DSA5203 Project Technical Report

Name: Ong Jian Ying Gary, Zhang Jia Jun

Student ID: A0155664X, A0251104Y

Running of Project Code

Ensure that Pytorch 2.0 is installed before running the code. GPU usage to test the functions are highly recommended to speed up the training and testing process.
If the code provided in the zip file does not run as intended, please head to the following Google Drive link to run:
https://drive.google.com/drive/folders/134aqyvo__Ku9zaalIgFIEiIeDGpvKpPA?usp=sharing
Save the test folder in the "data" folder and label the folder "test".
Make use of the "run.ipynb" to perform running of the train and test set.

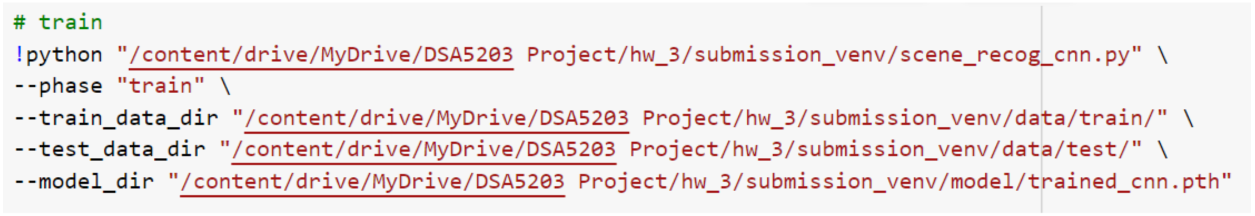
Zip file

Run the command line function as follows for training:
python scene_recog_cnn.py --phase "train" --train_data_dir "./data/train/" --test_data_dir "./data/test/" --model_dir "./model/trained_cnn.pth"

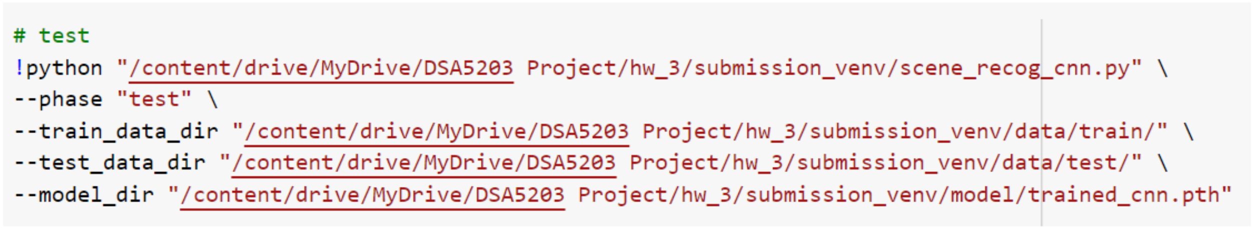
Run the command line function as follows for testing:
python scene_recog_cnn.py --phase "test" --train_data_dir "./data/train/" --test_data_dir "./data/test/" --model_dir "./model/trained_cnn.pth"

Google Colab

Head over to "run.ipynb" and run the following 2 cells:



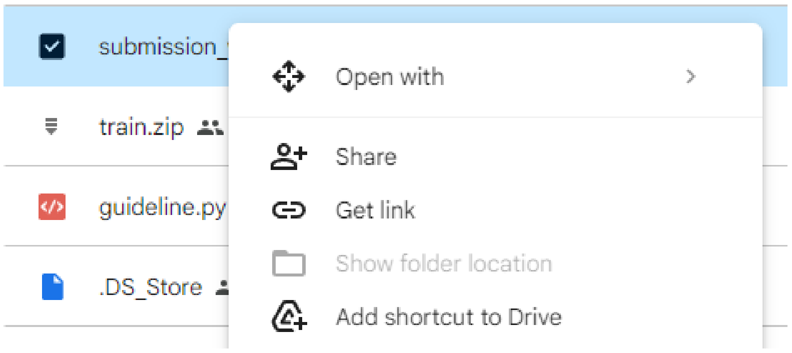
```
# train
!python "/content/drive/MyDrive/DSA5203 Project/hw_3/submission_venv/scene_recog_cnn.py" \
--phase "train" \
--train_data_dir "/content/drive/MyDrive/DSA5203 Project/hw_3/submission_venv/data/train/" \
--test_data_dir "/content/drive/MyDrive/DSA5203 Project/hw_3/submission_venv/data/test/" \
--model_dir "/content/drive/MyDrive/DSA5203 Project/hw_3/submission_venv/model/trained_cnn.pth"
```



