Computational Intel. | Report

Assignment Section B

Student: Vladimir W. Schmadlak
Id: st20283948

Github Repository: https://github.com/VladimorCodebreaker/CompInt-CIS6005-Assignment-SectionB.git

Important Notice: For a comprehensive review of the complete code that generates the outputs discussed in this report, please refer to my GitHub repository. This document includes only selected snippets of the output for illustrative purposes. For full understanding and evaluation, I kindly invite you to visit the GitHub link provided. Furthermore, please note that the feedback from “Section A” of the lecturer's remarks has been meticulously applied. I hope this report is thorough enough. Cheers :)

Introduction

This report outlines the process and findings of the assignment for the “Computational Intelligence” course, focusing on the application of Convolutional Neural Networks (CNNs) for object recognition using the CIFAR-10 dataset. The assignment involved creating, training, and evaluating two distinct neural network models.

Dataset Overview | CIFAR-10

The CIFAR-10 dataset, a subset of the 80 million tiny images dataset, contains 60,000 32x32 color images in 10 different classes, with 6,000 images per class. It is widely used in the field of computer vision for object recognition tasks. It was collected by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton, and has become a standard for evaluating the performance of image recognition algorithms due to its balance and complexity, simulating real-world conditions where object recognition systems must identify and classify diverse subjects.

Creation of Validation Set and Data Metadata

A validation set was created from the original training data by employing a stratified split that ensures the class distribution remains consistent across the training and validation sets. Specifically, 20% of the training data was segregated to form the validation set using the `train_test_split` function from `sklearn.model_selection`, with `random_state=42` to ensure reproducibility. This resulted in a training set with 40,000 images and a validation set with 10,000 images. The test set remained unchanged with 10,000 images.

Metadata

- **Training Data Shape**: 40,000 samples, each of 32x32 pixels in 3 RGB color channels.
- **Validation Data Shape**: 10,000 samples with identical dimensions to the training set.
- **Testing Data Shape**: 10,000 samples, consistent with training and validation sets.
- **Data Types**: Image pixel values were initially of integer type ranging from 0 to 255 and were normalized to float types ranging from 0 to 1.
- **Class Labels**: 10 categories named as 'airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck'.
- **Class Distribution**: Equal distribution with each class having 6,000 images in the original dataset.
- **Label Encoding**: Labels were one-hot encoded to transform categorical labels into a binary matrix representation required for neural network classification.

Justification of Preprocessing Steps

Normalization

Pixel values in the images are initially in the range of 0 to 255 (integers). These are normalized to a range of 0 to 1 (floats) to aid in the training process. Normalization makes the training faster and reduces the chances of getting stuck in local optima since we're dealing with smaller and more manageable values.

```
x_train_all, x_test = x_train_all / 255.0, x_test / 255.0
```

One-Hot Encoding

The labels are one-hot encoded to transform them from a single integer to a binary matrix. In the context of a multi-class classification problem, this encoding is crucial for the proper functioning of the neural network, ensuring that each class is represented as a distinct entity in a way the network can understand and process.

```
y_train_all = to_categorical(y_train_all, num_classes=10)
y_test = to_categorical(y_test, num_classes=10)
```

Validation Set Creation

20% of the training data is segregated to form a validation set. This is done to evaluate the model's performance on unseen data during the training process. The choice of 20% strikes a balance between having enough data for a robust validation and retaining sufficient data for training the models. The `random_state` parameter ensures that the split is reproducible, an important aspect for experimental consistency.

```
x_train, x_val, y_train, y_val = train_test_split(x_train_all, y_train_all, test_size=0.2, random_state=42)
```

Data Split Shapes

Post-split, the shapes of the new training and validation sets are displayed to confirm the successful split. This step is important for verifying that the data has been partitioned as intended and that the models will receive data in the expected format.

```
print(f"New training data shape: {x_train.shape}")
print(f"New training labels shape: {y_train.shape}")
print(f"Validation data shape: {x_val.shape}")
print(f"Validation labels shape: {y_val.shape}")
```

