PROJECT SPECIFICATION

Create Your Own Image Classifier

Files Submitted

CRITERIA	MEETS SPECIFICATIONS	COMMENTS
Submission Files	The submission includes all required files. (Model checkpoints not required.)	The required files are submitted as part of the repo (root directory). These include • Image_Classifier_Project.ipynb • train.py • predict.py

Part 1 - Development Notebook

CRITERIA	MEETS SPECIFICATIONS	COMMENTS
Package Imports	All the necessary packages and modules are imported in the first cell of the notebook	The imports are done in cell <1>
Training data augmentation	torchvision transforms are used to augment the training data with random scaling, rotations, mirroring, and/or cropping	The training and test data is augmented using pytorch transform functions. From training data the transforms used are: RandomRotation(), RandomResizedCrop() and RandomHorizantalFlip(). For test and validation, the transforms used are: Resize() CenterCrop(). I tried additional techniques but the performance suffered. I tried the same settings for test and validation as training using flips and random crop! Again the network was not yielding the accuracy of more than 80% and test accuracy is always below 80%. Even if I tested with 50 epochs, the network

CRITERIA	MEETS SPECIFICATIONS	COMMENTS
		was struggling. This can be observed in cell <3>
Data normalization	The training, validation, and testing data is appropriately cropped and normalized	The normalization is done using torchvision Normalize() method for all three (training, test and validation) data sets This can be observed in cell <3>
Data batching	The data for each set is loaded with torchvision's DataLoader	The data is loaded by creating a DataLoader object (trainloader, testloader, validloader) for each of the data sets (train, test and validation) respectively. This can be observed in cell <3>
Data loading	The data for each set (train, validation, test) is loaded with torchvision's ImageFolder	The data for each set is loaded with torchvision’s ImageFolder method. This can be observed in cell <3>
Pretrained Network	A pretrained network such as VGG16 is loaded from torchvision.models and the parameters are frozen	The model is initialized using vgg16 network. This can be seen in cell<> The parameters are frozen using params.requires_grad = False

CRITERIA	MEETS SPECIFICATIONS	COMMENTS
		This can be observed in cell <11>
Feedforward Classifier	A new feedforward network is defined for use as a classifier using the features as input	A new classifier is created using classifier = nn.Sequential(nn.Linear(25088, 4096), nn.ReLU(), nn.Dropout(p=0.2), nn.Linear(4096, 102), nn.LogSoftmax(dim=1)) and the model is updated with model.classifier = classifier. This can be observed in cell <5> as part of the build_model function
Training the network	The parameters of the feedforward classifier are appropriately trained, while the parameters of the feature network are left	The first few times the model is trained using the code in cells <> Later on, the code is converted into functions and the retrained for better performace. The final network parameters I have chosen given me close to 89%

CRITERIA	MEETS SPECIFICATIONS	COMMENTS
	static	validation accuracy and the test accuracy is around 85% the poor performing results can be observed in cells <10(vgg16), 21, 26, 10(vgg19)> More over in the notebook, Image_Classification_Project-expriments.ipynb, the other training I have done is preserved. But I cleaned up the main notebook for submission.
Testing Accuracy	The network's accuracy is measured on the test data	The final test accuracy is <cell 10(vgg16)>
Validation Loss and Accuracy	During training, the validation loss and accuracy are displayed	During training I displayed the following info. Every time I run something, i’m a fan of estimating how much each step/epoch takes. So the information I captured are: Epochs Steps [time taken for steps] Training Loss Validation Loss Validation accuracy

CRITERIA	MEETS SPECIFICATIONS	COMMENTS
		The format is as below `Epoch: 1/5 | Steps:  30[ 14.202s] | Training Loss: 7.513 | Validation Loss: 3.203 | Validation Accuracy: 0.338`
Loading checkpoints	There is a function that successfully loads a checkpoint and rebuilds the model	A function (load_model())is defined to load the checkpoint. This can be seen in cell <41>
Saving the model	The trained model is saved as a checkpoint along with associated hyperparameters and the class_to_idx dictionary	A function (savemodel()) is defined to save the model parameters. This can be observed in cell 32
Image Processing	The process_image function successfully converts a PIL image into an object that can be used as input to a trained model	process_image() function is defined to address this rubric in cell<47>
Class Prediction	The predict function	Predict() function is defined to address this rubric in cell <49>

