UDACITY

Return to Classroom

DISCUSS ON STUDENT HUB

Dog Breed Classifier

REVIEW

CODE REVIEW

HISTORY

Meets Specifications

Good job you finished all the parts that needed redoing nicely!

Some things to do to go a step beyond the assignment -

- Improving your transfer learning model and read more about fine tuning them as majorly in production, transfer learning models are used and its very cost effective and less time consuming process than training models from scratch.
- Deploying these models on the edge like a browser (using tf.js) or on iot/mobile devices using tflite.
- Visualizing the features learnt by CNN layers for complex architectures to understand how layers learn and based on that experience you can always decide on the number of layers for a complex/simple image classification task.
- Investigate training time bottlenecks as sometimes the GPU utilization is not what it seems and there are other bottlenecks as well like i/o from CPU/ slower storage drives.

All the best for your AI journey :)

Files Submitted

The submission includes all required, complete notebook files.

All files present. Glad that you also added the images on which you tested which are outside the dataset.


download.jpeg


download1.jpeg


download2.jpeg


download3.jpeg


images.jpeg


images1.jpeg


images2.jpeg

```
Why bigger batch size and drop out was not giving the desired result?
```

- Larger batch sizes don't generalize well because the model cannot travel far enough in a reasonable number of training epochs and especially while using optimizers like Adam and SGD, the lower the batch size usually is the better
  and as you have an imbalanced dataset, it would also get stuck with some classes which get randomly sampled more in a batch.
- Dropout also doesn't work well with smaller networks especially with imbalanced data

---

```
Usually Adam outperformed over SGD but why not in this case?
```

Both can give decent results here it depends on smaller batch sizes and learning rate but with these CNN architectures, its seen that SGD performs better.
https://shaoanlu.wordpress.com/2017/05/29/sgd-all-which-one-is-the-best-optimizer-dogs-vs-cats-toy-experiment/
https://ruder.io/optimizing-gradient-descent/

```
Ordering transforms, how its impact the final result? Change order between
random horizontal flip with random rotation provided the better result.
```

It depends on how your image quality changes when you change the order as for example if you rotate first and then flip clearly the image is augmented in a different way that may be degrading the features.
https://towardsdatascience.com/improves-cnn-performance-by-applying-data-transformation-bf86b3f4cef4

```
How can I use resnet18 more effectively?
```

- Not sure how you can use it effectively but if you are using it for transfer learning, you will need to find best possible hyperparams by something like a grid search/automl and there will still be limitation as to how much it can achieve. Do look at denser networks aka resnet50 or densenet101 which capture more features

which capture more features.
https://towardsdatascience.com/an-overview-of-resnet-and-its-variants-5281e2f56035

- `replaced all FC layers` Also, for transfer learning models, you should never remove the FC layers instead just add new layers or just retrain from scratch if you want to check its potential and have compute to train.

Step 1: Detect Humans

The submission returns the percentage of the first 100 images in the dog and human face datasets that include a detected, human face.

```
human_perc.update()
```

- Interesting that you printed out the counter object in tqdm as here we are testing on 100 images and this works. But usually, if you have to calculate % over more images, this approach would not be good.
- Tip - You can look at using MTCNN instead of haarcascades.
  This blog has very good insights at implementing face detection with multiple algorithms ->
  https://www.pyimagesearch.com/2018/09/24/opencv-face-recognition/

Step 2: Detect Dogs

Use a pre-trained VGG16 Net to find the predicted class for a given image. Use this to complete a `dog_detector` function below that returns True if a dog is detected in an image (and False if not).

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

Glad that you used these values for normalization as these derived from imagenet and all the images here are also taken from the larger imagenet dataset.
https://forums.fast.ai/t/is-normalizing-the-input-image-by-imagenet-mean-and-std-really-necessary/51338

```
    if use_cuda:
        output = output.cuda()

    class_index = output.data.numpy().argmax()
```

You can use torch.no_grad() after putting the model in eval mode here to avoid backpropogation as we are only using it for inference.