Data Shape Analysis

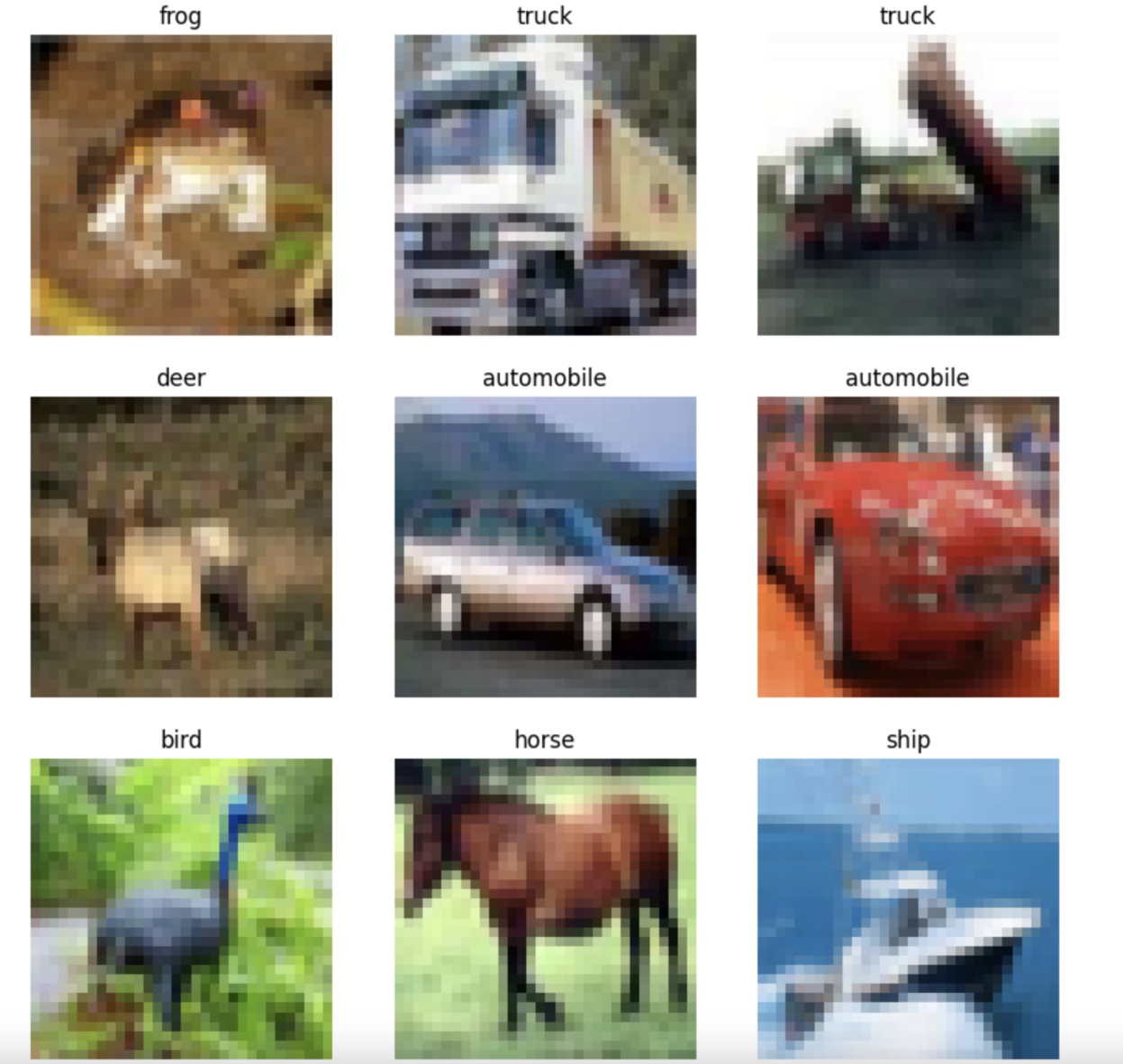
Training and Test Data Shape

The CIFAR-10 dataset comprises images with a shape of 32x32 pixels, each having 3 channels (RGB). The training set originally contains 50,000 images, while the test set has 10,000. Understanding the shape and structure of this data is crucial for designing the input layer of the neural networks and for ensuring that the preprocessing steps are appropriately applied.

```
Training data shape: (50000, 32, 32, 3)
Training labels shape: (50000, 1)
Test data shape: (10000, 32, 32, 3)
Test labels shape: (10000, 1)
```

Visualization of Dataset Samples

To gain a more intuitive understanding of the dataset, a few sample images are displayed, each labeled with its corresponding class (e.g., 'frog', 'automobile', etc.). This visualization not only showcases the variety within the dataset but also highlights the challenges inherent in image classification tasks, such as varying backgrounds and object scales.



Validation Data Shape

After splitting, the training set is reduced to 40,000 images, and a validation set of 10,000 images is formed. This split is vital for model evaluation during the training process without touching the test set. The validation set mirrors the shape and structure of the training and test sets, ensuring consistency in model evaluation.

```
New training data shape: (40000, 32, 32, 3)
New training labels shape: (40000, 10)
Validation data shape: (10000, 32, 32, 3)
Validation labels shape: (10000, 10)
```

Rationale for Having a Validation Set

A validation set serves several critical purposes in the machine learning workflow:

- It provides an unbiased evaluation of a model's performance during training, which is essential for hyper-parameter tuning and model selection.
- It helps detect overfitting early on, as a model performing well on training data but poorly on validation data is likely memorizing the training data rather than learning generalizable patterns.
- It allows for the testing of model robustness to new data before the final evaluation on the test set, which should remain untouched until the model's final assessment.

The use of a validation set is crucial for developing a model that not only learns from the training data but also generalizes well to unseen data!

Neural Network Architectures

Two distinct Convolutional Neural Network (CNN) models were developed to tackle the CIFAR-10 dataset's classification task. Each model was designed with a unique architecture to evaluate the influence of different structural configurations on performance.

Model 1 Architecture