CRITERIA	MEETS SPECIFICATIONS	COMMENTS
	successfully takes the path to an image and a checkpoint, then returns the top K most probably classes for that image	
Sanity Checking with matplotlib	A matplotlib figure is created displaying an image and its associated top 5 most probable classes with actual flower names	Sanity check is done in cell <67> a function sanity_check() is subsequently defined to do the same task.

Part 2 - Command Line Application

CRITERIA	MEETS SPECIFICATIONS	COMMENTS
Training a network	train.py successfully trains a new network on a dataset of images and saves the model to a checkpoint	Please take a look at the output here. (dsnd-pytorch) gvenkat@gvAwAuR7:~/DSND/DSND_Term1/projects/p2_image_classifier$ python train.py --arch 'vgg16' --n_inputs 25088 --n_hidden 4096 --n_classes 102 --epochs 1 --learning_rate 0.001 --batch_size 32 --drop_prob 0.5 --gpu --print_every 30 --checkpoint vgg16_checkpoint_l1_001_bs_32_dp_05_e1.pth Using cuda:0 THCudaCheck FAIL file=/opt/conda/conda-bld/pytorch_1544176307774/work/aten/src/THC/THCGeneral.cpp line=405 error=11 : invalid argument Epoch: 1/1 | Steps: 30[14.464s] | Training Loss: 6.324 | Validation Loss: 2.946 | Validation Accuracy: 0.390 Epoch: 1/1 | Steps: 60[29.152s] | Training Loss: 3.083 | Validation Loss: 2.122 | Validation Accuracy: 0.497 Epoch: 1/1 | Steps: 90[43.844s] | Training Loss: 2.749 | Validation Loss: 1.579 | Validation Accuracy: 0.608 Epoch: 1/1 | Steps: 120[58.368s] | Training Loss: 2.375 | Validation Loss:

CRITERIA	MEETS SPECIFICATIONS	COMMENTS
		1.429 | Validation Accuracy: 0.640 Epoch: 1/1 | Steps: 150[72.662s] | Training Loss: 2.212 | Validation Loss: 1.340 | Validation Accuracy: 0.634 Epoch: 1/1 | Steps: 180[86.997s] | Training Loss: 2.106 | Validation Loss: 1.121 | Validation Accuracy: 0.699 Accuracy achieved by the network on test images is: 73% gvenkat@gvAwAuR7:~/DSND/DSND_Term1/projects/p2_image_classifier$ ls -l vgg16_checkpoint_l1_001_bs_32_dp_05_e1.pth -rw-r--r-- 1 gvenkat gvenkat 1297067396 Jan 31 05:44 vgg16_checkpoint_l1_001_bs_32_dp_05_e1.pth (dsnd-pytorch) gvenkat@gvAwAuR7:~/DSND/DSND_Term1/projects/p2_image_classifier$
Training validation log	The training loss, validation loss, and validation accuracy are printed out as a network trains	From the previous run we clearly see that the training loss, validation loss and validation accuracy are all reported for each run. Epoch: 1/1 | Steps: 30[14.464s] | Training Loss: 6.324 | Validation Loss: 2.946 | Validation Accuracy: 0.390 Epoch: 1/1 | Steps: 60[29.152s] | Training Loss: 3.083 | Validation Loss:

