UDACITY

Return to Classroom

Landmark Classification & Tagging for Social Media

REVIEW

HISTORY

Meets Specifications

Great job! A really well-rounded submission.

- Do think of adding more features beyond this model like converting the text you get as a prediction to geocode and then geotagging the image which would make for a cool map visualization of user-uploaded images if you ever think of building a social media platform :D
- Also there was a Kaggle challenge as well by Google which was done for Landmark classification in 2020 so do check it out if you want to play with bigger dataset. (~100GB)
  https://www.kaggle.com/c/landmark-recognition-2020

All the best!

Files Submitted

The submission includes the required notebook file and HTML file. When the HTML file is created, all the code cells in the notebook need to have been run so that reviewers can see the final implementation and output.
All files are present.

Step 1: Create a CNN to Classify Landmarks (from Scratch)

The submission randomly splits the images at `landmark_images/train` into train and validation sets. The submission then creates a data loader for the created train set, a data loader for the created validation set, and a data loader for the images at `landmark_images/test`.

`splitfolders.ratio("/data/landmark_images/train", output="train_valid", seed=1337, ratio=(.8, .2), group_prefix=None)`

Tip - You can also use less images for validation (10%) split if you feel there aren't many images to validate on.

```
train_data = datasets.ImageFolder('./train_valid/train', transform=train_transform)
valid_data = datasets.ImageFolder('./train_valid/val', transform=test_transform)
test_data = datasets.ImageFolder('/data/landmark_images/test', transform=test_transfor
m)
# Save classes names
n_classes = len(train_data.classes)
classes = [class_.split(".")[1].replace("_", " ") for class_ in train_data.classes]
batch_size = 20
# Create data loaders
train_loader = torch.utils.data.DataLoader(train_data, batch_size, shuffle=True)
valid_loader = torch.utils.data.DataLoader(valid_data, batch_size, shuffle=True)
test_loader = torch.utils.data.DataLoader(test_data, batch_size, shuffle=True)
loaders_scratch = {'train': train_loader, 'valid': valid_loader, 'test': test_loader}
```

- Use of tensordataset and dataloader is done perfectly. Do read about more of the parameters offered in those two class object methods as it gives us number of workers and other multiprocessing options if the dataset is big.
- You can also use num_workers = -1 to utilize all CPUs present in a device if you want to utilise maximum possible parallelization.\
  https://pytorch.org/docs/stable/data.html
- subsetrandomsampler will always randomize validation and training set so in a way that is good to use instead of splitfolders as you needed to added shuffling in the loader.

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

```
Answer:

I randomly resized crop the images to 224x224 pixels for training and for testing, I di
d normal resized to 256, then center crop to 224x224 pixels size.then center crop to 22
4x224 pixels size. I have picked 224x224 pixels as the size of the input tensor.
Data augmentation has been applied on just the training subset of the dataset through r
```

```
andom rotations of 10 degrees and random horizontal flips to the images, also with rand
om and different color jitters.
```

- You should ideally use the same transform for test and validation set as that would be best way to replicate performance on both validation and test set but using augmentations on valid and test set is not necessary. (sometimes in some papers you will find that there is a bit of augmentation in test and valid set as well but those are very rare cases)
- Also pytorch's transform resize behaves differently when you pass only integer so keep that in mind\ https://pytorch.org/docs/stable/torchvision/transforms.html#torchvision.transforms.Resize

The submission displays at least 5 images from the train data loader, and labels each image with its class name (e.g., "Golden Gate Bridge").

- You had to display at least 5 images so done really well.
- You can play around with figsize to have a lesser gap in between them
- Glad that you string split to remove the number from label name

The submission chooses appropriate loss and optimization functions for this classification task.

```
## TODO: select loss function
import torch.nn as nn
import torch.optim as optim
criterion_scratch = nn.CrossEntropyLoss()
def get_optimizer_scratch(model):
    ## TODO: select and return an optimizer
    optimizer = optim.SGD(model.parameters(), lr=0.01)
    return optimizer
```

Tip - You can also try Adabound which is a blend of Adam and SGD
https://medium.com/syncedreview/iclr-2019-fast-as-adam-good-as-sgd-new-optimizer-has-both-78e37e8f9a34