Model 1 is a convolutional neural network designed to capture the hierarchical structure of images through a sequence of convolutional layers. The model starts with a 32-filter convolutional layer with a 5x5 kernel, followed by batch normalization and max pooling to reduce spatial dimensions. It then applies a dropout of 10% to prevent overfitting.

This pattern is repeated with varying filter sizes and dropout rates, allowing the network to learn more complex features at each level. After flattening the convolved features, the model employs a dense layer with 64 units and a final dense layer with 10 units corresponding to the class categories, using the `softmax` activation function for multi-class classification.

```
# Cell 4: First Neural Network Model

# Define the CNN architecture
model_1 = Sequential([
    Conv2D(filters=32, kernel_size=(5, 5), padding='same', activation='relu', input_shape=(32, 32, 3)),
    BatchNormalization(),
    MaxPooling2D((2, 2)),
    Dropout(0.1),

    Conv2D(filters=64, kernel_size=(3, 3), padding='same', activation='relu'),
    BatchNormalization(),
    MaxPooling2D((2, 2)),
    Dropout(0.2),

    Conv2D(filters=64, kernel_size=(5, 5), padding='same', activation='relu'),
    BatchNormalization(),
    MaxPooling2D((2, 2)),
    Dropout(0.2),

    Flatten(),
    Dense(64, activation='relu'),
    BatchNormalization(),
    Dropout(0.3),
    Dense(10, activation='softmax')
])

# Compile the model
model_1.compile(optimizer='adam',
                loss='categorical_crossentropy',
                metrics=['accuracy'])

# Print the model summary
model_1.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 32, 32, 32)        2432

 batch_normalization (Batch  (None, 32, 32, 32)        128
 Normalization)

 max_pooling2d (MaxPooling2  (None, 16, 16, 32)        0
 D)

 dropout (Dropout)           (None, 16, 16, 32)        0

 conv2d_1 (Conv2D)           (None, 16, 16, 64)        18496

 batch_normalization_1 (Bat  (None, 16, 16, 64)        256
 chNormalization)

 max_pooling2d_1 (MaxPoolin  (None, 8, 8, 64)          0
 g2D)

 dropout_1 (Dropout)         (None, 8, 8, 64)          0

 conv2d_2 (Conv2D)           (None, 8, 8, 64)          102464

 batch_normalization_2 (Bat  (None, 8, 8, 64)          256
 chNormalization)

 max_pooling2d_2 (MaxPoolin  (None, 4, 4, 64)          0
 g2D)

 dropout_2 (Dropout)         (None, 4, 4, 64)          0

 flatten (Flatten)           (None, 1024)              0

 dense (Dense)               (None, 64)                65600

 batch_normalization_3 (Bat  (None, 64)                256
 chNormalization)

 dropout_3 (Dropout)         (None, 64)                0

 dense_1 (Dense)             (None, 10)                650

=================================================================
Total params: 190538 (744.29 KB)
Trainable params: 190090 (742.54 KB)
Non-trainable params: 448 (1.75 KB)
_________________________________________________________________
```

