Image Classification Using Fine-Tuned VGG19

Group 4:
Hardik Singh (2101MC19)
Mohd Darish Khan (2101MC29)
Suryansh Jaiswal (2101MC41)

September 17, 2024

1 Introduction

This report summarizes the implementation of a deep learning model for image classification using the pre-trained VGG19 model in PyTorch. The task involves classifying images into 5 distinct categories. The primary objective is to fine-tune the model's final classification layer and evaluate its performance on a training and test set.

2 Model Architecture

We used a pre-trained VGG19 model from the `torchvision` library. To adapt the model to our task, we modified the final classification layer to output 5 classes, keeping the rest of the network's weights frozen. The detailed model architecture is shown in Figure **??**.

3 Training and Validation

The model was trained for 25 epochs using the Adam optimizer and cross-entropy loss. The training data underwent preprocessing transformations including resizing to 224x224, random horizontal flips, normalization using the ImageNet statistics, and conversion to tensors.

3.1 Training Results

The training process yielded the following results:

- Final Training Loss: 0.0008
- Final Training Accuracy: 100%

3.2 Validation Results

Similarly, the validation data evaluation yielded:

- Final Validation Loss: 0.0067
- Final Validation Accuracy: 100%

4 Test Results

After training, the model was evaluated on the test set, achieving an accuracy of 98%. The confusion matrix (Figure **??**) demonstrates the model's strong performance across all classes.

```
VGG(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace=True)
    (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU(inplace=True)
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (6): ReLU(inplace=True)
    (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (8): ReLU(inplace=True)
    (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (13): ReLU(inplace=True)
    (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (15): ReLU(inplace=True)
    (16): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (17): ReLU(inplace=True)
    (18): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (19): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (20): ReLU(inplace=True)
    (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (22): ReLU(inplace=True)
    (23): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (24): ReLU(inplace=True)
    (25): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (26): ReLU(inplace=True)
    (27): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (29): ReLU(inplace=True)
    (30): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (31): ReLU(inplace=True)
    (32): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (33): ReLU(inplace=True)
    (34): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (35): ReLU(inplace=True)
    (36): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(7, 7))
  (classifier): Sequential(
    (0): Linear(in_features=25088, out_features=4096, bias=True)
    (1): ReLU(inplace=True)
    (2): Dropout(p=0.5, inplace=False)
    (3): Linear(in_features=4096, out_features=4096, bias=True)
    (4): ReLU(inplace=True)
    (5): Dropout(p=0.5, inplace=False)
    (6): Linear(in_features=4096, out_features=5, bias=True)
  )
)
```

Figure 1: Figure 1: VGG19 Model Architecture with modified classification layer

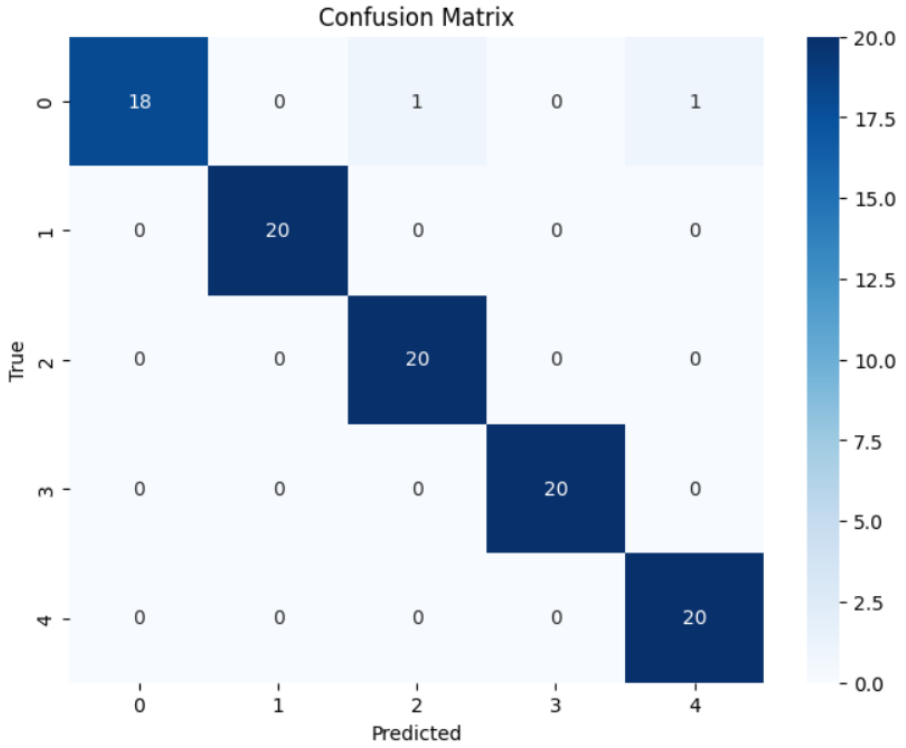

Figure 2: Figure 3: Confusion Matrix of Test Set

5 Conclusions

The model performed exceptionally well, achieving 100% training and validation accuracy, with a 98% accuracy on the test set. The results suggest that the model successfully learned to generalize the image classification task, though the high training accuracy may indicate potential overfitting. Further evaluation using techniques like cross-validation or testing on a larger dataset may provide deeper insights into the model's generalization.