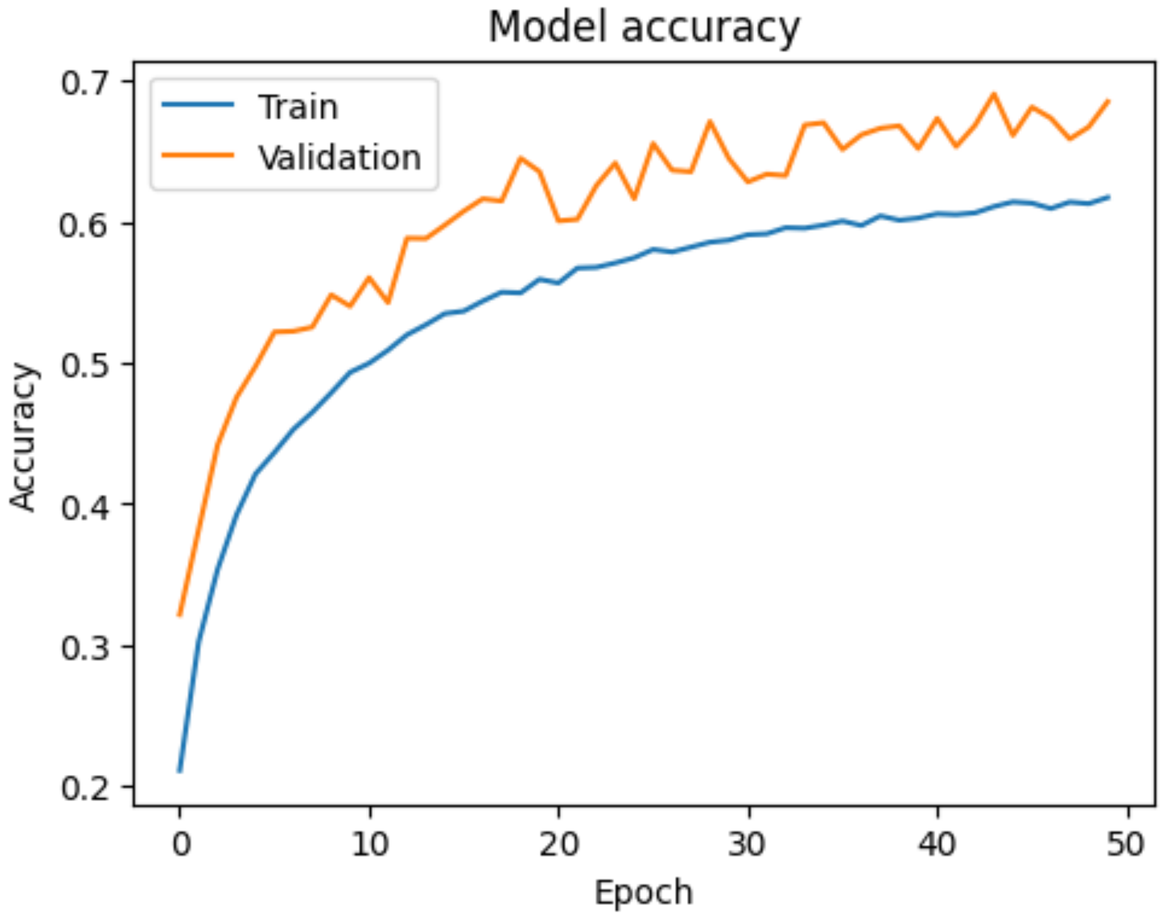
Model Accuracy and Loss

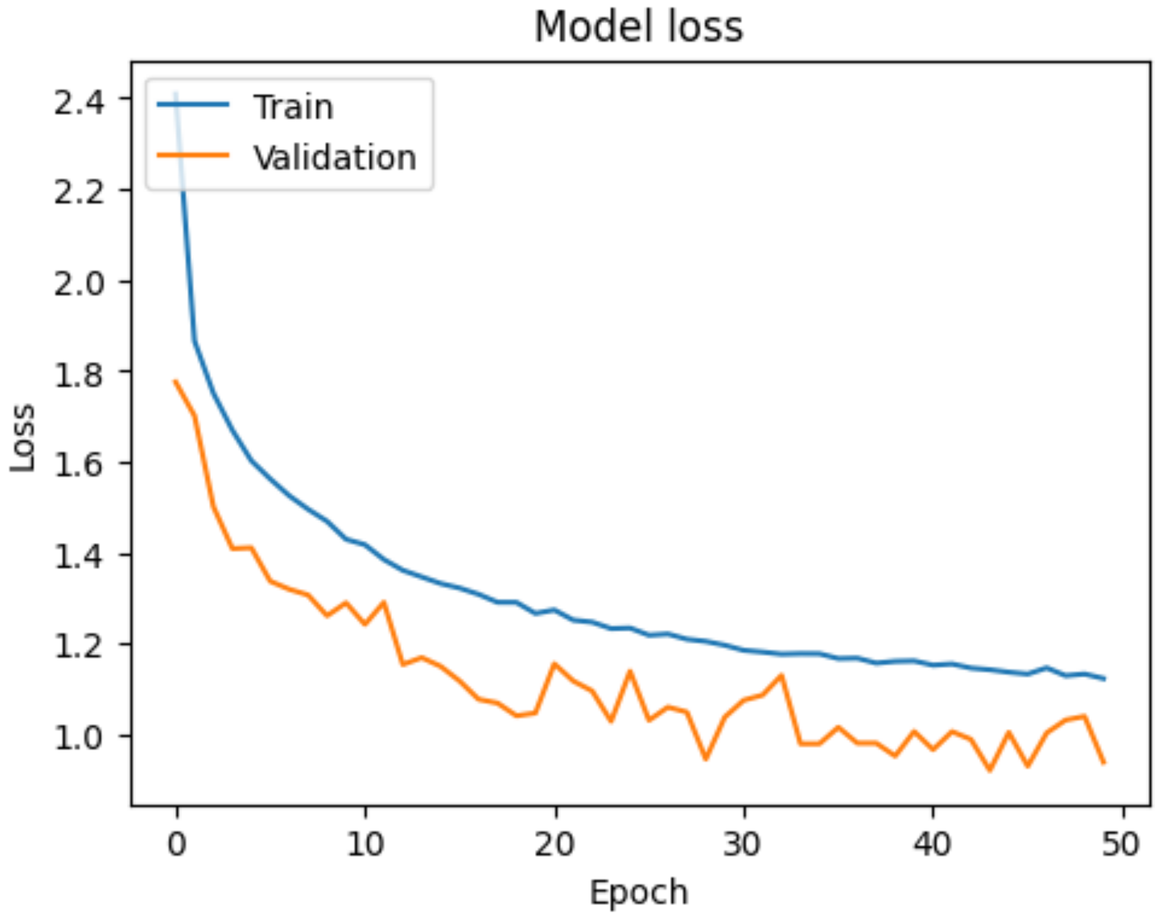
The plots show the training and validation accuracy and loss over 50 epochs:

- The training accuracy steadily increased and plateaued around 60%.
- The validation accuracy fluctuated around 65-70%, indicating some variability in generalization.
- Both the training and validation loss decreased over time, indicating that the model was learning effectively.