Model 2 Architecture

Model 2 follows a similar foundational structure but includes additional convolutional layers before max pooling to increase the depth of the feature extraction process. It starts with two 32-filter convolutional layers with a 3x3 kernel size, followed by batch normalization, max pooling, and a dropout of 10%. The pattern is replicated with 64 filters, and after flattening, a larger dense layer with 128 units is used before the final classification layer. This model is expected to capture more detailed features due to the added layers and larger dense layer, but it also risks overfitting, which is mitigated by a higher dropout rate of 50%.

```python
# Cell 6: Second Neural Network Model

# Define the CNN architecture
model_2 = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
    BatchNormalization(),
    Conv2D(32, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Dropout(0.1),

    Conv2D(64, (3, 3), activation='relu'),
    BatchNormalization(),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Dropout(0.3),

    Flatten(),
    Dense(128, activation='relu'),
    BatchNormalization(),
    Dropout(0.5),
    Dense(10, activation='softmax')
])

# Compile the model
model_2.compile(
    optimizer='rmsprop',
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

# Print the model summary
model_2.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_3 (Conv2D)           (None, 30, 30, 32)        896

 batch_normalization_4 (Bat  (None, 30, 30, 32)        128
 chNormalization)

 conv2d_4 (Conv2D)           (None, 28, 28, 32)        9248

 max_pooling2d_3 (MaxPoolin  (None, 14, 14, 32)        0
 g2D)

 dropout_4 (Dropout)         (None, 14, 14, 32)        0

 conv2d_5 (Conv2D)           (None, 12, 12, 64)        18496

 batch_normalization_5 (Bat  (None, 12, 12, 64)        256
 chNormalization)

 conv2d_6 (Conv2D)           (None, 10, 10, 64)        36928

 max_pooling2d_4 (MaxPoolin  (None, 5, 5, 64)          0
 g2D)

 dropout_5 (Dropout)         (None, 5, 5, 64)          0

 flatten_1 (Flatten)         (None, 1600)              0

 dense_2 (Dense)             (None, 128)               204928

 batch_normalization_6 (Bat  (None, 128)               512
 chNormalization)

 dropout_6 (Dropout)         (None, 128)               0

 dense_3 (Dense)             (None, 10)                1290

=================================================================
Total params: 272682 (1.04 MB)
Trainable params: 272234 (1.04 MB)
Non-trainable params: 448 (1.75 KB)
_________________________________________________________________
```

Both models were compiled with different optimizers—Model 1 with `adam` and Model 2 with `rmsprop`—and the same loss function, `categorical_crossentropy`, which is standard

for multi-class classification tasks.

