UDACITY

< Return to Classroom

Create Your Own Image Classifier

REVIEW

CODE REVIEW

HISTORY

Meets Specifications

Congratulations! You've breezed through this project.

Hi,

Thank you for taking the time to complete this project. You did a fantastic job with this!

Your submission is precise and succinct. The project is organised and presented in a professional manner. Well done!

Now its time to step up to the next level. Good luck! 👍

 `Additional resource for further learning`

Here is a good read about Transfer Learning_1 I hope it adds to your knowledge.

Files Submitted

The submission includes all required files. (Model checkpoints not required.)

 **- THIS SECTION MEETS SPECIFICATIONS**

All files are available. Great job!

Part 1 - Development Notebook

All the necessary packages and modules are imported in the first cell of the notebook

- THIS SECTION MEETS SPECIFICATIONS

Moving all the imports to the top is just a good practice as it helps in looking at the dependencies and the import requirements of the project in one go.

`Additional resources for your knowledge`

Here are a couple of good reads about image augmentation:

it is sometimes of advantage to import certain libraries locally into functions in which they are needed, so that they may not raise compatibility issues with other imported libraries globally (variable scope). This is a great thread that discussed why this is beneficial and has not computational cost involved in the majority of cases.

Kindly make some time to read through these resources, I hope they will further enhance your knowledge.

torchvision transforms are used to augment the training data with random scaling, rotations, mirroring, and/or cropping

- THIS SECTION MEETS SPECIFICATIONS

• You have used the torchvision.transforms well in order to enhance the volume of the data by augmenting it with techniques such as:

- random scaling ✓
- cropping ✓
- mirroring ✓
- rotation. ✓

`Additional resources for your knowledge`

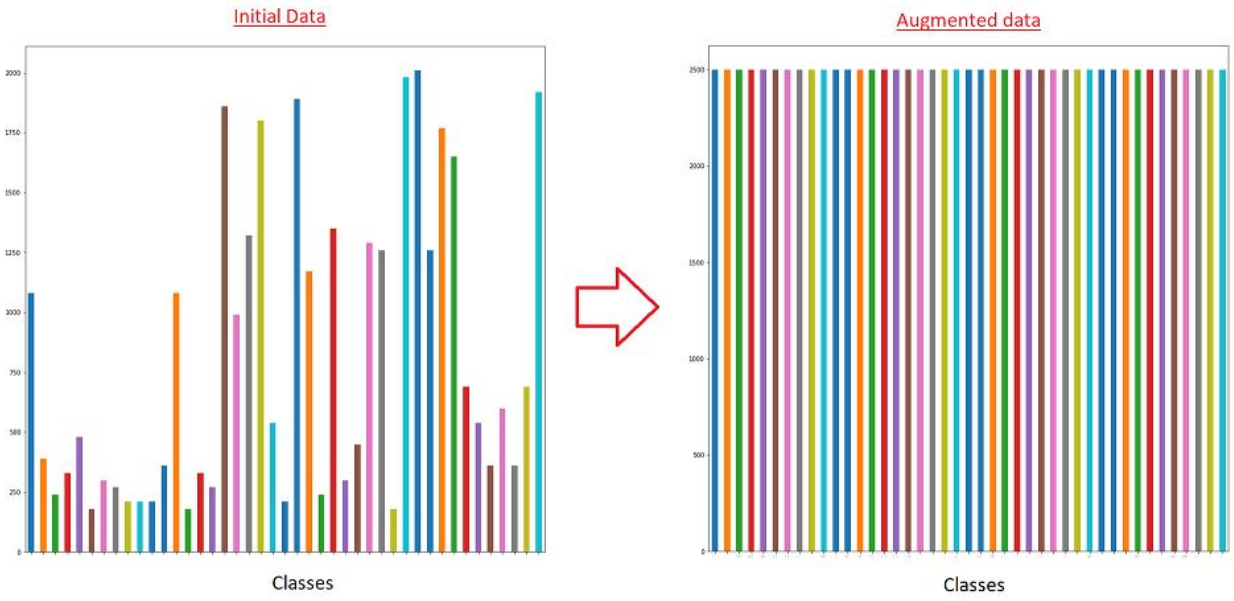
Here are some helpful resources for understanding image augmentations and extracting meaningful insights from them.

- You can learn more about the usefulness of augmenting data in Image Classification tasks (http://cs231n.stanford.edu/reports/2017/pdfs/300.pdf)
- Torchvision native augmentations which can freely be chosen from!

