Deep Learning in Computer Vision - Lab 1 Report

Rohin Andy Ramesh
University of Bordeaux
Course: IPCV, 4TTV911U

September 27, 2024

1 Introduction

In this task, we focused on classifying images of buildings into three architectural styles: Colonial, Modern, and Prehispanic. To accomplish this, we fine-tuned a pre-trained ResNet-18 model, leveraging its ability to extract rich features while adapting it for our specific classification task. The dataset, which contained images of the three architectural styles, was preprocessed by applying standard techniques like resizing, normalization, and data augmentation to enhance model generalization and prevent overfitting. During training, we monitored performance using metrics such as training and validation accuracy, loss curves, and confusion matrices. We also conducted a thorough misclassification analysis to understand areas where the model struggled. By analyzing these metrics, we gained insights into the model's performance and identified opportunities for improvement. This approach allowed us to evaluate the model's effectiveness in classifying the architectural styles with PyTorch as our deep learning framework.

2 Dataset

The dataset consists of images from three architectural styles: Colonial, Modern, and Prehispanic. There are 236 images in the training set and 48 in the validation set. Labels are derived from filenames and mapped to numerical classes during preprocessing. The function `split_data(train_dataset_path, test_dataset_path)` is used to generate a dictionary containing filenames and their associated labels for efficient dataset handling.

3 DataLoaders

The function `get_dataloaders(train_class_names, test_class_names)` is designed to prepare and return data loaders for training and validation datasets. It applies specific transformations to the images in both datasets: random resized cropping, horizontal flipping, and normalization for the training set, and resizing, center cropping, and normalization for the validation set.

After applying these transformations, the function initializes the datasets using a custom dataset class (CustomDataset), which takes the transformed images and their labels. Finally, data loaders are created for both the training and validation sets, using

a batch size of 4, with shuffling enabled for training and disabled for validation. The function returns these dataloaders, which will be used to feed data into the model during training and evaluation.

4 Transfer learning

The `train_model(dataloaders, n_epochs)` function is designed to train a ResNet-18 model, fine-tuning it for the task of architectural style classification. The function initializes a pre-trained ResNet-18 network and replaces the final fully connected layer (`net.fc`) to output 128 features (adaptable to 3 classes later). The model is transferred to the GPU if available.

The training process uses the **CrossEntropyLoss** as the loss function and the **SGD optimizer** with a learning rate of 0.0001 and momentum of 0.9. It defines an `accuracy` function that computes the correct predictions during both training and validation.

The training loop iterates over `n_epochs`, where for each epoch, it:

- Performs forward passes, computes the loss, and updates model parameters using backpropagation.
- Tracks running loss and accuracy after each batch of training data.
- Prints updates after every 20 batches, reporting the current loss.

After each epoch, validation is performed using the validation dataset, where the loss and accuracy are recorded. If the validation loss improves, the model is saved to a file (`resnet.pt`) indicating improvement.

The function returns the training and validation losses and accuracies as well as the trained model and the loss function, useful for further evaluation and analysis.

- Pretrained network: ResNet-18
- Final Layer: Linear layer with 128 output neurons
- Optimizer: SGD with learning rate 0.0001 and momentum 0.9
- Loss function: Cross-Entropy Loss

5 Model Training and Evaluation Pipeline

In this section, the model training and evaluation pipeline is outlined using several functions that work together to train the model, visualize results, and analyze performance.

First, the training and validation datasets are initialized using paths specified for the training and testing images: `split_data(train_dataset_path, test_dataset_path)` is called to split the dataset into dictionaries containing image filenames and corresponding labels for both the training and validation sets. These are then passed to the `get_dataloaders()` function, which returns the dataloaders for training and validation.

The model is trained using the `train_model()` function, which takes the dataloaders and the number of epochs (in this case, 50). It returns:

- `val_loss`: Validation loss
- `val_acc`: Validation accuracy
- `train_loss`: Training loss
- `train_acc`: Training accuracy
- `net`: The trained model
- `criterion`: The loss function used

After training, three functions are used for evaluation:

- `plotcurves(train_acc, val_acc, train_loss, val_loss)`: Plots the training and validation accuracy and loss curves over the epochs, providing insight into how the model's performance evolves.
- `plotcfmat(dataloaders, net)`: Plots the confusion matrix for the validation set, offering a visual breakdown of correct and incorrect classifications across the architectural styles.
- `plotmisclassified(dataloaders, net, criterion)`: Visualizes misclassified images from the validation set, helping to understand where the model is making mistakes and providing a basis for improving performance.

6 Results and Evaluation

6.1 Training and Validation Curves

The accuracy and loss curves for both training and validation sets are shown in Figures 1 and. We observed an improvement in both metrics over the epochs, and the model shows signs of convergence.