Comparative Analysis of CNN Models

Architectural Differences

- **Model 1** uses larger convolutional filters (5x5) in its first and third layers, while **Model 2** consistently employs smaller 3x3 filters. Larger filters in Model 1 can capture more spatial information in a single receptive field, potentially making it better at recognizing larger patterns. In contrast, the smaller filters in Model 2 are more focused on capturing finer details.

- **Depth and Complexity**: Model 2 is deeper with more convolutional layers before each max pooling layer, which might allow it to learn more complex and abstract features. However, this increased depth can also make it more prone to overfitting, which is somewhat mitigated by its higher dropout rates.

Optimizer Differences

- **Model 1** uses the `adam` optimizer, known for its adaptability in learning rates and suitability for a wide range of problems.

```
model_1.compile(optimizer='adam',
                loss='categorical_crossentropy',
                metrics=['accuracy'])
```

- **Model 2** employs `rmsprop`, which is also adaptive but can be more effective in scenarios where gradients are very noisy or have different variances.

```
model_2.compile(
    optimizer='rmsprop',
    loss='categorical_crossentropy',
    metrics=['accuracy'])
```

Advantages and Drawbacks

- **Model 1**:
  - **Advantage**: Potentially faster training due to fewer layers and larger filters capturing more information at once.
  - **Drawback**: Might not be as effective in learning intricate details due to larger filter sizes and fewer convolutional layers.

- **Model 2**:
  - **Advantage**: More capable of capturing detailed features due to its deeper architecture and smaller filters.
  - **Drawback**: Higher risk of overfitting despite the increased dropout rate and potentially longer training times due to more layers.

Suitability for CIFAR-10

- CIFAR-10 consists of small images (32x32 pixels), so the choice of filter size and network depth is crucial. Model 1's larger filters might be less effective for such small images, but its simpler architecture could be sufficient given the limited resolution. Model 2's smaller filters and deeper architecture might be more adept at capturing the subtleties within these small images, but at the cost of computational efficiency and a higher risk of overfitting.

Activation Function

Both models use the ReLU (`relu`) activation function for the hidden layers. ReLU is chosen for its efficiency and effectiveness in introducing non-linearity to the model, allowing it to learn more complex patterns. For the output layer, the `softmax` activation function is used in both models, which is standard for multi-class classification problems. Softmax turns logits, the numerical output of the last linear layer of a model, into probabilities by comparing the relative scale of each logit.

Loss Function

The loss function used in both models is categorical crossentropy (`categorical_crossentropy`). It measures the performance of the output of a classification model whose output is a probability value between 0 and 1. Categorical crossentropy compares the distribution of the predictions with the true distribution and produces a probability score that quantifies the model's prediction accuracy for a given sample.

Training Process

Epochs and Early Stopping

Both models were set to train for up to **30 epochs** with a batch size of 64. However, an early stopping mechanism was employed to halt training if the validation loss did not improve for three consecutive epochs. Model 1 completed 17 epochs, before early stopping was triggered, while Model 2 finished 12 epochs. This approach ensures that the models do not continue to train beyond the point of convergence, avoiding overfitting and unnecessary computation.

Reduce Learning Rate on Plateau

A callback to reduce the learning rate when the validation loss plateaus was used for both models. This technique helps the model to fine-tune its weights and potentially achieve better performance by taking smaller steps when it seems to have reached a stall in learning progress.

Model Evaluation and Results Discussion

Model Performance During Training

During the training process, Model 1 achieved a final accuracy of 80.8% on the validation set after 17 epochs, while Model 2 reached an accuracy of 79.1% after 12 epochs. The early stopping callback was instrumental in preventing overtraining, as it halted the training process once the validation loss stopped improving, indicating that the model had reached its optimal state given the architecture and data.

```
Epoch 17/30
625/625 [==============================] - 31s 50ms/step - loss: 0.5502 - accuracy: 0.8085 - val_loss: 0.7933 - val_accuracy: 0.7393 - lr:
```

```
Epoch 12/30
625/625 [==============================] - 32s 51ms/step - loss: 0.6104 - accuracy: 0.7916 - val_loss: 0.7106 - val_accuracy: 0.7533 - lr:
```

