Landmark Classification & Tagging for Social Media 2.0

Meets Specifications

Congratulations!

Some excellent work you have in here. The project is now complete and meets all the requirement.
I liked the way how the project is structured and is wonderful read for anyone attempting to a similar work.

I hope you had a great learning time doing this project and will keep the same spirit going forward when you tackle a real world problem

All the best and keep learning

A good read on between Dropout2d and nn.Drouput, Dropout2d drops entire channels instead of random elements. nn.Drouput defaults to the 1d behavior and it is not effective for convolutional layers. Look at this discussion for details:.

File Requirements

✓ **The submission includes at least the following files:**
cnn_from_scratch.ipynb
transfer_learning.ipynb
app.ipynb
src/train.py
src/model.py
src/helpers.py
src/predictor.py
src/transfer.py
src/optimization.py
src/data.py

Create a CNN to Classify Landmarks from Scratch (cnn_from_scratch.ipynb notebook)

✓

- **In the file 'src/data.py', all the YOUR CODE HERE sections have been replaced with code**
- **The data_transforms dictionary contains train, valid and test keys. The values are instances of transforms.Compose. At the minimum, the 3 set of transforms contains a Resize(256) step, a crop step (RandomCrop for train and CenterCrop for valid and test), a ToTensor step and finally a Normalize step (which uses the mean and std of the dataset). The train transforms should also contain, in-between the crop and the ToTensor, one or more data augmentation transforms.**
- **The ImageFolder instances for train, valid and test use the appropriate transform from the data_transforms dictionary (using the "transform" keyword of ImageFolder)**
- **The data loaders for train, valid and test use the right ImageFolder instance and use the batch_size, sampler, and num_workers that are given in input to the function**
- **In the notebook, the tests for this function were run and they are all PASSED**

Good jon

- Data.py file is correctly updated with code, as a suggestion, you can remove the YOUR CODE HERE when you update with the code
- You have used RandomCrop instead of CenterCrop in the training transforms. This takes a random crop of the image, which provides more variability during training time and results in a more robust model.
- In the test dataloader, you have added shuffle=False to prevent shuffling. This is a best practice to avoid variability when running the test, especially across models.
  Read more on Image Augmentation

✓

- **Answer describes each step of the image preprocessing and augmentation. Augmentation (cropping, rotating, etc.) is not a requirement, but highly recommended**

✓

- **The code gets an iterator from the train data loader and then uses it to obtain a batch of images and labels**
- **The code gets the class names from the train data loader**
- **In the notebook, the 'get_data_loaders' function is used to get the 'data_loaders' dictionary**
- **Then the function 'visualize_one_batch' is called and 5 images from the train data loader are shown with their labels**

- Within 'src/model.py', all the YOUR CODE HERE sections have been replaced with code
- Both the init and the forward method of the class MyModel have been filled
- The class MyModel implements a CNN architecture
- The output layer of the CNN architecture has num_classes outputs (i.e., the number of outputs should not be hardcoded, but should instead use the num_classes parameter passed to the constructor)
- If the CNN architecture uses DropOut, then the amount of dropout should be controlled by the "dropout" parameter of the init method
- The .forward method should *NOT* include the application of Softmax
- In the notebook, the tests for this function were run and they are all PASSED

Answer describes the reasoning behind the selection of architecture type, layer types and so on. The students should reuse some of the concepts learned during the class.

- In the file 'src/optimization.py', all the YOUR CODE HERE sections have been replaced with code
- The get_loss function returns the appropriate loss for a multiclass classification (CrossEntropy loss)
- The relative test for the 'get_loss function' is run in the notebook and is PASSED
- In the 'get_optimizer' function, both the SGD and the Adam optimizer are initialized with the provided input model, as well as with the learning_rate, momentum (for SGD), and weight_decay provided in input
- The relative test for the 'get_optimizer' function is run in the notebook and is PASSED

- In 'src/train.py', all the YOUR CODE HERE sections have been replaced with code
- In the function 'train_one_epoch', the model is set to training mode. Then a proper training loop is completed: the gradient is cleared, a forward pass is completed, the value for the loss is computed, and a backward pass is completed. Finally, the parameters are updated by completing an optimizer step.
- The tests relative to the function 'train_one_epoch' are run (from the notebook) and are all PASSED
- In the function 'valid_one_epoch', the model is set to evaluation mode (so that no gradients are computed), then within the loop a forward pass is completed, and the validation loss is calculated. There should be no backward pass here (this is different than the training loop).
- The tests relative to the function 'valid_one_epoch' are run (from the notebook) and are all PASSED
- In the 'optimize' function, the learning rate scheduler that reduces the learning rate on plateau is initialized. Then within the loop, the weights are saved if the validation loss decreases by more than 1% from the previous minimum validation loss. Then the learning rate scheduler is triggered by making a step.
- The tests relative to the function 'optimize' are run (from the notebook) and are all PASSED
- In the function 'one_epoch_test', the model is set to evaluation mode, a forward pass is completed within the loop, and the loss value is computed. Finally, the prediction is computed by taking the argmax of the logits.
- The tests relative to the function 'one_epoch_test' are run (from the notebook) and are all PASSED.

- Sensible hyperparameters are used (the default or not)
- The solution gets the dataloaders, the model, the optimizer and the loss using the functions completed in the previous steps
- The model is trained successfully (the train and validation loss decrease with the epochs)

- The student runs the testing code in the notebook and obtains a test accuracy of at least 50%

The test accuracy is now above 50 percent and with more tuning it will be higher

- In 'src/predictor.py', the .forward method is completed so that it applies the transforms defined in init (using self.transforms), uses the model to get the logits, applies the 'softmax' function across dim=1, and returns the result
- In the notebook, the tests are run and they are all PASSED
- In the notebook, all the YOUR CODE HERE sections have been replaced with code. In particular, the best weights from the training run are loaded, and torch.jit.script is used to generate the TorchScript serialization from the model
- In the next cell, the saved checkpoints/original_exported.pt file is loaded back using torch.jit.load, then all the remaining cells are run to get the confusion matrix of the exported model
- The diagonal of the confusion matrix should have a lighter color, signifying that most test examples are correctly classified

Use Transfer Learning (transfer_learning.ipynb notebook)

- In 'src/transfer.py', all the YOUR CODE HERE sections have been replaced with code
- All parameters of the loaded architecture are frozen, and a linear layer at the end has been added using the appropriate input features (as returned by the backbone), and the appropriate output features, as specified by the n_classes parameter
- The tests in Step 1 in the notebook are run and they are all PASSED

- The hyperparameters in the notebook are reasonable
- The function 'get_model_transfer_learning' is used to get a model
- The model is trained successfully (the train and validation loss decrease with the epoch)

- The submission provides an appropriate explanation why the chosen architecture is suitable for this classification task.

- Test accuracy is at least 60%

- Appropriate cells have been run to save the transfer learning model into “checkpoints/transfer_exported.pt”

Write Your App (app.ipynb notebook)

- All the YOUR CODE HERE sections in the notebook have been replaced with code
- One of the two TorchScript exports (either the cnn from scratch or the transfer learning model) are loaded using torch.jist.load
- The app is run. In the saved notebook, an image is shown that is not part of the training nor the test set, along with the predictions of the model.

Good work

Go ahead and deploy to a cloud based arch to make it public for all the people

DOWNLOAD PROJECT

RETURN TO PATH