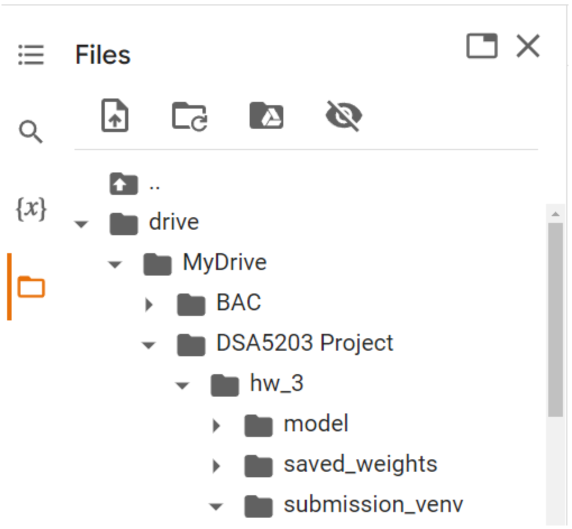
```
# test
!python "/content/drive/MyDrive/DSA5203 Project/hw_3/submission_venv/scene_recog_cnn.py" \
--phase "test" \
--train_data_dir "/content/drive/MyDrive/DSA5203 Project/hw_3/submission_venv/data/train/" \
--test_data_dir "/content/drive/MyDrive/DSA5203 Project/hw_3/submission_venv/data/test/" \
--model_dir "/content/drive/MyDrive/DSA5203 Project/hw_3/submission_venv/model/trained_cnn.pth"
```

Replace "/content/drive/MyDrive/DSA5203 Project/hw_3/submission_venv/" with your own path to the directory that the "run.ipynb" is in.

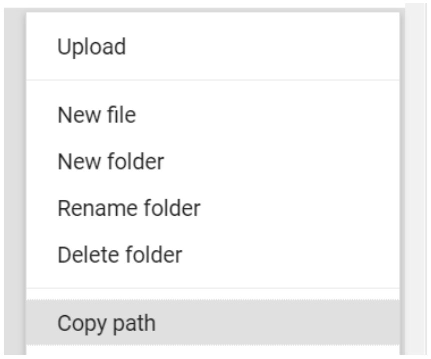
To get the file path, create a shortcut of the shared folder to your Google Drive.



Head back to “run.ipynb”, and locate the “submission_venv” folder.



Right click the “submission_venv” folder and select “Copy path” to get the directory that the “run.ipynb” is in.



To visualize the result of the function, please use the Google Link and head over to “run.ipynb” and run the following commands under the “Visualisation” header.

Technical Report

The project assignment requires 2 main functions to be written, namely train and test functions. The current project is based on the CNN-based implementation for classification of scenes of 15 classes as defined in the train folder given of 1500 images.

The project required a design and implementation of a CNN model. Thus, the methodology chosen was to utilize transfer learning. Transfer learning is a machine learning technique where a model that has been pre-trained on an image classification task is fine-tuned or adapted to perform a different but related task. The idea is to leverage the knowledge and features learned by the pre-trained model from the original task to improve performance on the new task, especially since we are only given a small dataset of 1500 images.

We utilize the ResNet50 model with pretrained weights selected as "IMAGENET1K_V2" from PyTorch library in the project and it offers the following advantages:

- The ResNet pretrained model was previously trained on ImageNet (large dataset) which contains millions of images and thousands of classes. By using the pre-trained weights from the model, the knowledge from the model is leveraged as it has already processed and identified features in a wide spectrum of images. This could improve the performance of the model in the current task.
- There would be faster training, as the pre-trained model can converge to a good solution with high validation accuracy much faster as compared to training a model from scratch, saving time and computational resources, as fewer epochs and smaller learning rates could be used to fine-tune the model.
- There would be better generalization and hence higher validation accuracy, as the pretrained model has already learnt to extract relevant features from a diverse set of images, which can help it generalize to the current new task. With a small dataset of 1500 images, overfitting could be a concern. By using pretrained models, we could mitigate the risk of overfitting and achieve better performance.
- As designing a custom architecture for a deep learning model from scratch can be challenging, instead we used an established architecture like ResNet50 so that the model definition complexity is avoided and fine-tuning the model can be the focal point.

We will now briefly explain the method utilized to develop the train and test functions fulfilling the project requirements. The code is set with a fixed random seed so that it is reproducible.