Test Set Evaluation

Upon completion of training, both models were evaluated on the test set, which had not been seen by the models during the training process. Model 1 achieved a test accuracy of 76.23% with a loss of 0.6942, while Model 2 attained a test accuracy of 75.63% with a slightly higher loss of 0.7143. The decrease in performance from validation to test accuracy suggests that while the models generalize well, there is room for improvement, potentially by refining the network architecture or employing additional regularization techniques.

```
Model 1 - Test Loss: 0.6942, Test Accuracy: 0.7623
Model 2 - Test Loss: 0.7143, Test Accuracy: 0.7563
```

Additional Experimentation: Varying the Number of Epochs

In addition to the main training sessions, which were conducted for up to 30 epochs, I conducted supplementary experiments to observe the impact of varying the number of epochs on model performance. Specifically, I trained both models for a shorter duration of 10 epochs. The results were as follows:

- For Model 1, the accuracy achieved after 10 epochs was 0.7793.

```
Epoch 10/10
625/625 [==============================] - 22s 34ms/step - loss: 0.6312 - accuracy: 0.7793 - val_loss: 0.6980 - val_accuracy: 0.7561 - lr: 3.
0000e-04
```

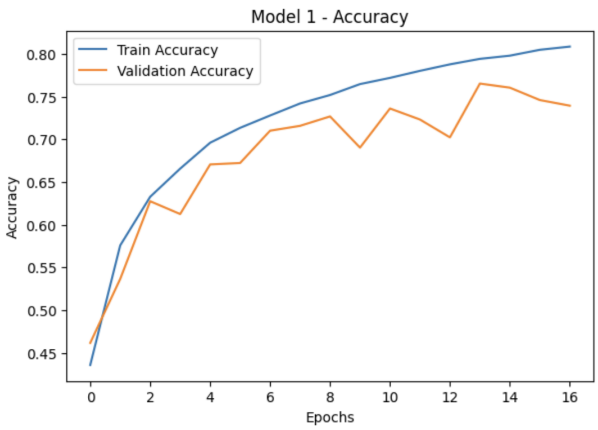
- For Model 2, the accuracy achieved after 10 epochs was 0.7778.

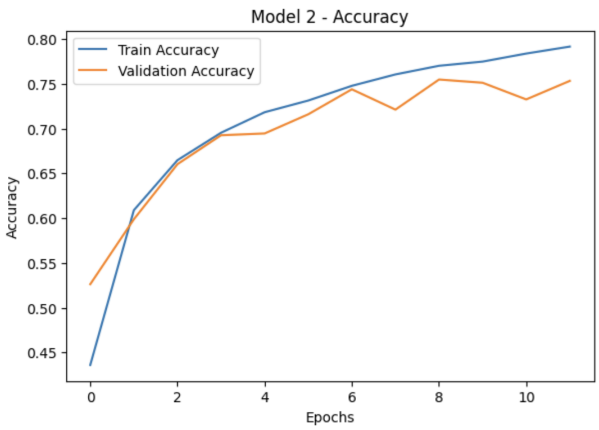
```
Epoch 10/10
625/625 [==============================] - 22s 35ms/step - loss: 0.6431 - accuracy: 0.7778 - val_loss: 0.6810 - val_accuracy: 0.7589 - lr: 0.
0010
```