The submission returns the percentage of the first 100 images in the dog and human face datasets that include a detected dog.

```
detected dog in human_files:   0%|          | 0/100 [00:17<?, ?it/s]
detected dog in dog_files: 100%|██████████| 100/100 [00:20<00:00,  4.91it/
s]
```

Perfect accuracy!

Step 3: Create a CNN to Classify Dog Breeds (from Scratch)

Write three separate data loaders for the training, validation, and test datasets of dog images. These images should be pre-processed to be of the correct size.

```
train_transforms = transforms.Compose([
    transforms.RandomResizedCrop(224),
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(10),
    transforms.ToTensor(),
    normalize
])

test_transforms = transforms.Compose([
    transforms.Resize(255),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    normalize
])
```

- Using separate transformation for test/validation data that don't need augmentation is good. You used the same one for validation/testing to keep it more consistent and removed the augmentation from them so done well!
- 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

```
loaders_scratch = {'train': torch.utils.data.DataLoader(train_data, batch_
size=batch_size, num_workers=num_workers, shuffle=True),
                   'test': torch.utils.data.DataLoader(test_data, batch_si
```

```
ze=batch_size, num_workers=num_workers, shuffle=True),
                  'valid': torch.utils.data.DataLoader(valid_data, batch_
size=batch_size, num_workers=num_workers, shuffle=True)}
```

- Shuffling training set only is good so that randomness is added and you don't train on set of same classes in sequence. For validation and test data loaders, it is not needed to shuffle validation or test set.

Answer describes how the images were pre-processed and/or augmented.

```
Answer:

I have chosen to randomly resize to crop images to 224 pixels. Then adding
a random horizontal flip and rotation helps reducing overfitting on the tr
aining data. Also, random rotations of 40 degrees both left and right will
also help in overfitting issues. Finally, I normalize the tensors.
```

Rotations of 40 degrees are a bit too much as you are already flipping the images as well.

- Do visualize the images post augmentation to see whether the images are not undergoing degradation/extra cropping which removes certain features of dogs from the image.
- You can also look at class imbalance for some of the dog breeds and use something like SMOTE -> https://github.com/ufoym/imbalanced-dataset-sampler

The submission specifies a CNN architecture.

```
Net(
  (conv1): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1))
  (conv2): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1))
  (conv3): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1))
  (conv4): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1))
  (conv5): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_m
ode=False)
  (fc1): Linear(in_features=6400, out_features=500, bias=True)
  (fc2): Linear(in_features=500, out_features=133, bias=True)
)
```

5conv 2 fc quite Lenet like architecture!

Answer describes the reasoning behind the selection of layer types.

Some more tips and observations while building scratch models

- You can also try and imitate smaller architectures like Lenet or a simple 3conv2fc like structure for training scratch models as well.
- Include batchnorm here as you have an imbalanced dataset
- You can visualize each layer and filters 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 which 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 a drastic overfitting then try to first check your data and architecture while training from scratch.

Choose appropriate loss and optimization functions for this classification task. Train the model for a number of epochs and save the "best" result.

- The learning rate is decent and you can try to increase it as well (0.01-0.05) using SGD with smaller networks.
- Also, try to use something like lr_scheduler which automatically decays your learning rate if needed.
  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.
- 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 trained model attains at least 10% accuracy on the test set.

```
Test Loss: 3.019906


Test Accuracy: 26% (220/836)
```

```
Epoch: 30 Training Loss: 3.054046 Validation Loss: 3.082624
```

- There is a very less overfitting here as training loss and validation loss ar every close so good job!
- 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

Step 4: Create a CNN Using Transfer Learning

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

```
loaders_transfer = {'train': torch.utils.data.DataLoader(train_data, batch
_size=batch_size, num_workers=num_workers, shuffle=True),
                    'test': torch.utils.data.DataLoader(test_data, batch_si
ze=batch_size, num_workers=num_workers, shuffle=True),
                    'valid': torch.utils.data.DataLoader(valid_data, batch_
size=batch_size, num_workers=num_workers, shuffle=True)}
```

- Remember to make a copy of the older data loaders if you are reusing them as if you directly assign them here and if you make any changes here, it changes the original ones as well as it creates a shallow copy and not a deep copy.

```
transfer_loaders_scratch = loaders_scratch.copy()
```

- You can create a copy of the data loaders by using .copy() and no need to redefine new ones.