CRITERIA	MEETS SPECIFICATIONS	COMMENTS
		2.122 | Validation Accuracy: 0.497 Epoch: 1/1 | Steps: 90[43.844s] | Training Loss: 2.749 | Validation Loss: 1.579 | Validation Accuracy: 0.608 Epoch: 1/1 | Steps: 120[58.368s] | Training Loss: 2.375 | Validation Loss: 1.429 | Validation Accuracy: 0.640 Epoch: 1/1 | Steps: 150[72.662s] | Training Loss: 2.212 | Validation Loss: 1.340 | Validation Accuracy: 0.634 Epoch: 1/1 | Steps: 180[86.997s] | Training Loss: 2.106 | Validation Loss: 1.121 | Validation Accuracy: 0.699
Model architecture	The training script allows users to choose from at least two different architectures available from torchvision.models	I have given an option to choose more than one model from torchvision.models. I tested with vgg16 and vgg19. And also with densenet121 (dsnd-pytorch) gvenkat@gvAwAuR7:~/DSND/DSND_Term1/projects/p2_image_classifier$ python train.py --arch 'densenet121' --n_inputs 1024 --n_hidden 120 --n_classes 102 --epochs 1 --learning_rate 0.001 --batch_size 32 --drop_prob 0.5 --gpu --print_every 30 --checkpoint densene121_checkpoint_l1_001_bs_32_dp_05_e1.pth /home/gvenkat/anaconda3/envs/dsnd-pytorch/lib/python3.7/site-packages/

CRITERIA	MEETS SPECIFICATIONS	COMMENTS
		torchvision/models/densenet.py:212: UserWarning: nn.init.kaiming_normal is now deprecated in favor of nn.init.kaiming_normal_. nn.init.kaiming_normal(m.weight.data) Using cuda:0 THCudaCheck FAIL file=/opt/conda/conda-bld/pytorch_1544176307774/work/aten/src/THC/THCGeneral.cpp line=405 error=11 : invalid argument Epoch: 1/1 | Steps: 30[12.756s] | Training Loss: 4.563 | Validation Loss: 4.411 | Validation Accuracy: 0.138 Epoch: 1/1 | Steps: 60[25.512s] | Training Loss: 4.381 | Validation Loss: 4.173 | Validation Accuracy: 0.160 Epoch: 1/1 | Steps: 90[38.551s] | Training Loss: 4.198 | Validation Loss: 3.892 | Validation Accuracy: 0.213 Epoch: 1/1 | Steps: 120[51.506s] | Training Loss: 4.022 | Validation Loss: 3.591 | Validation Accuracy: 0.237 Epoch: 1/1 | Steps: 150[64.518s] | Training Loss: 3.778 | Validation Loss: 3.348 | Validation Accuracy: 0.261 Epoch: 1/1 | Steps: 180[78.390s] | Training Loss: 3.559 | Validation Loss:

CRITERIA	MEETS SPECIFICATIONS	COMMENTS
		3.092 | Validation Accuracy: 0.344 Accuracy achieved by the network on test images is: 35% In addition if the inputs and hidden layers are given correctly other models should also work!
Model hyperparameters	The training script allows users to set hyperparameters for learning rate, number of hidden units, and training epochs	The following is an example run. python train.py --arch 'vgg16' --n_inputs 25088 --n_hidden 4096 --n_classes 102 --epochs 1 --learning_rate 0.001 --batch_size 32 --drop_prob 0.5 --gpu --print_every 30 --checkpoint vgg16_checkpoint_l1_001_bs_32_dp_05_e1.pth Please see the highlighted ones.
Training with GPU	The training script allows users to choose training the model on a GPU	python train.py --arch 'vgg16' --n_inputs 25088 --n_hidden 4096 --n_classes 102 --epochs 1 --learning_rate 0.001 --batch_size 32 --drop_prob 0.5 --gpu --print_every 30 --checkpoint vgg16_checkpoint_l1_001_bs_32_dp_05_e1.pth From the output we can clearly see the script saying it’s using cuda:0 (dsnd-pytorch) gvenkat@gvAwAuR7:~/DSND/DSND_Term1/projects/p2_image_classifier$

