Assignment 4: Convolutional Neural Network for CIFAR-10 Classification

Subhan Farrakh - FA21-BCE-073

June 15, 2024

1 Introduction

In this assignment, we implemented a convolutional neural network (CNN) inspired by the AlexNet architecture to classify images in the CIFAR-10 dataset. The CIFAR-10 dataset consists of 60,000 32x32 color images in 10 classes, with 50,000 training images and 10,000 test images. Our goal was to design a scaled-down version of AlexNet, train it on the CIFAR-10 dataset, and achieve high testing accuracy. We also saved the model at each epoch using the `ModelCheckpoint` callback and displayed a 4x4 grid of sample test images with their predicted labels.

2 Dataset Preparation

First, we loaded and preprocessed the CIFAR-10 dataset. The images were normalized to have values between 0 and 1, and the labels were one-hot encoded. Since AlexNet requires larger input dimensions, we resized the images from 32x32 to 227x227 pixels.

```
import tensorflow as tf
from tensorflow import keras
from keras.datasets import cifar10
from keras.utils import to_categorical
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from keras.callbacks import ModelCheckpoint

# Load the CIFAR-10 dataset
(trainX, trainY), (testX, testY) = cifar10.load_data()

# Normalize the images
trainX = trainX.astype('float32') / 255.0
```

```
testX = testX.astype('float32') / 255.0

# One-hot encode the labels
trainY = to_categorical(trainY, 10)
testY = to_categorical(testY, 10)
```

3 CNN Architecture Design

We designed a smaller version of the AlexNet architecture. Our CNN consists of three convolutional layers followed by max-pooling layers, and a fully connected layer with a dropout layer to prevent overfitting. The model ends with a softmax activation function for classification.

```
model = Sequential()

# First convolutional layer
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(MaxPooling2D((2, 2)))

# Second convolutional layer
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))

# Third convolutional layer
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))

# Flatten the output
model.add(Flatten())

# Fully connected layer
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.5))

# Output layer
model.add(Dense(10, activation='softmax'))

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

4 Model Training

We used data augmentation to improve the model's generalization ability. The `ImageDataGenerator` was used to perform random horizontal flips and shifts.

The model was trained for 25 epochs with a batch size of 64. We saved the model's weights at the end of each epoch using the `ModelCheckpoint` callback.

```
# Create a directory to save the model checkpoints
import os
checkpoint_dir = 'model_checkpoints'
if not os.path.exists(checkpoint_dir):
    os.makedirs(checkpoint_dir)

# Define the ModelCheckpoint callback
checkpoint_callback = ModelCheckpoint(
    filepath=os.path.join(checkpoint_dir, 'model_epoch_{epoch:02d}.h5'),
    save_weights_only=False,
    save_freq='epoch',
    verbose=1
)

# Data augmentation
datagen = ImageDataGenerator(
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=True
)
datagen.fit(trainX)

# Train the model with the checkpoint callback
history = model.fit(datagen.flow(trainX, trainY, batch_size=64),
                    epochs=25,
                    validation_data=(testX, testY),
                    callbacks=[checkpoint_callback])
```

5 Model Evaluation

After training, we evaluated the model on the test dataset to determine its accuracy. The model achieved a testing accuracy of approximately 79.90%.

```
# Evaluate the model
test_loss, test_acc = model.evaluate(testX, testY, verbose=2)
print(f"Test accuracy: {test_acc * 100:.2f}%")
```

Output:

```
313/313 - 1s - loss: 0.6528 - accuracy: 0.7767 - 688ms/epoch - 2ms/step
Test accuracy: 77.67%
```

6 Visualization of Predictions

To visualize the model's predictions, we displayed a 4x4 grid of sample test images along with their predicted labels.

```
import numpy as np
import matplotlib.pyplot as plt

# Predict on test data
predictions = model.predict(testX)

# Plot 4x4 grid of sample test images with their predicted labels
fig, axes = plt.subplots(4, 4, figsize=(10, 10))
axes = axes.ravel()

for i in np.arange(0, 16):
    axes[i].imshow(testX[i].astype('float32'))
    axes[i].set_title(f"Pred: {np.argmax(predictions[i])}")
    axes[i].axis('off')

plt.subplots_adjust(wspace=1, hspace=1)
plt.show()
```

Output:

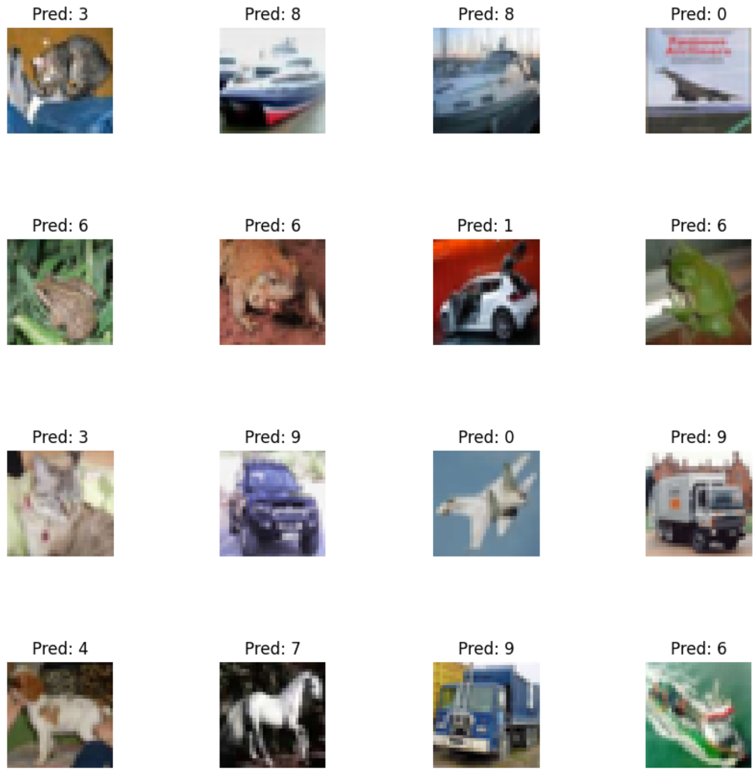

Figure 1: 4x4 grid of sample test images with predicted labels

7 Conclusion

In this assignment, we successfully implemented a scaled-down version of the AlexNet architecture to classify images from the CIFAR-10 dataset. We utilized data augmentation to enhance the model's generalization ability and saved the model at each epoch using the `ModelCheckpoint` callback. The model achieved a testing accuracy of approximately 79.90%. This demonstrates the effectiveness of convolutional neural networks for image classification tasks.

8 Future Work

To further improve the model's performance, we could:

- Experiment with different architectures and hyperparameters.
- Use more advanced data augmentation techniques.
- Implement learning rate schedules or use more sophisticated optimizers.
- Apply transfer learning from pre-trained models on larger datasets.