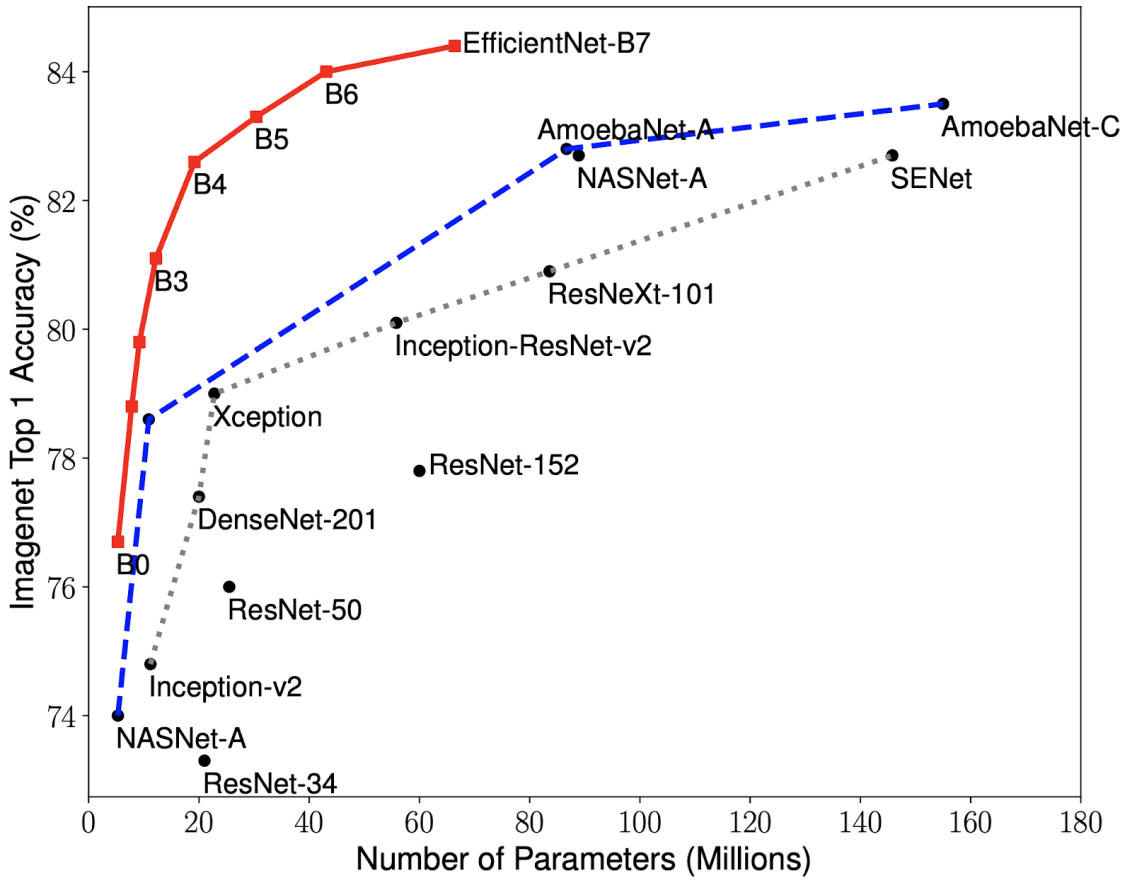
```
model_transfer = models.vgg16(pretrained=True)
```

- Good choice of Vgg16

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

- You can also look at other networks than VGG like Resnet50, Inception , Xception, Densenet which work well on imagenet in general as these classes are taken from the imagenet indeed for class label indexes as 268 is the last breed and then you have the wolves.\
  https://gist.github.com/yrevar/942d3a0ac09ec9e5eb3a
- 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.



