UDACITY

Logout

< Return to "Deep Learning" in the classroom

DISCUSS ON STUDENT HUB

Dog Breed Classifier

REVIEW | CODE REVIEW | HISTORY

Meets Specifications

Good Job!

You have successfully passed the project. The network is well coded. Although it's recommended to test the final model with a new set of images that it hasn't seen yet. Else, it can lead to some bias and data leakage.
To further improve the results, I encourage you to try using other models like Xception, ResNeXt.

Keep up the great work!

Files Submitted

The submission includes all required, complete notebook files.

The set of files are correctly submitted.

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.

```
Detected Face in 99 percentage of human images.
Detected Face in 6 percentage of dogs images.
```

Haarcascade classifier is working pretty well. ✌
It's great that you have tried the face_recognition library too.

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 rightly loaded. Transforms are also coded correctly.

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 1 percentage of human images.
Detected dog in 100 percentage of dogs images.
```

It's working pretty well with our first 100 images. ✅

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.

The dataloaders and transforms are well coded for the train, test and validation dataset.

I noticed that you have used num_workers = 8. This sometimes gives error, so it's better to keep it zero. You can read a bit about it on this page.

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

The answer mentions all the steps chosen to pre-process the images.

The submission specifies a CNN architecture.

Perfect!

The CNN is well coded. Our combination of Convolution layers work like a feature selector and FC layers work like classified.
It's great that you have used 5 Convolution layers with ReLU, Maxpooling and Batch Normalizaiton layers.
The set of fully connected layers also help to build the final part of network.

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()

### TODO: select optimizer
optimizer_scratch =  optim.SGD(model_scratch.parameters(), lr=0.01, momentum=0.9)
```

The loss function and SGD optimizer are used perfectly. I encourage you to try RMSProp optimizer too.

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

Amazing!

```
Test Loss: 3.331998


Test Accuracy: 21% (178/836)
```

The model is working pretty well. ✌

Step 4: Create a CNN Using Transfer Learning

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

It's great that you have used the DenseNet161. The final fully connected layer is replaced with a new combination.

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

Perfect!

You have mentioned the abstract info about densenet161 model.
It would great if you share some stats too. The densenet161 is the best performing model among other available densenet models:

Rate this review

The 1-crop error rates on the imagenet dataset with the pretrained model are listed below.

Model structure	Top-1 error	Top-5 error
densenet121	25.35	7.83
densenet169	24.00	7.00
densenet201	22.80	6.43
densenet161	22.35	6.20

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

Training the model for 25 epochs is acceptable. I encourage you to try training it for more epochs and observe the difference in performance of model.

Accuracy on the test set is 60% or greater.

Awesome!

```
Test Loss: 0.980882


Test Accuracy: 85% (714/836)
```

The network is well trained. It's test accuracy as 85% is perfect.

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 predict_breed_transfer() function takes in the path of image and uses the model to predict the breed of dog.

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.

Step 6: Test Your Algorithm

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

The model is tested on a diverse set of images. It's working pretty well at classifying most of the images. Although you have used the same set of images that were used to train the model. It can create a bias while predicting the classes.
It's recommended to test the performance of model only with totally new images that model hasn't seen yet.

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

The StepLR is a great approach. You should definitely try using it. This helps the model to converge. I encourage you to try other transfer learning models too like Xception, ResNeXt etc.

DOWNLOAD PROJECT

RETURN TO PATH

Rate this review