Kindly make some time to read through these resources, I hope they will further enhance your knowledge.

Additional suggestion

Data augmentation artificially increases the size of your training data set and thus makes the model less prone to overfitting. It helps balance an uneven dataset such as the illustration below:



The training, validation, and testing data is appropriately cropped and normalized

- THIS SECTION MEETS SPECIFICATIONS

Nicely done performing the resize to 256 before the crop operation for test and validation datasets. The resizing operation is important because if we did not resize the image and directly crop it, we might lose important regions of the image that could be helpful in classifying the image. Resizing reduces the size of the image while still retaining full information of the image unlike a crop which blindly extracts sections of the image.

Additional resources for your knowledge

Here are a couple of good reads about image cropping and normalizing:

- Imagenet requirements
- Cropping and normalizing the datasets

Do give it a read, I hope they add to your knowledge.

Additional suggestion

You have resized and normalized the data in accordance with the input size accepted by the pretrained models.
To emphasize even more on the importance of normalization I would like to point out that standardizing your data will deliver a well-conditioned input to your model that will be easier for your network to learn from.

The data for each set (train, validation, test) is loaded with torchvision's ImageFolder

- THIS SECTION MEETS SPECIFICATIONS

The code looks good here as well. Well done!

Additional resources for your knowledge

Here are some helpful resources for understanding data visualisations and extracting meaningful insights from them.

- Python torchvision.datasets.ImageFolder Examples
- Detailed explanation of the use of imagefolder in pytorch

Do give it a read, I hope they add to your knowledge.

The data for each set is loaded with torchvision's DataLoader

- THIS SECTION MEETS SPECIFICATIONS

Each set is correctly loaded and you have chosen an appropriate batch size. You can also try trimming the batch size to 32.

Additional resources for your knowledge

Here are some helpful resources for understanding data visualisations and extracting meaningful insights from them.

- Dataloader for sequential and image dataset using Pytorch Deep learning framework
- A detailed example of how to generate your data in parallel with PyTorch

Do give it a read, I hope they add to your knowledge.

Additional suggestion

Validation and testing sets do not require to shuffle the data, since these datasets for evaluating the performance of the network. Training is the only dataset that requires shuffle since the model needs to learn from random batches in order to apply batch gradient descent successfully.

A pretrained network such as VGG16 is loaded from torchvision.models and the parameters are frozen

- THIS SECTION MEETS SPECIFICATIONS

Freezing the layers' weights is extremely important; otherwise, if we train the model without freezing, there will be huge updates on the model weights because the data for training is different from the Imagenet data that these pre-trained models were trained on. This will lead to a model that performs worse instead of performing well. However, once we have trained the final classifier for a while, we can unfreeze the upper layers and train them using a lower learning rate. We do this type of training in cycles.

Additional resources for your knowledge

Here are some helpful resources for understanding data visualisations and extracting meaningful insights from them.

- Transfer learning guide
- Excellent, in this application of transfer learning we opted for completely freezing the parameters, due the relatively small size of our dataset. 🗳️ This way we fully profit of the trained features and maps of the pretrained models weights and biases trained on the ImageNet Dataset!

Please read through these articles when you have some time. I hope you will find it useful.

Additional suggestion

The process to use a pre-trained model is well-established:

1. Load in pre-trained weights from a network trained on a large dataset
2. Freeze all the weights in the lower (convolutional) layers: the layers to freeze are adjusted depending on similarity of new task to original dataset
3. Replace the upper layers of the network with a custom classifier
4. Train only the custom classifier layers for the task thereby optimizing the model for smaller dataset

A new feedforward network is defined for use as a classifier using the features as input

- THIS SECTION MEETS SPECIFICATIONS

Nice work here creating a new classifier using nn.Sequential.

Additional resources for your knowledge

I am attaching a couple of additional reading resources on feedforward network:

- Dropout layer to avoid overfitting
- PyTorch LogSoftmax vs Softmax for CrossEntropyLoss

Please take the time to read them; I hope you learn something new from these resources.

Additional suggestion

Batch normalization is a technique that is typically done by transforming the mini-batch data to zero mean and unit variance before the non-linearity function to try to help improve the gradients of the outputs of the non-linear functions. If you are

curious about it you can find more information in this video where you are given a simple but very illustrative visual example of how batch-normalization works. In Pytorch you can find this layer here

`

The parameters of the feedforward classifier are appropriately trained, while the parameters of the feature network are left static

- THIS SECTION MEETS SPECIFICATIONS

The weights and biases of the classifier are appropriately trained while the features of the pretrained model remain static, nice!