CRITERIA	MEETS SPECIFICATIONS	COMMENTS
		python train.py --arch 'vgg16' --n_inputs 25088 --n_hidden 4096 --n_classes 102 --epochs 1 --learning_rate 0.001 --batch_size 32 --drop_prob 0.5 --gpu --print_every 30 --checkpoint vgg16_checkpoint_l1_001_bs_32_dp_05_e1.pth Using cuda:0 THCudaCheck FAIL file=/opt/conda/conda-bld/pytorch_1544176307774/work/aten/src/THC/THCGeneral.cpp line=405 error=11 : invalid argument Epoch: 1/1 | Steps: 30[14.464s] | Training Loss: 6.324 | Validation Loss: 2.946 | Validation Accuracy: 0.390 Epoch: 1/1 | Steps: 60[29.152s] | Training Loss: 3.083 | Validation Loss: 2.122 | Validation Accuracy: 0.497 Epoch: 1/1 | Steps: 90[43.844s] | Training Loss: 2.749 | Validation Loss: 1.579 | Validation Accuracy: 0.608 Epoch: 1/1 | Steps: 120[58.368s] | Training Loss: 2.375 | Validation Loss: 1.429 | Validation Accuracy: 0.640 Epoch: 1/1 | Steps: 150[72.662s] | Training Loss: 2.212 | Validation Loss: 1.340 | Validation Accuracy: 0.634 Epoch: 1/1 | Steps: 180[86.997s] | Training Loss: 2.106 | Validation Loss: 1.121 | Validation Accuracy: 0.699

CRITERIA	MEETS SPECIFICATIONS	COMMENTS
		Accuracy achieved by the network on test images is: 73% (dsnd-pytorch) gvenkat@gvAwAuR7:~/DSND/DSND_Term1/projects/p2_image_classifier$
Predicting classes	The predict.py script successfully reads in an image and a checkpoint then prints the most likely image class and it's associated probability	The following is the predict run. In this particular case I used top_k = 1 (dsnd-pytorch) gvenkat@gvAwAuR7:~/DSND/DSND_Term1/projects/p2_image_classifier$ python predict.py --checkpoint vgg16_checkpoint_l1_001_bs_32_dp_05_e10.pth --image "flower_data/test/1/image_06754.jpg" --top_k 1 --category_names "cat_to_name.json" --gpu Using cuda:0 [0.780955970287323] ['pink primrose'] [1]
Top K classes	The predict.py script allows users to print out the top K	Please take a look at the output below. I used topk=5 and we see their

CRITERIA	MEETS SPECIFICATIONS	COMMENTS
	classes along with associated probabilities	classes and associated probabilities (dsnd-pytorch) gvenkat@gvAwAuR7:~/DSND/DSND_Term1/projects/p2_image_classifier$ python predict.py --checkpoint vgg16_checkpoint_l1_001_bs_32_dp_05_e10.pth --image "flower_data/test/1/image_06754.jpg" --top_k 5 --category_names "cat_to_name.json" --gpu [0.780955970287323, 0.11035141348838806, 0.03172770142555237, 0.012553058564662933, 0.01197933778166771] ['pink primrose', 'mexican petunia', 'petunia', 'carnation', 'mallow'] [1, 98, 51, 31, 97]
Displaying class names	The predict.py script allows users to load a JSON file that maps the class values to other category names	This is implemented in predict.py in lines 172-173 with open(args.category_names, 'r') as f: cat_to_name = json.load(f)
Predicting with GPU	The predict.py script allows users to use the GPU to calculate the predictions	I have given option to use gpu. From the following run, we can clearly see the same. (dsnd-pytorch)

CRITERIA	MEETS SPECIFICATIONS	COMMENTS
		gvenkat@gvAwAuR7:~/DSND/DSND_Term1/projects/p2_image_classifier$ python predict.py --checkpoint vgg16_checkpoint_l1_001_bs_32_dp_05_e10.pth --image "flower_data/test/1/image_06754.jpg" --top_k 1 --category_names "cat_to_name.json" --gpu Using cuda:0 [0.780955970287323] ['pink primrose'] [1] (dsnd-pytorch) gvenkat@gvAwAuR7:~/DSND/DSND_Term1/projects/p2_image_classifier$