Firstly, the train() function is required and it is the main function for training the model. It processes the incoming data, constructs the CNN model (to be saved) and completes the training procedures and saves the trained model as trained_cnn.pth. Steps taken to define the function are as listed below:

1. We first loaded the dataset from the train folder given by enumerating the train data directory and getting the list of images and their labels. A pandas dataframe was utilized to store the sample information and their encoded labels. We utilized a 80-20 training-validation split to ensure that we have visibility of how our model is performing during the training process. The dataframe information is utilized to ensure that the samples being split are stratified to ensure class balance in the validation and train dataset.

2. The hyper parameters for the train function are set. This includes the batch size, defining number of classes, number of workers for parallelism, optimizer function (Adam was chosen as the default), number of epochs (set at 100) and learning rate that was trialed and found to be suitable at 0.001.
3. The image dataset class was defined in a separate python script titled 'image_dataset.py', which contains the ImageDataset class, which is used to create a dataset from a list of image file paths and their corresponding labels. The class inherits from the Dataset class in PyTorch and is going to be utilized with PyTorch DataLoader for loading and processing the images during model training and validation.
4. We define some preprocessing steps for the images using the PyTorch transforms module, create custom datasets for training and validation, and set up DataLoader for efficient data loading and processing during training. The trans variables is defined as the image preprocessing pipelines for training and validation. This includes resizing the images to a fixed size (224x224) as ResNet50 was trained on the same sized images to ensure compatibility and converting them to tensors. Then, the images were normalized using the specified mean (0.485, 0.456, 0.406]) and standard deviation ([0.229, 0.224, 0.225]) values, which are specific to the ImageNet dataset on which the ResNet50 model was pre-trained on, this ensures that the new images have similar value distributions and helps ResNet50 to better generalize on the new dataset.
5. The train_set and valid_set were defined and created from instances of the custom ImageDataset class. Thereafter, train_dl and valid_dl were defined as DataLoader objects that are used in training, which can handle tasks like batching and parallel data loading using multiple worker threads.
6. Next, the ImageClassificationBase class is defined to contain methods to handle the training and validation process, as well as output logging. The training_step function defines a single batch of images and labels as input, generates predictions using the current model and calculates the loss using the cross-entropy loss function. Cross-entropy loss function is suitable for a multi-class classification problem. The loss is used to update model weights during the training process. Similarly, the validation_step function takes a batch of images and labels as input during the validation phase. It will generate predictions using the current model, calculate loss and compute accuracy of the predictions. It then returns a dictionary containing both the detached validation loss and accuracy, which are used for monitoring model performance and fine-tuning. Other functions like validation_epoch_end and epoch_end are utilized for logging the average validation loss and accuracy and for output data logging of the training process respectively.
7. Accuracy function is defined for the output labels
8. The ResnetCnnModel class is defined as a custom neural network class for image classification using ResNet50 as a feature extractor, inheriting from ImageClassificationBase class and nn.Module, which provided the methods for handling the training and validation process. We freeze the weights of the pretrained ResNet50 layers so that we use these layers as feature extractors, and their weights will not be updated during training. A custom network using the nn.Sequential module for a model head was attached to ResNet50, which contained the following:

    a. A fully connected layer that takes the output features from pretrained ResNet50 and maps them to 512 features
    b. A ReLu activation function
    c. A dropout layer with 25% dropout rate for regularization purposes to prevent overfitting
    d. Another fully connected layer that maps the 512 features to 256 features
    e. Another ReLu activation function
    f. A dropout layer with 50% dropout rate for additional regularization
    g. A final fully connected layer that maps the 256 features to the desired number of output classes which is 15 in num_classes variable.

    This was to ensure that we only train the last few layers in the head model to fit the pretrained model to our custom dataset to achieve good results.

9. The dataset was then loaded to the GPU.
10. We defined an evaluate function to evaluate the performance of a given model given a validation dataset using a validation data loader, which calculates the average validation loss and accuracy for the validation set.
11. We defined a fit function to train the neural network using the already defined hyperparameters in step 2. The training fit process includes a learning rate scheduling and early stopping to optimize model performance. Learning rate scheduling is important as we will start with an initial learning rate that may be suitable for the early stages of training, but may be too high for the later stages. A high learning rate in the later stages can cause model weights to oscillate and prevent further learning. Having a learning rate schedule in the fit function will allow the optimizer to make smaller, more precise updates to the model's weights as it converges to a more optimal solution, achieving better model performance and faster convergence. The early stopping is defined for training the model on a small dataset, which has a high risk of overfitting, in which the model learns the noise in data instead of the underlying patterns and leads to poor performance on validation data. Early stopping monitors the model's validation loss and stops the training when the performance stops improving for a predefined number of patience which is set at 1 epoch, this prevents the model from overfitting and ensures that the best model with the highest validation accuracy is saved during training. Continuing further, we ensure that the history is saved during the training process so that we may visualize the plots of validation loss and validation accuracy for the purpose of this report.
12. The final train function is complete by taking the directory containing training data, model directory where the trained model will be saved. We will always default to training the model when the train function is called, based on the project specified requirements. When the model is finished training, it will be loaded from the model directory and tested on the validation dataset and returns the history list, which contains the training history.