 `Additional resources for your knowledge`

I am attaching a couple of additional reading resources that explain about feedforward classifier training:

- How to optimize in Keras
- Other optimizers

Do give it a read, I hope they add to your knowledge.

 `Additional suggestion`

You did an excellent job Here you can find a quick guide of other optimizers as well as a great plot showing how each one of them behaves..

The network's accuracy is measured on the test data

- THIS SECTION MEETS SPECIFICATIONS

Great job on getting good accuracy on the testing set!

 `Additional resources for your knowledge`

Here are some helpful resources for understanding improving accuracy of your model.

- Improve Models From 80% to Over 90% Accuracy
- 8 proven ways of improving accuracy

Kindly make some time to read through these resources, I hope they will further enhance your knowledge.

 `Additional suggestion`

Great job here in obtaining the required accuracy. In this project, you can reach up to 95% accuracy by tweaking hyperparameters, such as:

- optimizer
- hidden nodes and number of hidden layers
- learning rate
- epochs

During training, the validation loss and accuracy are displayed

- THIS SECTION MEETS SPECIFICATIONS

Both validation loss and accuracy are being captured in the logs

 `Additional resources for your knowledge`

Here are a couple of good reads best practices for logging:

- The training loss (the error or difference between predictions and true values) is the negative log likelihood (NLL). The NLL loss in PyTorch expects log probabilities, so we pass in the raw output from the model's final layer.
- The optimizer is Adam, an efficient variant of gradient descent that generally does not require hand-tuning the learning rate. During training, the optimizer uses the gradients of the loss to try and reduce the error ("optimize") of the model output by adjusting the parameters. Only the parameters we added in the custom classifier will be optimized.

Please read through these articles when you have some time. I hope you will find it useful.

  `Additional suggestion`

As we continue training, the training loss will only decrease, but the validation loss will eventually reach a minimum and plateau or start to increase. We ideally want to stop training when the validation loss is at a minimum in the hope that this model will generalize best to the testing data.Recommend you to check out early stopping where we stop training when the validation loss has not decreased for a number of epochs. When using early stopping, every epoch in which the validation loss decreases, we save the parameters so we can later retrieve those with the best validation performance.

There is a function that successfully loads a checkpoint and rebuilds the model

- THIS SECTION MEETS SPECIFICATIONS

Nicely done writing the `loadModel` method to successfully load the checkpoint and rebuild the model.

  `Additional suggestion`

In this type of transfer learning application, we might not want to save the fully pretrained models parameters (but only the classifiers parameters), as we have completely frozen the models parameters, we can alternative just call the attributes in the load function and therefore keep the checkpoint file small in size. This is particularly helpful for a model architecture like densenet, which has only fully connected layers, resulting in checkpoint files of 1- 2 GB in size.

The trained model is saved as a checkpoint along with associated hyperparameters and the class_to_idx dictionary

- THIS SECTION MEETS SPECIFICATIONS

Great job saving the major hyperparameters in the checkpoint! This practice will make it easy to retrain your model in the future!

Additional resources for your knowledge

Here are some helpful resources for understanding saving the checkpoint.

- Improving model using checkpointing
- Resuming training from checkpoint for Pytorch tutorials.

Please read through these articles when you have some time. I hope you will find it useful.


Additional suggestion

I recommend checking out the below example for a full snippet containing all hyperparameters:

```
checkpoint = {'input_size': (3, 224, 224),
              'output_size': 102,
              'hidden_layers': hidden_layers,
              'batch_size': batch_size,
              'learning_rate': lr,
              'model_name': model_name,
              'state_dict': model.state_dict(),
              'optimizer': optimizer.state_dict(),
              'epoch': epochs,
              'class_to_idx': model.class_to_idx}

torch.save(checkpoint, 'checkpoint.pth')
```

The process_image function successfully converts a PIL image into an object that can be used as input to a trained model

- THIS SECTION MEETS SPECIFICATIONS

Great job, your predict function works perfectly and you have unsqueezed the 1st dimension, since you only loaded a single image, and the model expects a 4-dimensional tensor with the batch size being the first dimension, great job!

