UDACITY

‹ Return to "Deep Learning" in the classroom

DISCUSS ON STUDENT HUB

Dog Breed Classifier

REVIEW

HISTORY

Meets Specifications

Great submission! 👍

All functions were implemented correctly, the detectors easily meet the required accuracies and the final algorithm seems to work quite well.

Files Submitted

The submission includes all required, complete notebook files.

All required files are included 👌

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.

```
Number of faces detected: 97 out of 100 images.  Percentage: 97.0
Number of incorrect faces detected: 18 out of 100 images.  Percentage: 18.0
```

Perfect!

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).

Well done! You implemented the code to load the image, normalize it, resize/crop it and extract the result from VGG16.

SUGGESTION

The best results are obtained by first resizing to 256x256 pixels and then center cropping to 224x224 pixels.

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

```
Number of dogs detected: 0 out of 100 humans.  Percentage: 0.0
Number of dogs detected: 93 out of 100 dogs.  Percentage: 93.0
```

Great! Note that the CNN dog detector has many fewer false positives as the face detector.

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 preprocessing is correctly implemented, the images are resized to 224x224 pixels and appropriately normalized.

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

Good description of the preprocessing!

Note that for data augmentation we want the images to still resemble the original images, it is a good practice to plot some images to see the effect.

The submission specifies a CNN architecture.

The architecture follows a typical VGG style with multiple convolutional and max pooling layers, good job!

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.

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

```
Test Accuracy: 17% (150/836)
```

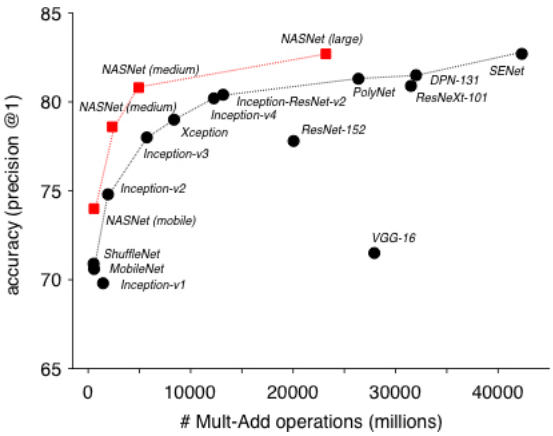
This is really impressive for a from scratch model with such limited data, well done!

Step 4: Create a CNN Using Transfer Learning

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

```
model_transfer = models.resnet50(pretrained=True)
```

ResNet-50 is a good choice. You could try some of the other pre-trained models provided by `torchvision.models` (https://pytorch.org/docs/stable/torchvision/models.html) to see which one can give you the highest accuracy. For practical applications there is always a trade-off between network size (number of operations) and accuracy:



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

Great description of your approach! Also note that our dataset is very similar to the Imagenet data set so we should see good results with transfer learning.

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

Accuracy on the test set is 60% or greater.

```
Test Accuracy: 88% (742/836)
```

The test accuracy exceeds the required 60%, well done!

Compared to training from scratch (step 3), transfer learning results in a pretty impressive accuracy. To further increase the accuracy you could try to lower the learning rate once validation accuracy stabilizes or use a learning rate schedule, see https://pytorch.org/docs/stable/optim.html#how-to-adjust-learning-rate.

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.

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.

Your algorithm gives the correct output! 👍

To give some extra information to the user you could think about adding the predicted probability of a dog breed and showing an (example) image of the predicted dog breed.

Step 6: Test Your Algorithm

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

The algorithm was sufficiently tested 👌

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

I would probably train the transfer model some more to improve it. Another way potentially improve it will be to do additional data augmentation like luminance adjustment or more rotation.

These are all good improvements!

Since neural networks are very data hungry the biggest gain in accuracy, one can usually get from feeding more data to your network for training.

DOWNLOAD PROJECT

RETURN TO PATH