The submission specifies a CNN architecture.

```
        self.conv1 = nn.Conv2d(3, 16, 3, padding=1)
        self.conv2 = nn.Conv2d(16, 32, 3, padding=1)
        self.conv3 = nn.Conv2d(32, 64, 3, padding=1)
        self.pool = nn.MaxPool2d(2, 2)
        self.fc1 = nn.Linear(28 * 28 * 64, 256)
        self.fc2 = nn.Linear(256, n_classes) # n_classes = 50
        self.dropout = nn.Dropout(0.3)
        self.leaky_relu = nn.LeakyReLU(negative_slope=0.2)
        self.batch_norm2d = nn.BatchNorm2d(32)
        self.batch_norm1d = nn.BatchNorm1d(256)
```

- 3conv 2 fc is decent.
- For starting out any scratch model, you can start slow and small like a simple 3conv2fc)like you did) as it is a well-balanced network to get the features to avoid memory errors and just a plain run to see if it is giving decent results and then going for the kill by using more robust ones or look at improving data or increasing dataset size.

Answer describes the reasoning behind the selection of layer types.

```
Answer:

CNN contains of 3 layers as follows
1-Conv1 - 224x224 16 | Leaky relu activation function
2-Conv2 - 112x112 32 | Leaky relu activation function
3-Conv3 - 56x56 64 | Leaky relu activation function and 2 fully connected layers and dr
opout of 30 % probability, Dropout of 30% probability,Dropout has been used to avoid ov
erfitting and Max-pooling layers to focus on the main target features via dividing the
 image by a factor of 2. I attempted to speed up the training process and to perform so
me regularization to the model via Batch Normalization.
```

- You can also try and imitate smaller architectures like LeNET like structures for training scratch models as well.
- Including batch norm can also be done in general (as you did) during these tasks but if you check and find that the dataset is imbalanced then only it gives good results. Here you can probably avoid using batch norm as well because it's a balanced dataset.

- You can visualize each layer and filter with what shape they are learning at each layer to make it more intuitive to builds CNNs from scratch.
  https://cs231n.github.io/understanding-cnn/
- You can reduce/remove dropout while training from scratch as the model would learn slowly because of this for FC layers are the ones that tune the weights to give to features obtained from CNN layers to align to classes so having this lesser or no dropout is suggested initially so do keep that in mind. If you see drastic overfitting then try to first check your data and architecture while training from scratch.

The submission implements an algorithm to train a model for a number of epochs and save the "best" result.

```
        ## TODO: if the validation loss has decreased, save the model at the filepath s
tored in save_path
        if valid_loss <= valid_loss_min:
            print('Validation loss decreased ({:.6f} --> {:.6f}).  Saving model ...'.fo
rmat(valid_loss_min,

valid_loss))
            torch.save(model.state_dict(), save_path)
            valid_loss_min = valid_loss
```

- You can also use lr_scheduler which automatically decays your learning rate as you have trained for quite a lot of epochs (40)\
  https://stackoverflow.com/questions/59017023/pytorch-learning-rate-scheduler
- You can always plot the losses against epochs to see if the training set goes into overfitting/underfitting mode or not so do make decisions from it initially by just training it for 5-10 epochs.
- Also, we can implement early stopping as well if you feel some epsilon-delta (1e^-04) change is not happening consistently for n epochs in validation loss. https://machinelearningmastery.com/early-stopping-to-avoid-overtraining-neural-network-models/

The submission implements a custom weight initialization function that modifies all the weights of the model. The submission does not cause the training loss or validation loss to explode to `nan`.

```
def custom_weight_init(m):
    ## TODO: implement a weight initialization strategy
    if isinstance(m, nn.Conv2d):
        n = m.kernel_size[0] * m.kernel_size[1] * m.out_channels
        m.weight.data.normal_(0, np.sqrt(2. / n))
        if m.bias is not None:
            m.bias.data.zero_()
    elif isinstance(m, nn.BatchNorm2d):
        m.weight.data.fill_(1)
```

```
        m.bias.data.zero_()
    elif isinstance(m, nn.Linear):
        n = m.in_features
        y = 1.0/np.sqrt(n)
        m.weight.data.normal_(0, y)
        m.bias.data.zero_()
```

- It has been seen with certain techniques like xavier and kaiming significant improvements are gained in results so you can train for 10 epoch and see which one is working well and if the loss is coming as Nan values then you need to debug your training loop above.
- https://paperswithcode.com/method/he-initialization
- https://towardsdatascience.com/weight-initialization-in-neural-networks-a-journey-from-the-basics-to-kaiming-954fb9b47c79

The trained model attains at least 20% accuracy on the test set.

```
Validation loss decreased (2.616753 --> 2.564936). Saving model ...
```