After running the training function, the best model was found to have a validation accuracy of 93.9%, with early stopping implemented at the 8th epoch, with no learning rate reductions required. The best model was obtained from the 6th epoch, as depicted in the validation accuracy plot below:

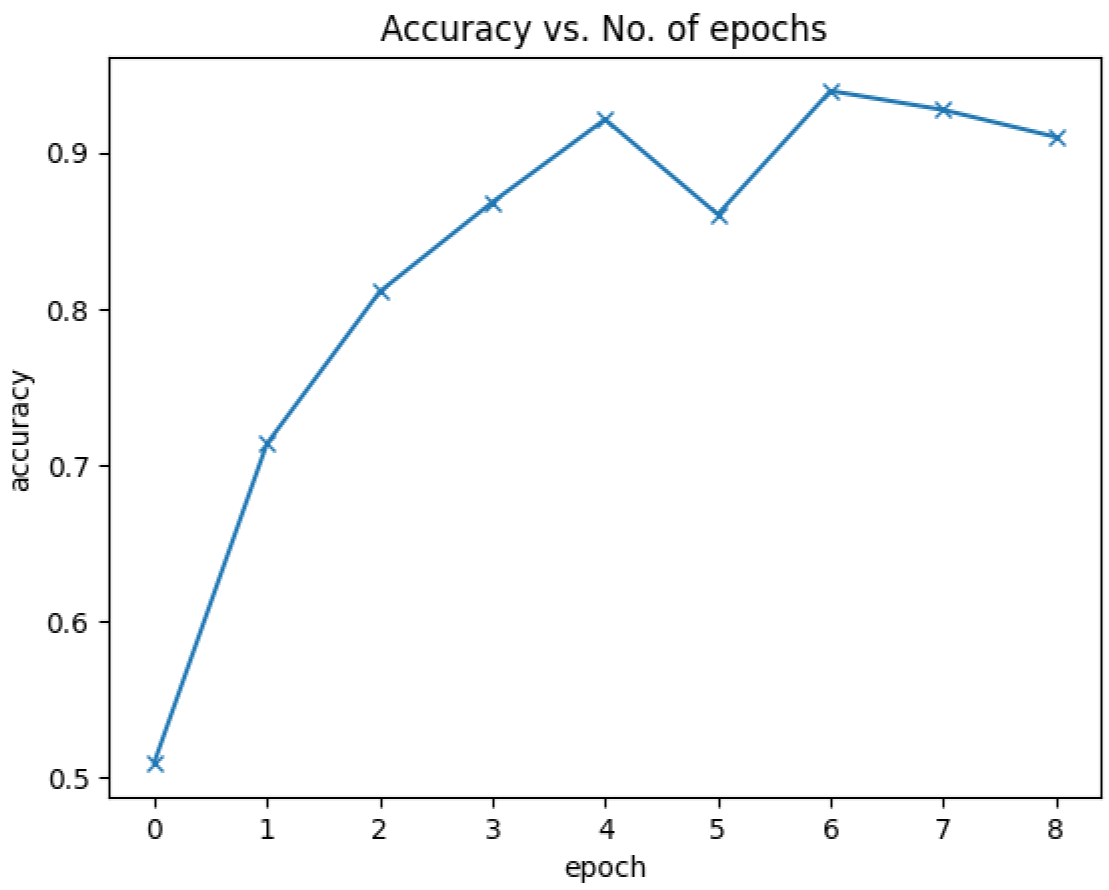


Figure 1: Validation Accuracy vs No. of training epochs

A plot of the training versus the validation loss can be monitored, and helps us to identify the extent of overfitting or underfitting during the training process. This can help to ensure a well-generalized model. From the plot, we can observe that the validation loss is closely matching to the training loss, and the early stopping at the 8th epoch prevented any further overfitting from occurring, showing the value of early stopping in the implementation.