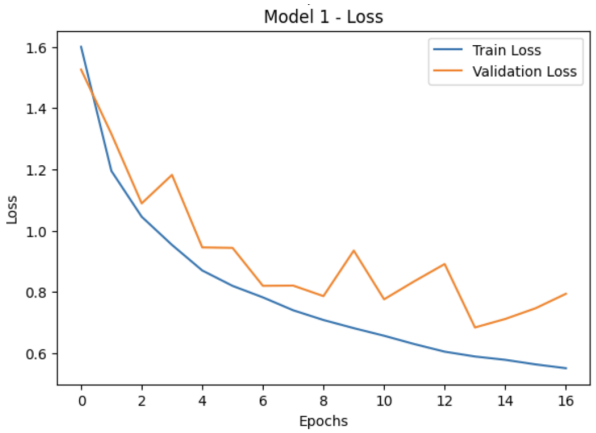
These outcomes were insightful, indicating that even with a reduced number of epochs, the models were able to achieve comparable accuracies to those obtained in longer training sessions. This suggests that for some models, a shorter training period might be sufficient to reach a near-optimal state, which can be especially beneficial in terms of reducing computational resources and time. It is important to note, however, that while these results are promising, shorter training periods might not always be ideal for more complex datasets or models.

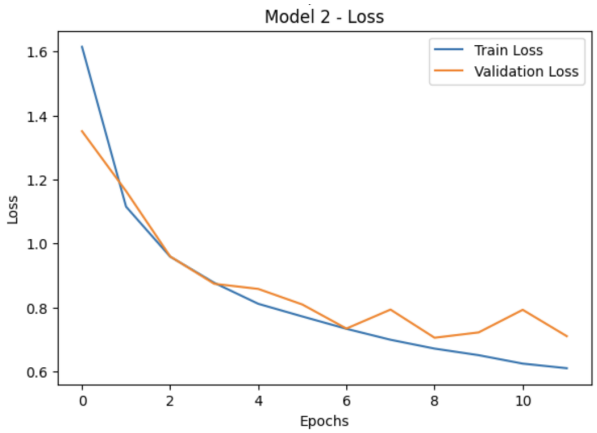
Analysis of Learning Curves

The learning curve graphs reveal that both models' validation accuracy closely tracks the training accuracy, which is a good sign of generalization. However, there are some fluctuations in the validation loss and accuracy, particularly for Model 1. This could be a sign of the model beginning to overfit to the training data, but the early stopping mechanism effectively mitigates this by halting training at an appropriate time.



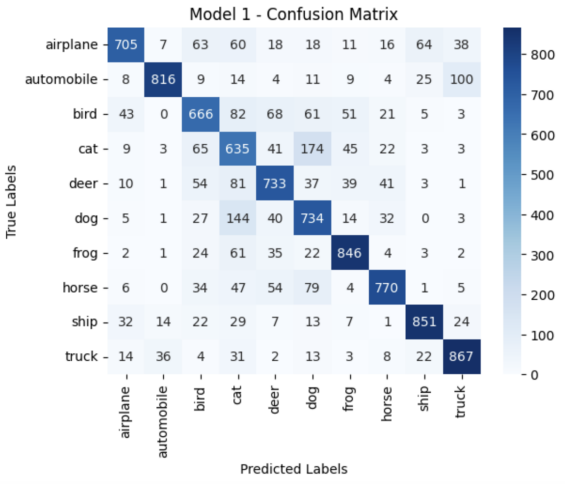


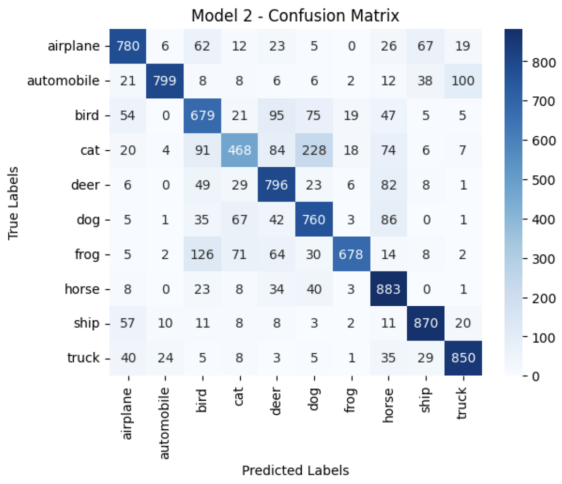




Confusion Matrix and Classification Report

The confusion matrices provide valuable data into the specific areas where each model excels or struggles. For instance, both models perform well on classes with distinctive features, such as 'automobile' and 'ship'. However, they confuse classes with similar features, like 'cat' and 'dog'. The classification reports further support this, with high precision, f1-score and recall for classes like 'automobile' and lower scores for 'cat', indicating challenges in distinguishing these classes accurately.