Results

Model Accuracy and Loss





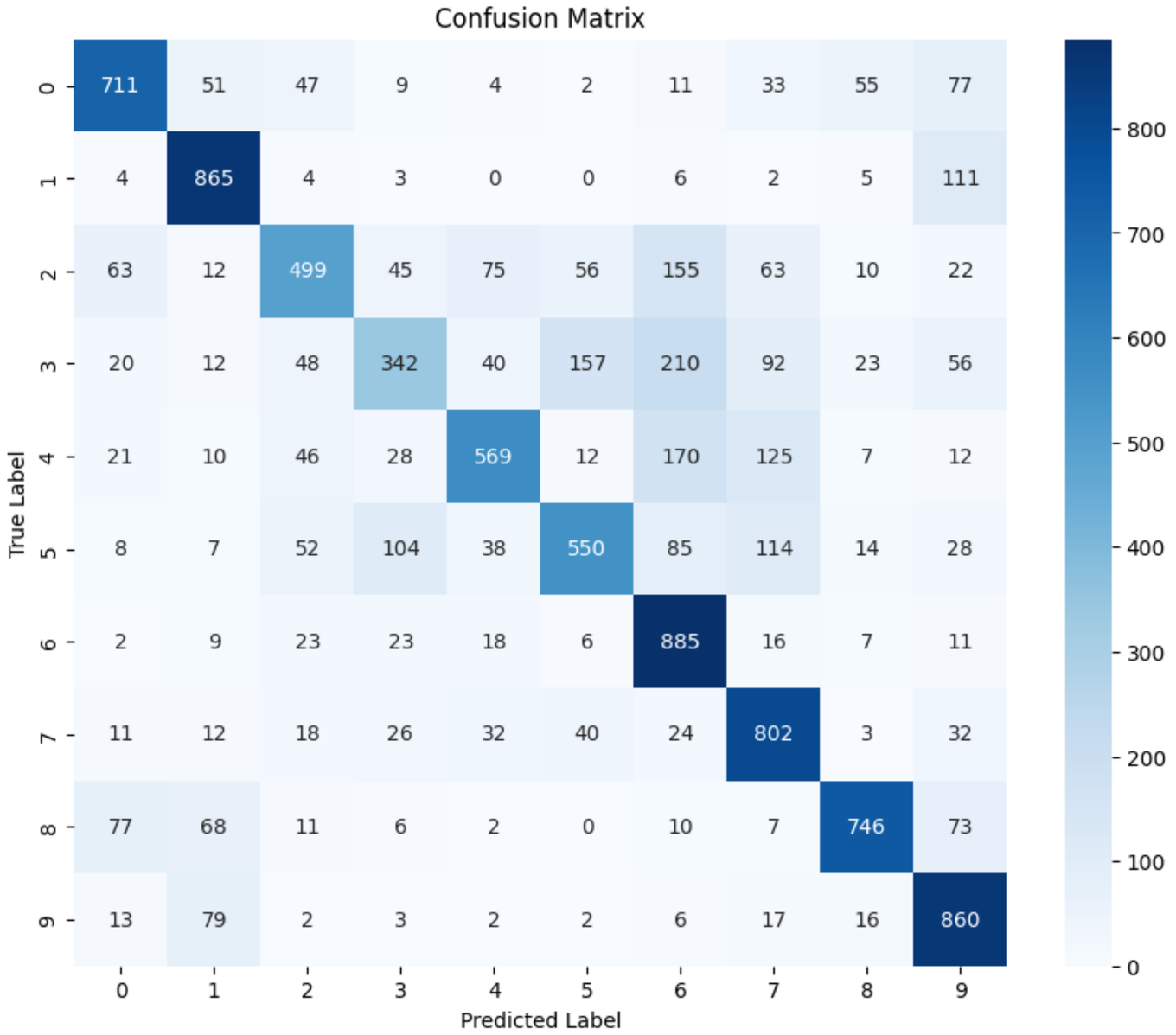
Results

Confusion matrix

- The diagonal elements indicate correct classifications.
- Some classes, like ‘airplane’ and ‘automobile’, had high accuracy, while others, like ‘bird’ and ‘cat’, had higher misclassification rates.
- Misclassifications can often be seen between visually similar classes, such as ‘cat’ and ‘dog’, ‘automobile’ and ‘truck’, etc.

Results

Confusion matrix



Classification Report

Precision: Measures the accuracy of positive predictions. High precision for a class means that there are few false positives.

Recall: Measures the ability to find all positive samples. High recall for a class means that there are few false negatives.

F1-Score: Harmonic mean of precision and recall, providing a balance between the two. High F1-score for a class means a good balance of precision and recall.

The macro average precision and recall were around 69%, with an overall accuracy of 68%.

	precision	recall	f1-score	support
airplane	0.81	0.64	0.71	1000
automobile	0.82	0.82	0.82	1000
bird	0.74	0.42	0.53	1000
cat	0.56	0.32	0.41	1000
deer	0.68	0.56	0.61	1000
dog	0.69	0.54	0.60	1000
frog	0.47	0.93	0.63	1000
horse	0.74	0.72	0.73	1000
ship	0.82	0.77	0.79	1000
truck	0.59	0.93	0.72	1000
accuracy			0.66	10000
macro avg	0.69	0.66	0.66	10000
weighted avg	0.69	0.66	0.66	10000

Reflections

Model Generalization: Using a validation set helped ensure the model generalized well to new, unseen data.

Performance: The model achieved reasonable accuracy but struggled with certain classes due to higher misclassification rates.

Challenges: Improving model performance for classes with lower accuracy, like ‘bird’ and ‘cat’. Understanding why these classes have higher error rates and addressing those specific issues can help improve overall performance.

Improvements: Potential for improvement with more complex architectures, such as deeper networks, residual networks, and additional regularization techniques. Hyperparameter tuning, such as adjusting learning rates and dropout rates, could also yield better performance.

Visualization and Analysis: Visualizing the results using confusion matrices and classification reports provided valuable insights into where the model was performing well and where it needed improvement.