Additional suggestion

While most of the logic here are correct, you need to remember that the output of the model prediction is not the exact class of the image, but class labels/indexes. When you load the dataset using the ImageFolder function, it automatically converts the available classes into numeric integers. Example, say you had classes 'dog, car, elephant', then ImageFolder would convert them to '0, 1, 2' respectively and the class_to_idx dictionary would have the mapping of it, something like
{
'cat':0,
'doc':1,
'elephant':2
}
You can get the classes by doing something like this to reverse it, where you can query the label and get the exact class.
`idx_to_class = {val: key for key, val in model.class_to_idx.items()} top_classes = [idx_to_class[each] for each in tn_classes]`.

The predict function successfully takes the path to an image and a checkpoint, then returns the top K most probably classes for that image

- THIS SECTION MEETS SPECIFICATIONS

You just need to make the image have a smaller side sized at 256 and maintain the aspect ratio, which you have taken care. Good job!

Additional resources for your knowledge

Here are some helpful resources for PIL library.

- Some interesting tricks with PIL
- Basic functions with PIL

Kindly make some time to read through these resources, I hope they will further enhance your knowledge.

Additional suggestion

You did an excellent job here, while revisiting the project feel free to use PIL functionalities. The reason why manual transformations are more effective code and the light weight solution in this instance, is that we only want to prepare 1 image at a time for a prediction, when using pytorch´s generic dataloader you essentially built a file structure around this image, which needs more computing, hence the manual transformations are the more effective solution, programming wise.

A matplotlib figure is created displaying an image and its associated top 5 most probable classes with actual flower names

- THIS SECTION MEETS SPECIFICATIONS

Well done, your sanity_check works as required, great job!

 `Additional resources for your knowledge`

I am attaching a couple of additional reading resources that explain about plotting your findings:

- Sanity Check: Overfitting on a small dataset
- Testing Your PyTorch Models with Torcheck

Please read through these articles when you have some time. I hope you will find it useful.

  `Additional suggestion`

[Why a sanity_check?] The prediction may not be correct, meaning at instances the picture you have chosen does not correlate to the prediction.
There are numerous explanations for this, with errors in class-to-index handling amongst others, however the most important problematic, engineers are facing in regards to this - which is also why sanity-checks were introduced - is called data leakage
Frequently in data science competitions online, there are teams yielding 'too good to be true' accuracies, by having data leakage issues, as e.g. the model having access to the test data already prior to training, then those models are not production ready, and need to be reworked, this is what the sanity check is for.

Part 2 - Command Line Application

train.py successfully trains a new network on a dataset of images and saves the model to a checkpoint

- THIS SECTION MEETS SPECIFICATIONS

Great job! train.py successfully trains a new network on a dataset of images and saves the model to a checkpoint

The training loss, validation loss, and validation accuracy are printed out as a network trains

- THIS SECTION MEETS SPECIFICATIONS

Awesome! Training loss, validation loss etc. are being printed correctly during the training.

 `Additional resources for your knowledge`

Here are a couple of good reads about logging and debugging:

- 9 Steps of Debugging Deep Learning Model Training
- Stop Using “Print” and Start Using “Logging”

Please read through these articles when you have some time. I hope you will find it useful.

You did an excellent job logging these insightful observations. I encourage you to go through this video which has some fascinating insights in the ML based analytics.

The training script allows users to choose from at least two different architectures available from torchvision.models

Great to see that you are letting the user choose from multiple different architectures available from torchvision.models, appreciated!

The training script allows users to set hyperparameters for learning rate, number of hidden units, and training epochs

All the required hyperparameters are taken in as argument. Good job!

You did a fantastic job allowing users to set the hyperparameters. I also recommend setting default values for the same as well as inforomative tip via help such as minimum and maximum acceptable values.

The training script allows users to choose training the model on a GPU

Great implementation and good work using cuda() or gpu at the places required!

The predict.py script successfully reads in an image and a checkpoint then prints the most likely image class and it's associated probability

 - THIS SECTION MEETS SPECIFICATIONS

The predict.py perfectly reads an image and a checkpoint to print out the most probable image class. Well done!

The predict.py script allows users to print out the top K classes along with associated probabilities

 - THIS SECTION MEETS SPECIFICATIONS

The top K classes are printed alongwith their probabilities. Great job!

The predict.py script allows users to load a JSON file that maps the class values to other category names

 - THIS SECTION MEETS SPECIFICATIONS

Yes the script has the capability to load from a JSON FIle. Great job!

The predict.py script allows users to use the GPU to calculate the predictions

 - THIS SECTION MEETS SPECIFICATIONS

Great implementation here as well and good work using cuda at the places required. Great job!

DOWNLOAD PROJECT

RETURN TO PATH