Classification Report for Model 1:

	precision	recall	f1-score	support
airplane	0.85	0.70	0.77	1000
automobile	0.93	0.82	0.87	1000
bird	0.69	0.67	0.68	1000
cat	0.54	0.64	0.58	1000
deer	0.73	0.73	0.73	1000
dog	0.63	0.73	0.68	1000
frog	0.82	0.85	0.83	1000
horse	0.84	0.77	0.80	1000
ship	0.87	0.85	0.86	1000
truck	0.83	0.87	0.85	1000
accuracy			0.76	10000
macro avg	0.77	0.76	0.77	10000
weighted avg	0.77	0.76	0.77	10000

Classification Report for Model 2:

	precision	recall	f1-score	support
airplane	0.78	0.78	0.78	1000
automobile	0.94	0.80	0.87	1000
bird	0.62	0.68	0.65	1000
cat	0.67	0.47	0.55	1000
deer	0.69	0.80	0.74	1000
dog	0.65	0.76	0.70	1000
frog	0.93	0.68	0.78	1000
horse	0.70	0.88	0.78	1000
ship	0.84	0.87	0.86	1000
truck	0.84	0.85	0.85	1000
accuracy			0.76	10000
macro avg	0.77	0.76	0.76	10000
weighted avg	0.77	0.76	0.76	10000

Neural Network Design Approach

- **Layer Architecture**: Both models used convolutional layers to effectively learn the spatial hierarchies in image data. Model 1 was designed with increasing dropout rates and filter sizes to progressively abstract image features at different levels. Model 2 added complexity by increasing the depth of convolutional layers before max pooling to extract more granular features.
- **Regularization**: Dropout layers were strategically placed to reduce overfitting by randomly deactivating a subset of neurons during training. This encourages the network to learn more robust features that generalize better to unseen data.
- **Normalization**: Batch normalization was applied after each convolutional operation to stabilize learning and accelerate the training process by normalizing the inputs to

each layer.

- **Activation Functions**: The ReLU activation function was used for its non-linear properties and computational efficiency, allowing models to learn complex patterns in the data. The `softmax` function in the output layer calculates a probability distribution over the target classes.

Model Results Summary

- **Model 1**: Exhibited a high training accuracy, but also displayed signs of overfitting as evidenced by the larger gap between training and validation accuracy. It achieved a higher accuracy on the test set, suggesting that the network architecture and regularization strategies were effective to a certain extent.

- **Model 2**: Converged faster than Model 1, possibly due to the deeper layer architecture, which might have captured the necessary features more quickly. However, it suffered slightly in generalization as seen by the lower test accuracy.

Conclusion

Through this project, I've come to understand the delicate balance between model complexity and practical performance. The importance of meticulous evaluation was highlighted, revealing that true model strength lies in its ability to generalize, not just fit to training data. This underscores the nature of designing models that are both effective and efficient, a balance that is important in the real-world application of machine learning.

Experiencing the transition from theory to practice deepened my comprehension of neural networks and their quirks. It was a hands-on lesson in patience and precision, teaching me to scrutinize beyond surface-level metrics. This subject has been thus great in shaping my approach to problem-solving, emphasizing the need for a holistic view that encompasses both the technical and practical aspects of it.

As I move forward, I look forward to explore more nuanced machine learning strategies, drawing from the lessons of this assignment and feedback given by my lecturers to enrich my future work in the field. My journey in machine learning is far from over. This project has sparked a deeper curiosity and a desire to delve into more complex and challenging problems, pushing the boundaries of what I can achieve.