UDACITY

‹ Return to Classroom

Landmark Classification & Tagging for Social Media 2.0

REVIEW

CODE REVIEW 16

HISTORY

Meets Specifications

Great submission! 👏 👏

Now you can think about taking this to the next step.
In addition to the suggestions given in the review comments here are some more:

1. Fine-tune the hyperparameters for the given dataset and the model for better accuracy and performance. Start with standard values given in research papers and use grid search for fine-tuning. https://www.kaggle.com/willkoehrsen/intro-to-model-tuning-grid-and-random-search.
2. Experiment with transfer learning: using different models, fine-tuning different layers and how the hyper-parameters are adjusted accordingly.
3. You can try cutout augmentation. https://www.henryailabs.com/3-2-19.html
4. Explore what each layer is doing, http://yosinski.com/deepvis
   Try GRADCAM to identify what the network is seeing in the input and how it is classifying.
5. Think about how to increase accuracy? How to regularize the network? Think about how to make the training faster and improve model performance?
6. Try the model on a different dataset.
7. Read more related research papers and implement variations and solutions.
8. Create an app.

All the best!! 👍 👍

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

All required files are submitted. Thank you.

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

✅ Three separate data loaders for the training, validation, and test datasets are created. The PyTorch's DataLoader class is used correctly. Learn more about Dataloaders here

⚠️ Test Accuracy: 49% (616/1250) It should be at least 50%
You may have to revisit your augmentations.
Correct Augmentation can improve the model performance and a bad one can break it and even hinder the performance. The augmentation increases the data and take more time and space resources, whenever we use it should be worth it.
So how much of augmentation is enough?
Which augmentations do you think make sense for the give landmark dataset?

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

My code first resizes the image to 256 and then crops to 224. I picked 224 as the input size because it is the recommended input size for using pytorch's pre-trained models. I did decide to augment the dataset via RandAugment, a typical set of augmentations for natural images. I added this augmentation with the goal of improving my model's robustness, thus improving test accuracy.

Please revisit the augmentations while you try to get an accuracy of at least 50%.
As mentioned before:
Correct Augmentation can improve the model performance and a bad one can break it and even hinder the performance. The augmentation increases the data and take more time and space resources, whenever we use it should be worth it.
So how much of augmentation is enough?
Which augmentations do you think make sense for the give landmark dataset?

- **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**

Iterator is used to get and visualize data correctly.
You have printed images with labels.
Well done 👍 .

- **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**

- 4 layers of convolutional layers gives good enough depth for the model to learn details of the image. Well done.
- Good use of pooling layers. It extracts important features and save a lot of time and resources.
- 2 fully connected layers give a good depth to the model. It helps to bring all the features together and make a big feature that gives a meaningful output.
- Last layer has the num_classes as the size of output..
- Good use of batch normalization for faster convergence.

- Good use of dropout layers. It regularizes the network.

Suggestions:

I would have liked to know the reason behind the choice of Leaky Relu over Relu.

The model is deep(many layers) and wide(many parameters) enough. Thats good. 👍 👍
Read more about convolutional arithmetic here. https://arxiv.org/abs/1603.07285

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.

I decided to use 5 convolutional layers so that my model could be sufficiently expressive. I used dropout layers to reduce my model's tendency to overfit the training data. I made my model output a 50-dimensional vector to match with the 50 available landmark classes.

The details of the layers and the chosen parameters are articulated clearly.
I would have liked to know the reason behind the choice of Leaky Relu over Relu.

- **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**

CrossEntropyLoss is the right loss function. 👍
https://discuss.pytorch.org/t/difference-between-cross-entropy-loss-or-log-likelihood-loss/38816/7

Optimizers used correctly with parameters.
Check out the comparative study on optimizers. link.

- **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.**

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. Well done!
Validation function implemented correctly.
Test function implemented correctly.
Test case passed!

ReduceLROnPlateau is a good LR optimization choice.

- **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)**

batch_size = 64
valid_size = 0.2
num_epochs = 100
num_classes = 50
dropout = 0.4
learning_rate = 0.003
opt = 'sgd'
weight_decay = 0.05

Hyper-parameters are balanced. Training looks fine. Well done. 👍👍
You can still tune them to increase the accuracy > 50%

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

Test Accuracy: 49% (616/1250)

Accuracy should be > 50%. Please follow the suggestions mentioned in the review comments and work on it.

- 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

Prediction steps implemented correctly.
Confusion matrix implemented correctly.
https://www.geeksforgeeks.org/confusion-matrix-machine-learning/

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 transfer learning is correctly implemented using pre-trained model, its layers are correctly frozen. The classifier is correctly replaced with new trainable FC layer with num_classes as output size. Well done 👍👍
https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html

- 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)

–

batch*size = 64 # size of the minibatch for stochastic gradient descent (or Adam)*
valid_size = 0.2 # fraction of the training data to reserve for validation
num_epochs = 50 # number of epochs for training
num_classes = 50 # number of classes. Do not change this
learning_rate = 0.001 # Learning rate for SGD (or Adam)
opt = 'adam' # optimizer. 'sgd' or 'adam'
weight_decay = 0.0 # regularization. Increase this to combat overfitting

Well balanced hyper-parameters. You can decrease number of epochs and learning rate for pre-trained models.

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

I decided to use ResNet18 for the base of my model, since it performs fairly well on ImageNet and is not too large of a model. Also, since ResNet18 was trained for the ImageNet task, it is a good model to use for this landmark classificaiton task, since both ImageNet and this landmark task use images of natural scenes.

Clear explanation.

Resnet is a good choice. Read about the power of residual learning here

- Test accuracy is at least 60%

Test Accuracy: 79% (988/1250)
meets the expectation. Well done. 👏👏

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

Training looks good. The model is saved.

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.

```
In [2]: wgs = [Label("Please upload a picture of a landmark"), btn_upload, btn_run, out_pl]
        wgs.extend(labels)

        VBox(wgs)
```

```
In [3]: print(labels)

        [Label(value='16.Eiffel_Tower (prob: 1.00)'), Label(value='20.Matterhorn (prob: 0.00)'), Label(value='29.Petronas_Towers (prob: 0.00)'), Label(value='14.Terminal_Tower (prob: 0.00)'), Label(value='
        17.Changdeokgung (prob: 0.00)')]

In [4]: wgs = [Label("Please upload a picture of a landmark"), btn_upload, btn_run, out_pl]
        wgs.extend(labels)

        VBox(wgs)
```

Correct results. 👍👍

DOWNLOAD PROJECT

16 CODE REVIEW COMMENTS ›

RETURN TO PATH

Rate this review

START