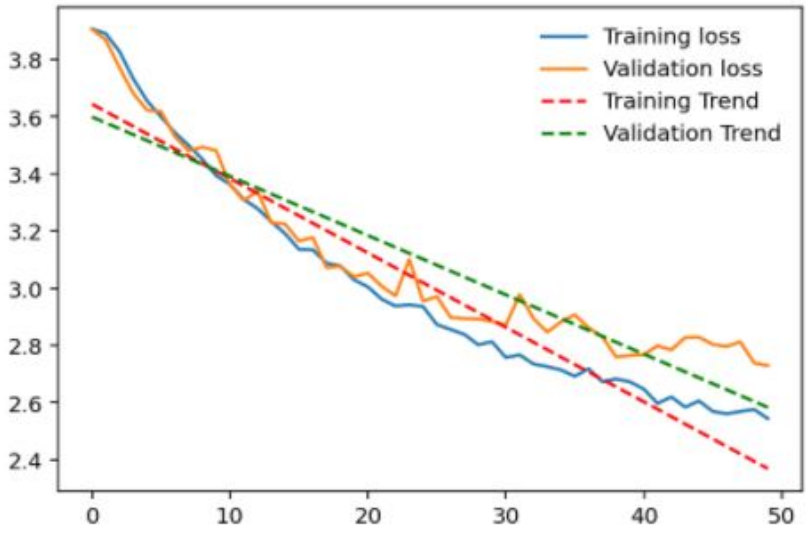
```
Test Loss: 2.435316


Test Accuracy: 39% (497/1250)
```

This is good accuracy!

- If there is overfitting/underfitting it can help to identify the root cause early on. Its fine for testing out a scratch model as in production usually we don't use models trained from scratch unless they need to be super customizable and light weight.
- Here our aim was to do a quick and easy scratch model training iteration and reach a certain level of accuracy to understand how to go about training models.
  https://medium.com/analytics-and-data/overview-of-the-different-approaches-to-putting-machinelearning-ml-models-in-production-c699b34abf86
- You can also make a plot like this to see if its worth training more as if the trend has saturated, you can stop training as well.

```
Out[10]: <matplotlib.legend.Legend at 0x16b81b788b0>
```



Step 2: Create a CNN to Classify Landmarks (using Transfer Learning)

The submission specifies a model architecture that uses part of a pre-trained model.

`loaders_transfer = loaders_scratch.copy()`

Glad that you to used copy() to keep them consistent instead of redefining them again as directly assigning the transfer loader variable to scratch one creates a shallow copy so if there is any change in transfer data loaders, it will also reflect in scratch loaders.

`model_transfer = models.vgg16(pretrained=True)`

Interesting choice of Vgg16.