Train your model for a number of epochs and save the result wth the lowest validation loss.

You could have just called the training function we used previously directly here.
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.

```
Test Loss: 0.611963


Test Accuracy: 80% (676/836)
```

- This is good accuracy.

- You can try using denser networks/networks with more parameters like Densenet and also checking the augmentations to the data to see if this can be pushed to 90s.

The submission includes a function that takes a file path to an image as input and returns the dog breed that is predicted by the CNN.

```
normalize = transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                 std=[0.229, 0.224, 0.225])

    transform = transforms.Compose([
        transforms.Resize(255),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        normalize
    ])
```

Tip - You can call the same preprocessing which you did to test/validation image here in general when using for inference for consistency.

```
```
    output = model_transfer(img_tensor)
    _, prediction = torch.max(output.data,1)
    breed_name = class_names[prediction-1]
```

Additionally, if you want you can also check which breeds and probabilities are occurring per prediction to see top n results, instead of just printing the argmax index value as that would help you to give more information about the possible near resembling breeds.
https://pytorch.org/docs/stable/generated/torch.topk.html

## Step 5: Write Your Algorithm

**The submission uses the CNN from the previous step to detect dog breed. The submission has different output for each detected image type (dog, human, other) and provides either predicted actual (or resembling) dog breed.**

Done well.

- The algorithm needs you to first check for dog detection, then human detection and if both are not there then no class should be returned as dog detector is more accurate that human detection model which we have it also makes sense to use it first to reduce wrong detections. Your code first checks the last case first which is also fine.
- Additionally, if you want, you can also make some funny dog filters to add on human faces by detecting landmark points and then making a fun application
```

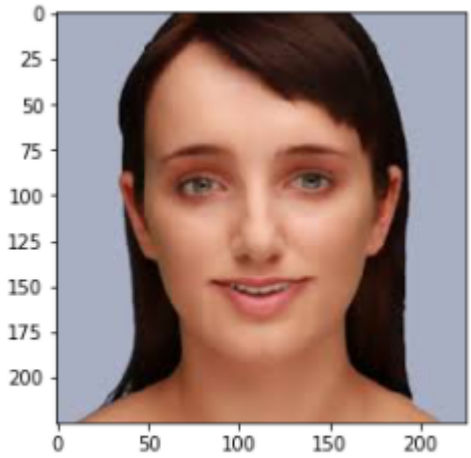
https://www.pyimagesearch.com/2017/04/10/detect-eyes-nose-lips-jaw-dlib-opencv-python/

Step 6: Test Your Algorithm

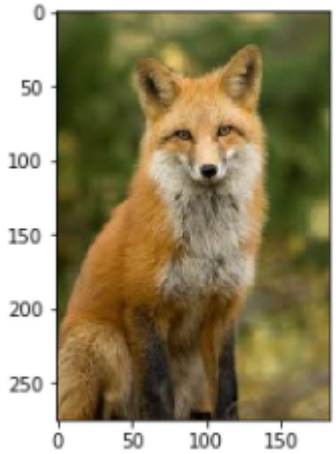
The submission tests at least 6 images, including at least two human and two dog images.

Glad that you tested on images that are not in the dataset as well as the edge case of not having any human or dog in the image aka an animated cat.
Do look at the face detector as one of the image of fox was detected as human

```
no dogs or humans in the picture
```



```
This is a human picture who looks like Petit basset griffon vendeen
```



```
This is a human picture who looks like Norwegian elkhound
```

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

```
Try more on resnet18 as my earlier trys were not successful.
For Vgg16, can try to replace last two layers or all three fully connected
layers to see change in output.
Try on more different pre trained models.
What if we pass wolf or fox images.
```

- Do try this for resnet by testing on simple dataset first -> https://www.kaggle.com/pintu161/transfer-learning-in-pytorch-using-resnet18
- Additionally think of class imbalance problem too and how you can tackle that here
- Also think of what would happen if the image has multiple dogs or human faces.

DOWNLOAD PROJECT

RETURN TO PATH