Haidi Azaman (A0216941E)

Kho Tze Jit (A0215110E)

Objective:

This project aims to implement image classification techniques covered in the course, focusing on developing an application customized for scene categorization.

Dataset:

The training dataset comprises 1,500 images, evenly distributed across 15 classes. Each class contains 100 images, depicting scenes such as Kitchen, Bedroom, and Office. To facilitate model training, we partition the dataset into training and validation sets using a 90-10 split.

For the training set, we employ various data augmentation techniques to enhance model robustness, while the validation set undergoes only the ToTensor() operation. Further details regarding the data augmentation methods will be provided later in the report.

Method

Data Preprocessing

After loading the images, we process the image paths using the cv2.imread library, specifying the cv2.IMREAD_GRAYSCALE argument to ensure that the images are loaded in grayscale. Our training script was developed from scratch, starting with the definition of a dataset class. In the getitem method of this class, we load an image from the image path in real time and apply transformations on the fly. We chose a batch size of 256 for our dataset and resized our images to 224x224 pixels.

Model Architecture - ResNet50

For our model architecture, we opted to use ResNet50 (He et. al, 2016). ResNet50 is a convolutional neural network architecture introduced by Microsoft Research. It consists of 50 layers, including convolutional layers, batch normalization layers, activation functions, and skip connections (or shortcuts). The skip connections enable the model to learn residual mappings, which helps alleviate the vanishing gradient problem and allows for the training of deeper networks.

ResNet50 has several strengths:

- Deep Architecture: ResNet50 is a deep neural network with 50 layers, allowing it to learn complex features from input images.
- Residual Connections: The use of residual connections helps mitigate the vanishing gradient problem, making it easier to train deep networks. These connections also facilitate the training of deeper models without encountering degradation in performance.
- Pre-trained Weights: ResNet50 is often used as a pre-trained model on large datasets such as ImageNet. Pre-training on such datasets allows the model to learn rich feature representations, which can be fine-tuned on smaller datasets for specific tasks like scene classification. **However, for our implementation we did not use the pretrained weights as per assignment requirements and instead trained it from scratch.**
- State-of-the-art Performance: ResNet50 has demonstrated state-of-the-art performance on various image classification tasks, making it a popular choice for many applications.

By using ResNet50 as our model architecture (Figure 1), we leverage its depth and skip connections to learn intricate patterns in the image data, which can lead to improved performance in scene classification tasks.

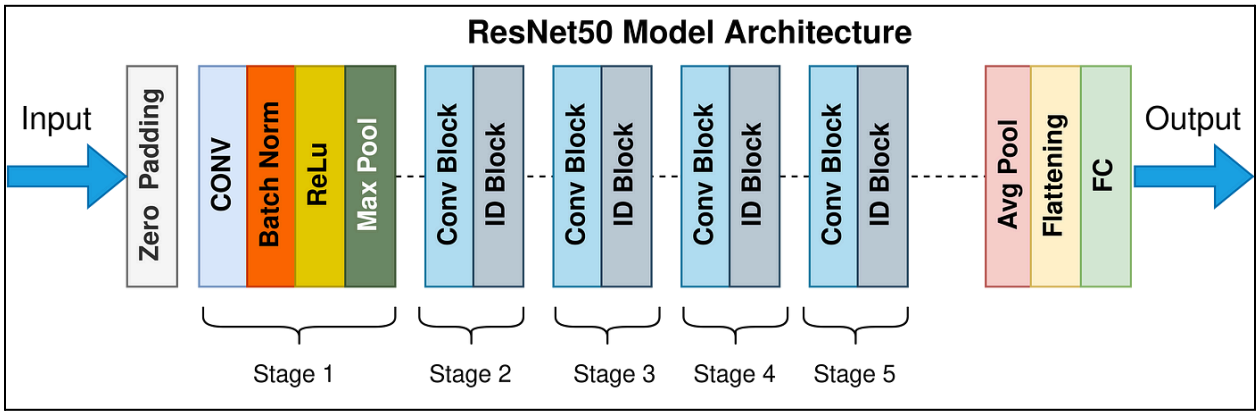

Figure 1: Model architecture of ResNet50

Model Modifications

To adapt the ResNet50 model from torchvision, originally designed for RGB input and 1000-dimensional output, we made specific modifications. Firstly, we altered the initial layer of the model to accommodate grayscale input by tweaking the input channels for the 1st Conv2d layer to 1, down from the usual 3. Subsequently, we adjusted the final classifier layer to address our 15-class classification problem, converting it to an nn.Linear layer with 15 output neurons.

The ResNet50 model used was based on the implementation found here: https://github.com/JayPatwardhan/ResNet-PyTorch/blob/master/ResNet/ResNet.py

This implementation is a lot less bloated than the default pytorch implementation leading to an approximately 40% faster training time per epoch.

Model Training

Due to the large model size and number of weights, training could not be done efficiently locally or on free online platforms like Google Colab. Instead, the model was trained using NUS HPC resources.

Implementation trick

- Early Stopping: Early stopping is a technique used during model training to prevent overfitting. It involves monitoring the model's performance on a validation set and stopping the training process when the performance starts to degrade, indicating that further training may not lead to improved results.
- Retraining with Full Training Set: After identifying the optimal epoch using early stopping on a validation set, retraining the model with the full training set allows the model to learn from the entire dataset and potentially improve its performance further.
- Data Augmentation: Data augmentation is a technique used to artificially increase the size of the training dataset by applying various transformations to the existing data samples. This helps improve the model's generalization and robustness by exposing it to a wider range of variations in the input data.
- LR scheduler, ReduceLROnPlateau: this scheduler dynamically adjusts the learning rate during training based on the validation loss plateau.
    - Plateau Detection: If the validation loss stays the same (or stops getting better), it means the model might not be improving anymore. Reducing the learning rate further could help the model get unstuck.
    - Patience: This tells the scheduler how long to wait before making a decision. It waits for a certain number of epochs without seeing any improvement before taking action.

- Learning Rate Reduction: When the scheduler sees that the validation loss hasn't improved for a while (as determined by patience), it decreases the learning rate by a certain factor. This helps the model adjust its steps in the right direction.
- Adaptive Adjustment: The scheduler keeps watching the validation loss. If it's still not improving, the scheduler reduces the learning rate even more. This helps the optimizer find its way through the loss landscape, hopefully leading to better results.

Types of Data Augmentation:

a. Random Horizontal Flip: This transformation horizontally flips the image with a certain probability, introducing variations in object orientation.
b. Random Rotation: Rotates the image by a random angle within a specified range, simulating variations in object orientation and viewpoint.
c. Random Invert: Inverts the colors of the image randomly, introducing variations in color appearance.
d. Gaussian Blur: Applies Gaussian blurring to the image, introducing variations in image smoothness and reducing noise.

Example code:

```
tfms = transforms.Compose([
    transforms.ToTensor(),
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation((-30, 30)),
    transforms.RandomInvert(),
    transforms.GaussianBlur(kernel_size=3, sigma=(0.1, 2.0)),
])
image = tfms(image)
```

References

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).