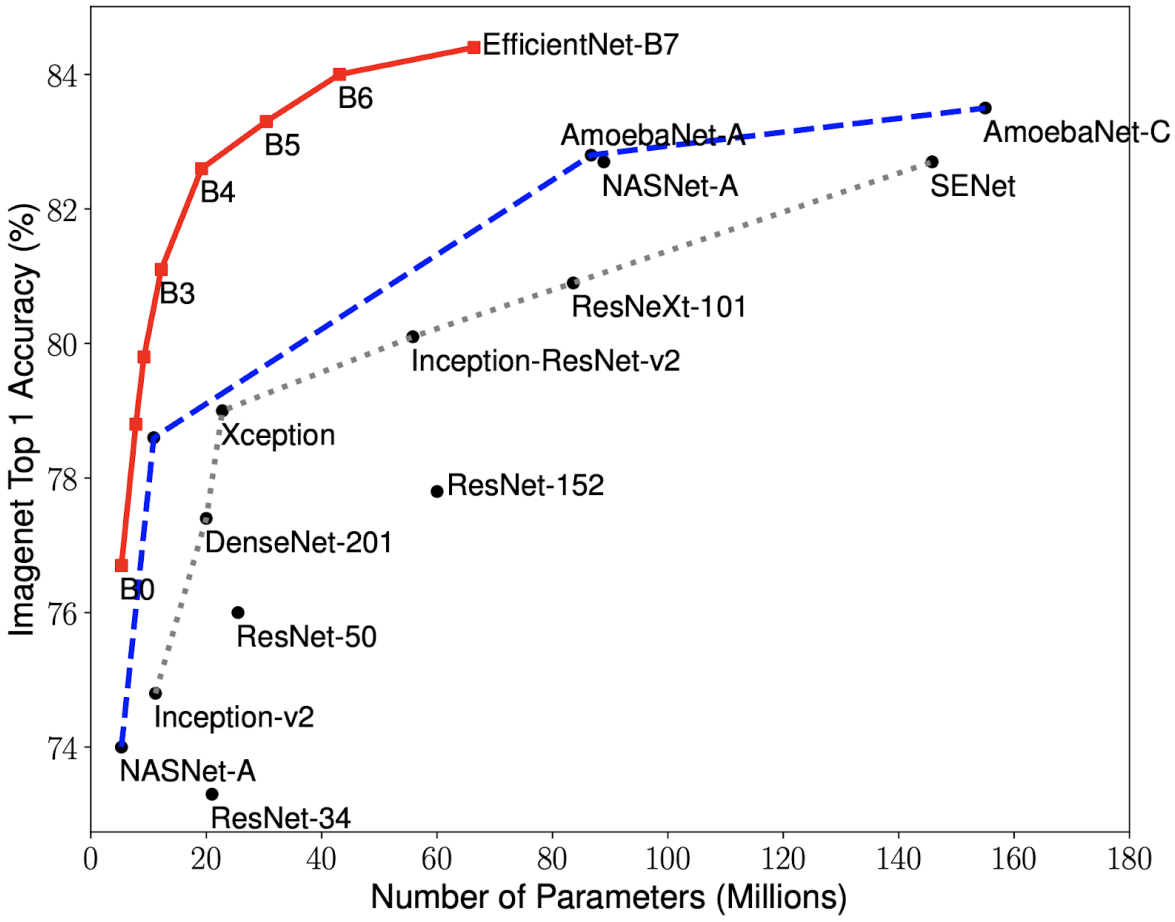
The submission details why the chosen architecture is suitable for this classification task.

```
Answer:

i choosed VGG-16 becase it is better than DenseNet-161 .this arch gives us a good archi
tecture and taking all preceding feature-maps as input.and that gives me higher scores
and faster performace so my accuracy 78% after 15 epoches
```

- You can also look at other networks than VGG and Resnet like Inception , Xception, EfficientNet which work well on imagenet in general
- Also, Resnet is comparatively lesser in terms of parameters to train while accuracy is almost similar so you can try these models if you want keeping in mind the size of these models and inference time that they take

if your applications require faster response, then you have to do some tradeoff in deciding which one to use.



The submission uses model checkpointing to train the model and saves the model weights with the best validation loss.

```
# TODO: train the model and save the best model parameters at filepath 'model_transfer.
pt'
train(15, loaders_transfer, model_transfer, get_optimizer_transfer(model_transfer), cri
terion_transfer,
      use_cuda, 'model_transfer.pt')
#-#-# Do NOT modify the code below this line. #-#-#

# load the model that got the best validation accuracy
model_transfer.load_state_dict(torch.load('model_transfer.pt'))
```

- You can use high learning rate here as ideally we should be training them for fewer epochs (2-10) so its good to get better results as fast as we can.
- This is a tad bit long article but you can use it to refer anytime you have any doubts regarding transfer learning (you can treat it like a book chapter for reference) -> https://towardsdatascience.com/a-comprehensive-hands-on-guide-to-transfer-learning-with-real-world-applications-in-deep-learning-212bf3b2f27a

Accuracy on the test set is 60% or greater.

```
Epoch: 15 Training Loss: 0.911632 Validation Loss: 0.984043
```

```
Test Loss: 0.820919


Test Accuracy: 77% (971/1250)
```

- This is decent accuracy which passes the rubric.
- You can try using denser networks/networks with more parameters like EfficientNet, using SGD as optimizer and also checking the augmentations to the data to see if this can be pushed to 90s.

Step 3: Write Your Landmark Prediction Algorithm

The submission implements functionality to use the transfer learned CNN from Step 2 to predict top k landmarks. The returned predictions are the names of the landmarks (e.g., "Golden Gate Bridge").

```
    transform = transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])])

    img = transform(img)
    img.unsqueeze_(0)

    if use_cuda:
        img = img.cuda()

    model_transfer.eval()
    top_values, top_idx = output.topk(k)
```

- Glad that you used the same transform function which was used for test set. You can even directly call it instead of redefining.
- Good use of topk which gives you descending order k results\
https://pytorch.org/docs/stable/generated/torch.topk.html

The submission displays a given image and uses the functionality in "Write Your Algorithm, Part 1" to predict the top 3 landmarks.

