Transfer Learning Approach to Classifying the Intel Image Dataset

Shakuntala Mitra

February 19, 2025

1 Experiments

1.1 Experiments on Intel Image Classification Dataset

The Intel Image Classification Dataset is a dataset of approximately 25,000 images distributed under 6 classes. The 6 classes are:

'building': 0, 'forest': 1, 'glacier': 2 , 'mountain': 3 , 'sea': 4, 'street': 5

Each image has the shape 150x150x3, where the last dimension indicates the number of color channels (RGB images, so channels=3). There are about $14,000$ images in the Train set and $3,000$ images in the Test set. To create the Validation set, the Train set was split using an $80:20$ ratio, as in train : validation.

For transfer learning, I chose to use ResNet-18, which was pre-trained on the ImageNet dataset. Thus, all images were resized to 224x224x3 for compatibility with CNN models pre-trained on ImageNet. The pixel values in each of the three color channels were normalized using the mean and standard deviations of the pixel values from the ImageNet dataset. Input normalization helps with faster convergence and stable training.

Furthermore, to increase the robustness and generalizability of the model, some image transformations were applied as data augmentations to the training and test datasets. The chosen data augmentations were random horizontal and vertical flips, with a 50% chance of each flip being applied, and a color jitter.

1.2 Neural Net Architecture

The neural network architecture of ResNet-18 has 18 total layers, as shown in the following diagram.

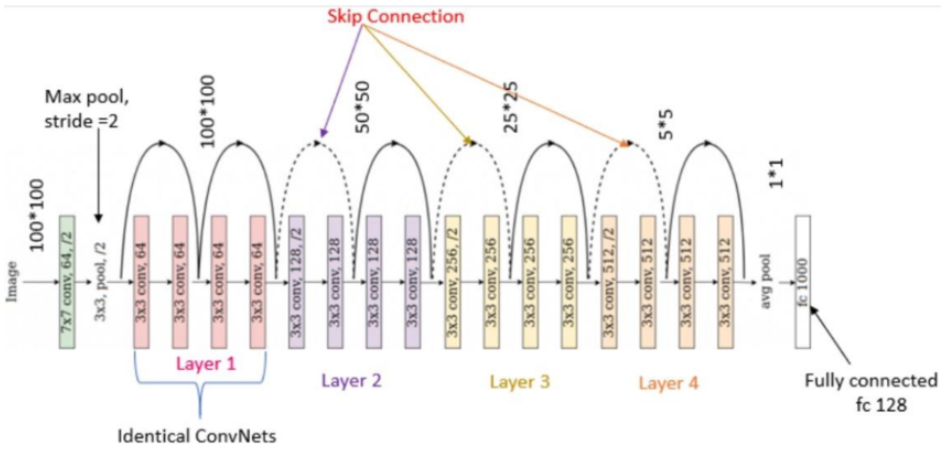

Figure 1: Original ResNet-18 Architecture

ResNet-18 uses Rectified Linear Unit (ReLU) activation units as well as batch normalization. The ReLU activation function introduces non-linearity by outputting the input directly if it's positive, and otherwise outputting 0. The function is defined as $f(x) = max(0, x)$. ReLU is useful because it allows ResNet to have a deeper structure so as to learn more complex patterns efficiently, while avoiding the vanishing gradient problem that came with using earlier activation functions such as sigmoid or tanh.

1.3 Comparing Transfer Learning Methods

Two transfer learning methods were compared: using fixed feature extractors vs fine-tuning the entire ResNet-18 model. The initial weights used for ResNet-18 were the default weights provided by Torch Hub, which came from the pre-training on the ImageNet dataset. Of course, I manually set the seed for PyTorch and NumPy to seed the RNG for all devices and ensure a level of reproducibility for the experiments.

The one hyperparameter which differed between the two transfer learning methods was the learning rate, which is used by the Adam optimizer. For the fixed feature extractor method, I set the learning rate to $1e^{-3}$. The slightly higher learning rate is acceptable since only the last layer is being fine-tuned during the training loop. For the full fine-tuning method, I set the learning rate to $1e^{-4}$. The lower learning rate is to avoid catastrophic forgetting.

Comparing the results of both transfer learning methods, both had validation accuracy not greater than 20%. Since there are six output classes, a naive classifier would yield an accuracy of 16.67%. The fixed feature extraction method achieved a validation accuracy of 12%, and fine-tuning the entire model achieved a validation accuracy of 20%. Thus, both methods had low or comparable accuracy to a naive classifier after training. However, this is most likely due to the low number of training epochs (only 10 epochs). I only trained for 10 epochs since I, regrettably, spent the majority of my time debugging code and CUDA errors. When fine-tuning the entire model, I noticed a larger percent increase in the validation accuracy over the 10 epochs than when only the last layer was fine-tuned. While more empirical evidence is needed to draw solid conclusions, I would expect that fine-tuning the full model would perform better than fine-tuning just the last layer.

The fixed feature extraction method, where only the last layer is fine-tuned, is typically enough to get good performance for small datasets. Fine-tuning the full model is more appropriate when using a very large, complex dataset. Since the Intel Image dataset is very large, I would expect that the method of fine-tuning the entire ResNet-18 model would yield better results.

1.4 Ideas for Further Experimentation

The validation accuracies achieved were inconclusive due to the low number of training epochs, so for future development, I would try experimenting with increasing the number of training epochs to see the impact on the validation accuracy. The number of epochs could be tuned in the same way as any other hyperparameter.

Additional data augmentations, such as adding noise, grain, rotations, affine transformations, etc. could benefit the performance of the model by increasing the diversity of the training dataset. One important note is that I applied the data augmentations in my experiments to both the training and test datasets, and therefore to the validation dataset as well. Conceptually, this does not make sense - data augmentations should only be applied to the training dataset, not to the test or validation datasets. This could have been avoided by splitting the training set into train vs validation first, and then applying the transformations only to the training subset.

A larger ResNet architecture might suit this task better. ResNet-18 is the most lightweight of all the ResNet model architectures. This means that it has the fewest parameters and faster training time, in exchange for a tradeoff in accuracy and resolution of detected features compared to ResNet-34 or ResNet-50. I would be curious to compare the performance of ResNet-18 vs ResNet-34 vs ResNet-50 for the Intel Image Classification dataset.