Summary

Data Partitioning: I carefully partitioned the CIFAR-10 dataset which comprises 60,000 images across 10 classes (6,000 images per class), was first divided into 50,000 training images and 10,000 test images. I then further split the training set into 40,000 training images and 10,000 validation images using the train_test_split function from Scikit-Learn.

Model Architecture: My neural network architecture included multiple convolutional layers, MaxPooling layers, Dropout layers for regularization, and fully connected layers. The ReLU activation function was used to introduce non-linearity, and the softmax function in the output layer was used for multi-class classification.

Training Process: The model was trained for 50 epochs using the Adam optimizer and categorical crossentropy loss function. Data augmentation techniques were applied to improve generalization.

Performance Metrics: My model achieved a test accuracy of 68.29% and a test loss of 0.9502. The classification report and confusion matrix provided insights into the model's strengths and areas for improvement.

Challenges and Insights: I observed that certain classes, such as 'bird' and 'cat', had higher misclassification rates. Addressing these specific issues can help improve overall performance.

Future work

- **Model Complexity:** Experiment with deeper networks and advanced architectures like residual networks.
- **Hyperparameter Tuning:** Fine-tune hyperparameters, including learning rates and dropout rates.
- **Transfer Learning:** Utilize pre-trained models to leverage knowledge from larger datasets.
- **Enhanced Data Augmentation:** Apply more sophisticated data augmentation techniques to further improve generalization.
- **Early Stopping:** Implement early stopping to prevent overfitting and optimize training time.

Thank you for listening.

References

- Krizhevsky, A., & Hinton, G. (2009). Learning Multiple Layers of Features from Tiny Images. Retrieved from https://www.cs.toronto.edu/~kriz/cifar.html

- ResearchGate. (2023). Activation function: ReLU (Rectified Linear Activation). [online]
  Available at: https://www.researchgate.net/figure/Activation-function-ReLu-ReLu-Rectified-Linear-Activation_fig2_370465617 [Accessed 12 July 2024]