Image Classification using Convolutional Neural Networks on CIFAR-10 Dataset

- **Author:** Arshia Goshtasbi
- **GitHub:** @Arshiagosh

Introduction

Image classification is a fundamental task in computer vision and machine learning, where the goal is to assign a class label to an input image. One of the most popular and well-studied datasets for image classification is the CIFAR-10 dataset, which consists of 60,000 32x32 color images distributed across 10 classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck.

In this report, we will focus on classifying images from the CIFAR-10 dataset using Convolutional Neural Networks (CNNs) implemented with the Keras library. CNNs have proven to be highly effective for image classification tasks due to their ability to learn and extract meaningful features from the input images.

Dataset and Preprocessing

The CIFAR-10 dataset is already divided into a training set of 50,000 images and a test set of 10,000 images. Before feeding the data into the neural network, we perform the following preprocessing steps:

1. **Normalization**: The pixel values of the images are normalized to a range of 0 to 1 by dividing them by 255 (the maximum pixel value). This step helps in faster convergence during training.
2. **One-Hot Encoding**: The class labels are converted into a one-hot encoded format, which is a vector representation where each class is represented by a binary value (0 or 1) at a specific index position.
3. **Data Augmentation**: To increase the diversity of the training data and improve the model's generalization ability, we apply various data augmentation techniques such as rotation, horizontal and vertical shifts, flipping, zooming, and brightness adjustments.

Model Architecture

For the image classification task, we design a Convolutional Neural Network (CNN) using the Keras Sequential model. The architecture consists of the following layers:

1. **Convolutional Layers**: Multiple convolutional layers with 3x3 kernels, ReLU activation functions, and batch normalization.
2. **Max Pooling Layers**: Max pooling layers are used to downsample the feature maps and introduce translation invariance.
3. **Dropout Layers**: Dropout layers are added after convolutional and max pooling layers to prevent overfitting.
4. **Flatten Layer**: A flatten layer is used to convert the multi-dimensional feature maps into a 1D vector.
5. **Dense Layer**: A fully connected dense layer with a softmax activation function for multi-class classification.

The specific number of layers, filters, and other hyperparameters can be tuned and experimented with to achieve the highest accuracy.

Model Training

The model is compiled with the Adam optimizer, categorical cross-entropy loss function, and accuracy metric. We employ callbacks such as ReduceLROnPlateau to dynamically adjust the learning rate during training and EarlyStopping to stop the training if no improvement is observed for a certain number of epochs.

The model is then trained on the preprocessed CIFAR-10 training data for a specified number of epochs, using the data augmentation generator. During training, we monitor the training and validation loss and accuracy to ensure the model is learning effectively.

Results and Evaluation

After training, we evaluate the model's performance on the unseen test data and report the test accuracy and loss. Additionally, we plot the learning curves, showing the training and validation loss and accuracy over the epochs.

The trained CNN model achieved an accuracy of **82%** on the CIFAR-10 test dataset.

To further analyze the model's performance, we generate and display some sample predictions using the Matplotlib library. This includes showing the input images along with their corresponding predicted labels by the model.

```
# Sample code for displaying predictions
import matplotlib.pyplot as plt

# Get a batch of test images and labels
test_images, test_labels = X_test[:10], y_test[:10]
```

```python
# Make predictions on the test images
predictions = model.predict(test_images)

# Get the class labels
class_names = ['airplane', 'automobile', 'bird',
               'cat', 'deer', 'dog', 'frog',
               'horse', 'ship', 'truck']

# Plot the images and predicted labels
fig, axes = plt.subplots(nrows=2, ncols=5, figsize=(15, 6))
for i, ax in enumerate(axes.flat):
   ax.imshow(test_images[i])
   predicted_label = class_names[np.argmax(predictions[i])]
   true_label = class_names[np.argmax(test_labels[i])]
   ax.set_title(f"Predicted: {predicted_label}\nTrue: {true_label}")
   ax.axis('off')
plt.show()
```

This visualization helps us qualitatively assess the model's performance and identify potential areas for improvement.

Conclusion and Future Work

In this report, we demonstrated the process of training a Convolutional Neural Network for image classification on the CIFAR-10 dataset using the Keras library. We covered the essential steps, including data preprocessing, model architecture design, training, and evaluation.

To further improve the model's performance, we can experiment with different architectures, hyperparameters, and regularization techniques. Additionally, employing more advanced data augmentation methods or generating synthetic data can help increase the dataset's diversity and enhance the model's generalization ability.

Furthermore, we can explore transfer learning techniques by leveraging pre-trained models on larger datasets, such as ImageNet, and fine-tuning them on the CIFAR-10 dataset. This approach can potentially yield better results and faster convergence.

Overall, this project serves as a stepping stone for understanding and applying Convolutional Neural Networks to real-world image classification problems, and the techniques learned can be extended to other computer vision tasks.