Calvin James Tyler

UPB

HOMEWORK ASSIGNMENT #1

The One with the CNNs

# Description

This report covers the implementation and testing of various convolutional neural network (CNN) components and tasks, aiming to enhance understanding and proficiency in building and fine-tuning CNNs. The assignment is structured into three main tasks, each focused on specific aspects of CNN operations and architectures.

# Task 1: Implementation of Convolutional Layer Forward Pass

Objective: Implement the forward pass of a custom convolutional layer, handling key parameters such as kernel size, input and output channels, stride, dilation, and grouping.

## 1. Implementation Details:

MyConvStub: This convolution operation integrates parameters including kernel size, number of input and output channels, bias, stride, dilation, and groups. Each parameter was configured to allow flexible convolution operations, accommodating for different types of inputs.

MyFilterStub: A 2D blur filter was applied across all input channels independently. Each channel was blurred using the same filter to ensure consistency.

## 2. Testing:

- Conducted unit tests from `test\_conv.py` to verify the accuracy of the forward pass.

- Confirmed correct application of the blur filter to input volumes, ensuring appropriate convolutional transformations.

Results: Both convolution operations were validated, with tests confirming that the forward pass is accurately implemented.

# Task 2: Developing and Training a Simple CNN for Image Classification

Objective: Build, train, and test a basic CNN for classifying the Imagenette dataset, which includes 10 classes, with a model depth limit of five layers.

## 1. Model Design:

Activation Functions and Pooling Layers

In this CNN architecture, the following activation functions and pooling layers were chosen:

ReLU Activation Function:

Justification: ReLU (Rectified Linear Unit) is applied after each convolutional and hidden fully connected layer to introduce non-linearity and mitigate the vanishing gradient problem. ReLU’s computational efficiency and ability to zero out negative values also lead to sparse activations, which improves learning and generalization.

Usage: ReLU is used after each convolutional layer and in the first fully connected layer to allow the network to learn complex, non-linear relationships between features.

Max Pooling:

Justification: Max pooling is applied after each ReLU activation to reduce the spatial dimensions of feature maps. This operation retains the most prominent features, making the model more robust to small positional shifts and reducing the computational burden of subsequent layers.

Usage: A 2×2 max pooling layer is used following each convolutional layer, halving the dimensions of the feature maps while preserving dominant features.

Softmax Activation Function:

Justification: Softmax is applied in the output layer to generate a probability distribution over the target classes, enabling the model to make clear, interpretable multi-class predictions.

Usage: Softmax is applied to the final output layer, providing class probabilities for the classification task.

These choices optimize the network’s performance by balancing computational efficiency with robust feature extraction and enabling effective multi-class predictions.

## 2. Experimental Training:

- Batch Normalization: Compared training with and without batch normalization to assess its impact on convergence and model stability.

- Dropout Regularization: Applied dropout in the final linear layers to reduce overfitting and improve generalization.

- Data Augmentation: Utilized techniques such as horizontal flipping and color jittering to enhance the dataset’s diversity.

## 3. Results Analysis:

- Loss and Accuracy Curves:

- Plotted separate graphs for each experimental condition (batch normalization, dropout, and data augmentation) to visualize training/test loss and accuracy.

- Confusion Matrix: Created a confusion matrix for the 10 classes to analyze the model’s performance per class.

Overall, training with all three methods (normalization, regularization, and augmentation) yielded the best performance, surpassing the 60% accuracy target on test data.

# Task 3: Fine-Tuning a Pretrained CNN (ResNet-18) for Image Classification

Objective: Fine-tune the ResNet-18 model using transfer learning techniques, with a focus on adapting Batch Normalization layers to the new dataset.

## 1. Implementation:

- Loaded a pre-trained ResNet-18 model as the backbone.

- Followed PyTorch’s transfer learning framework, using ResNet-18 as a feature extractor.

- Experimented with unfreezing Batch Normalization layers to adapt mean and standard deviation per batch to the Imagenette dataset.

## 2. Results Analysis:

- Loss and Accuracy Curves: Documented changes in training and test loss as well as accuracy to measure the effectiveness of fine-tuning.

- Confusion Matrix Comparison: Compared the model’s confusion matrix with the one generated in Task 2.

- Impact of Batch Normalization Unfreezing: Evaluated whether unfreezing the Batch Normalization layers improved or hindered performance, providing insights into how normalization impacts model adaptability.

Fine-tuning the Batch Normalization layers led to [indicate findings, e.g., improved performance due to adapted feature scaling, or minimal impact if no significant changes were noted].

# Summary

This assignment demonstrated essential CNN concepts and techniques, from implementing basic convolution operations to designing and training a CNN and fine-tuning a pre-trained model. These tasks reinforced understanding in CNN architecture, parameter tuning, and the advantages of transfer learning.