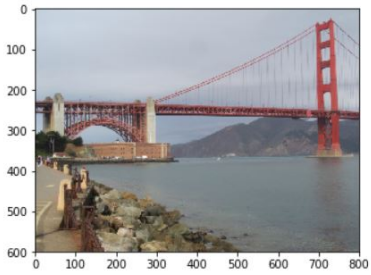
- Done well.
- You can also print the probabilities of the top 3 outcomes as text along with each image for more better understanding how model is predicting.

```
In [17]: def suggest_locations(img_path):
             # get landmark predictions
             predicted_landmarks = predict_landmarks(img_path, 3)

             ## TODO: display image and display landmark predictions
             img = Image.open(img_path).convert('RGB')
             plt.imshow(img)
             plt.show()

             print(f"Actual Label: {img_path.split('/')[2][3:].replace('_',' ').split('.')[0]}")
             print(f"Predicted Label in order: Is this picture of the\n {predicted_landmarks[0]}, {predicted_landmarks[1]}, or {predic
         _landmarks[2]}?")


         # test on a sample image
         suggest_locations('images/test/09.Golden_Gate_Bridge/190f3bae17c32c37.jpg')
```

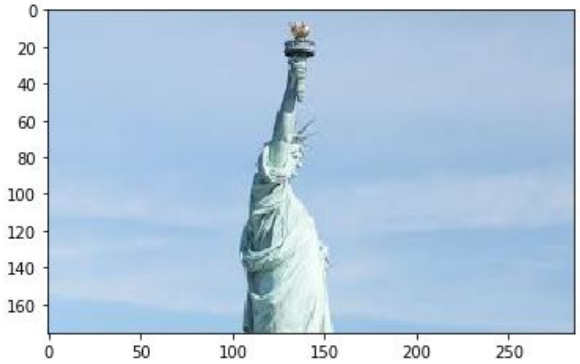



```
Actual Label: Golden Gate Bridge
Predicted Label in order: Is this picture of the
 Golden Gate Bridge, Forth Bridge, or Brooklyn Bridge?
```

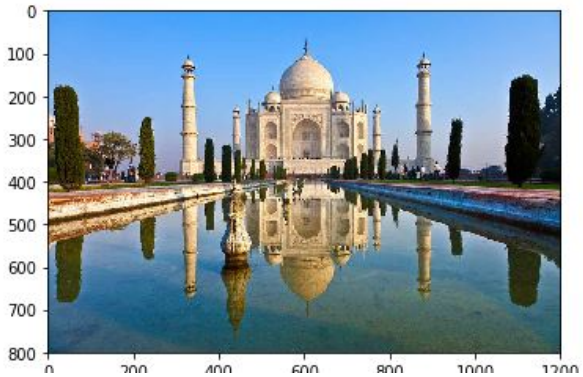
The submission tests at least 4 images.

- Tested well on images.
- Do try to test more monuments given from the test dataset and also other ones outside the dataset to see how close they get to predict. As it only contains 50- classes, there would be interesting outputs for monuments/locations which are not in the given dataset.
- You can also try and check for non-dataset monument images online of the classes which we have trained and see different predictions which the model makes to similar looking landmarks.

```
Actual Label: nload (1)
Predicted Label in order: Is this picture of the
 Sydney Harbour Bridge, Sydney Opera House, or Monumento a la Revolucion?
```



```
Actual Label: nload
Predicted Label in order: Is this picture of the
 Eiffel Tower, Temple of Olympian Zeus, or Stockholm City Hall?
```



Submission provides at least three possible points of improvement for the classification algorithm.

```
Answer: (Three possible points for improvement)

The outputs are pretty good
Possible points for improvement: 1- Providing more images to train 2- Doing more data a
ugmentations techniques 3- Maybe Trying other transfer learning models
```

Some more points related to data ->

- There maybe class imbalance as well so check if that is the case. If yes then maybe using SMOTE to oversample minority classes might help
- More data is needed as you rightly mentioned for different classes as there are only 50 classes and the number of samples are also low so always try to think also in terms of a data-centric approach while starting out the project as fewer data will take you to a certain accuracy only but working on data might help you push boundaries even more.
- If you are interested, there was a data centric competition just launched a few months back so you can take part and understand while keeping the model fixed you can work more on data.

DOWNLOAD PROJECT

RETURN TO PATH

Rate this review

START