UDACITY

‹ Return to Classroom

DISCUSS ON STUDENT HUB

Dog Breed Classifier

REVIEW

HISTORY

Requires Changes

12 specifications require changes

Well Done!

This is a great first submission. You have coded the network pretty well. Although a couple of things need to be rectified. I have added comments to help you rectify those.

Hope it helps!

Files Submitted

The submission includes all required, complete notebook files.

Please consider submitting the IPython notebook too.

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.

Great!

```
human_detect: 0.98
dog_detect: 0.17
```

The HaarCascade classifier is used correctly to test the performance of detecting humans in dogs and human images.

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).

The VGG16 model is loaded correctly. The range of indices are rightly chosen in the `dog_detector` method.

```
    return (151 <= index and index <= 268) # None # true/false
```

Suggestion

It's highly recommended to use the normalization transform too.

Required

It's mandatory to change the mode of model to evaluation mode before using it for testing. I encourage you to check this Q&A: https://discuss.pytorch.org/t/model-eval-vs-with-torch-no-grad/19615

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

```
human_detected_as_dog: 0.0
dog_detected_as_dog: 0.8
```

The results can be improved by further using the normalization transform: `transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])`. You can see a significant improvement in the results.

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.

Great Efforts!

The set of transforms are well coded. It's great that you have used the augmentation techniques with the training data transform. The normalization transform is also rightly used.

Required

I noticed that you have missed shuffling the training data. Due to this, the model is not predicting correctly. This can be done by passing `shuffle=True` in:

```
train_loader = torch.utils.data.DataLoader(train_data, batch_size=32)
```

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

Consider working on the above suggested thing and update the answer.

The submission specifies a CNN architecture.

Awesome!

The convolution neural network is perfectly coded up. You have used the combination of convolution layers, pooling layers and batch normalization to build the feature detector part.
The final fully connected layer helps to build the classifier part.

Suggestions

- It would be better if you use the dropout layer in between the convolution layers too. This helps to reduce the chance of overfitting.
- Consider using 1 more fully connected layer with the final classifier part. This helps the model to perform well.

Answer describes the reasoning behind the selection of layer types.

Choose appropriate loss and optimization functions for this classification task. Train the model for a

number of epochs and save the "best" result.

```
criterion_scratch = nn.CrossEntropyLoss() #None

### TODO: select optimizer

optimizer_scratch = torch.optim.SGD(model_scratch.parameters(), lr=0.01) #
None
```

The LossFunction and SGD optimizer are used perfectly. The learning rate as 0.01 is acceptable.

The trained model attains at least 10% accuracy on the test set.

The network is getting trained but due to the issue with preprocessing, it's not learning the patterns.

Step 4: Create a CNN Using Transfer Learning

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

Good Job!

The VGG19 network is a great choice for transfer learning. It's loaded correctly and customization is done for using it to solve breed classification problem:

```
model_transfer.classifier[6] = nn.Linear(4096, 133, bias=True)
```

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

The VGG19 is acceptable for this step. It would be great if you mention some of the steps chosen to customize this for solving breed classification problem.

The second part of question asks to justify the reason of using this specific model in place of other model. One such reason can be that VGG19 is a simple model that worked pretty well on ImageNet dataset.

Suggestion

You can try using other transfer learning models like VGG with Batch normalization, ResNet etc.

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

The notebook shows some checkpoint related error. Consider debugging it and try to rectify this.
If you face any challenges, you can reach out to us via https://knowledge.udacity.com/

Accuracy on the test set is 60% or greater.

Try using the model to test the performance.

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.

The `process_image_to_tensor` function is not defined in the notebook. Consider attaching its code too.
It's mandatory to change the mode of model to evaluation mode before using it for testing. I encourage you to check this Q&A: https://discuss.pytorch.org/t/model-eval-vs-with-torch-no-grad/19615

Step 5: Write Your Algorithm

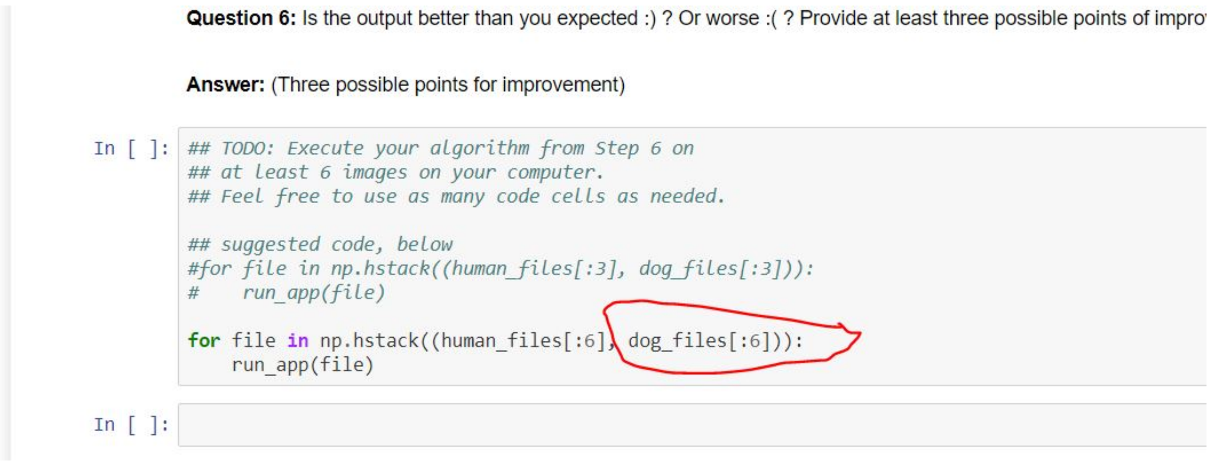
The submission uses the CNN from the previous step to detect dog breed. The submission has different output for each detected image type (dog, human, other) and provides either predicted actual (or resembling) dog breed.

The run_app method is well coded. It uses the model to classify the image.

Step 6: Test Your Algorithm

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

Please consider testing the model on a diverse set of images.



Question 6: Is the output better than you expected :) ? Or worse :(? Provide at least three possible points of impro

Answer: (Three possible points for improvement)

```
In [ ]: ## TODO: Execute your algorithm from Step 6 on
        ## at least 6 images on your computer.
        ## Feel free to use as many code cells as needed.

        ## suggested code, below
        #for file in np.hstack((human_files[:3], dog_files[:3])):
        #    run_app(file)

        for file in np.hstack((human_files[:6], dog_files[:6])):
            run_app(file)

In [ ]:
```

The dog_files were already fed to the model at the time of training. You should be using a new set of images to test the model.

I encourage you to try clicking some of the images or downloading a few from internet to test the model.

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

Consider sharing your observations to improve the model's performance.

RESUBMIT

DOWNLOAD PROJECT

Best practices for your project resubmission

Ben shares 5 helpful tips to get you through revising and resubmitting your project.

Watch Video (3:01)

RETURN TO PATH