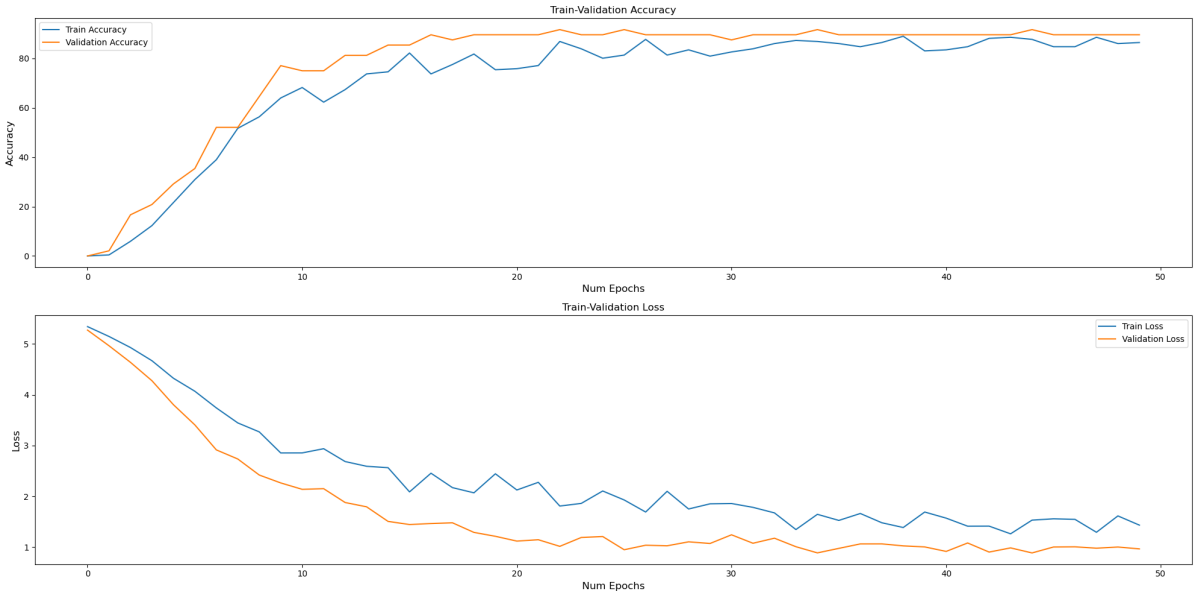

Figure 1: Accuracy and Loss Curve

6.2 Confusion Matrix

To further evaluate the model's performance, we generated a confusion matrix . The matrix shows the number of correct and incorrect predictions for each class. The model performed well on most categories, although some confusion between certain styles was observed.

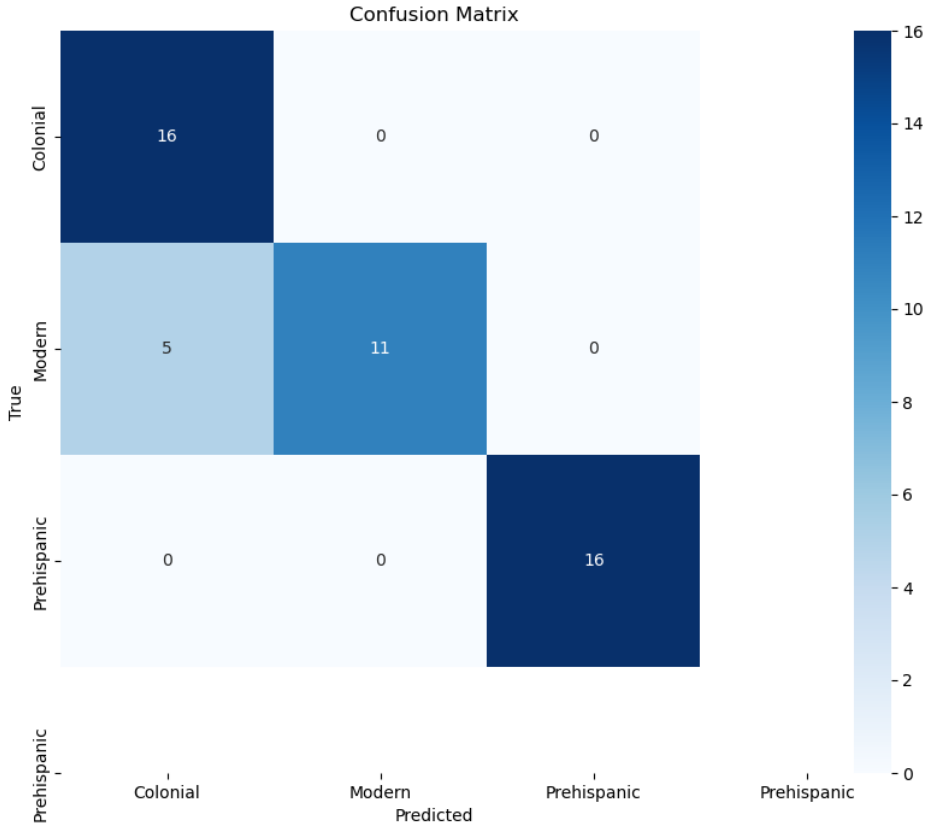

Figure 2: Confusion Matrix

7 Conclusion

In this lab, we successfully fine-tuned a pre-trained ResNet-18 model for the task of classifying architectural styles. The model achieved reasonable accuracy on both the training and validation sets, and the results were visualized using accuracy/loss curves and a confusion matrix. Future work could involve more advanced data augmentation, hyperparameter tuning, and possibly experimenting with different architectures or ensemble methods.

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 (CVPR), 770–778.

- Deng, J., Dong, W., Socher, R., Li, L., Li, K., & Fei-Fei, L. (2009). ImageNet: A large-scale hierarchical image database. In 2009 IEEE Conference on Computer Vision and Pattern Recognition, 248–255.

- Liu, S., Tian, G., Xu, Y. (2019). A novel scene classification model combining ResNet based transfer learning and data augmentation with a filter. Neurocomputing (Amsterdam), 338, 191–206.

- Niu, S., Liu, Y., Wang, J., Song, H. (2020). A Decade Survey of Transfer Learning (2010-2020). IEEE Transactions on Artificial Intelligence, 1(2), 151–166. https://doi.org/10.1109/TAI.2021.3054609