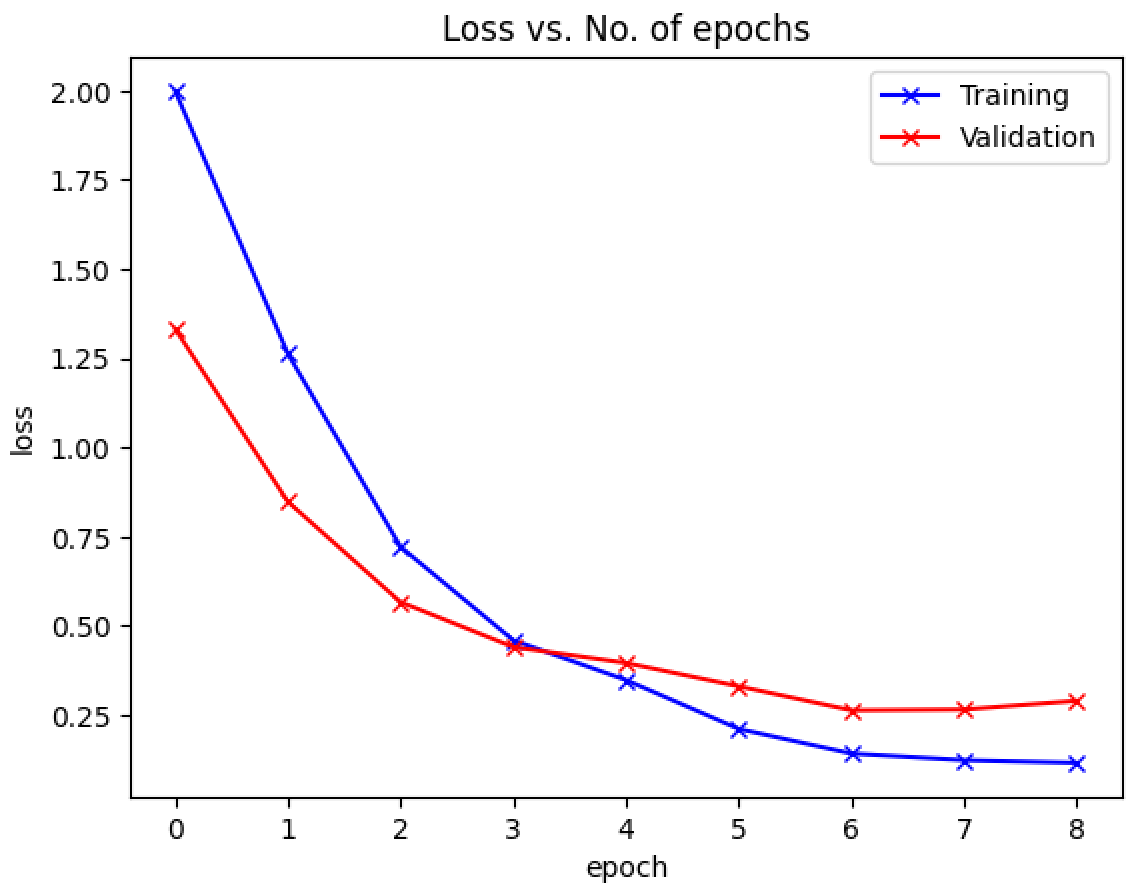


Figure 2: Comparison of training loss versus validation loss

From this process, the training of the pretrained ResNet50 with custom head layers defined is considered to be complete. We now define the function required for the test function. The test() function is required to take in the test data directory and the trained_cnn.pth model in the current directory and output the test accuracy. The following steps are taken to define the test function:

1. The test function will load the test dataset, by iterating over the image files in the test_data_dir and constructing lists containing image file paths, their corresponding labels and encoded labels. These lists are used to create the data frame test_df
2. Define the test image transformations using the already defined test_trans so that we are similarly able to resize and normalize the images to match the format of ImageNet inputs that was used to train ResNet50, ensuring that the new images have similar value distributions and helps the trained CNN model to better generalize on the new dataset.
3. The test dataset and DataLoader is created using test_df, and a test_dl DataLoader is created, which loads the images in batches during evaluation. The DataLoader is converted to a DeviceDataLoader, which ensures the data is transferred to the appropriate device (CPU or GPU) during the evaluation.
4. Load the pretrained CNN model trained_cnn.pth file that is existing in the model folder (RESNET_WEIGHTS_PATH). If a GPU is available, the model is loaded there, else it will be loaded in the CPU. Model is then transferred to the appropriate device using the to_device function.

5. The test dataset is then evaluated, which takes the pretrained model and the test_dl DataLoader as input, the function computes the test accuracy of the model on the blind test dataset.
6. Print and return the test accuracy as the output of the test function.

From this process, the test function can be considered to be complete and this will conclude the project report.