UDACITY

‹ Return to "Deep Learning" in the classroom

Dog Breed Classifier

REVIEW

HISTORY

Meets Specifications

Awesome job on the project! Now you know how what-dog works.
To make the net run faster, we could possibly use some compression.


Files Submitted

The submission includes all required, complete notebook files.

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.

Nice!

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

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.

Good work! The means can be calculated dynamically with something like `train_set.train_data, axis=(0,1,2))/255)`: https://discuss.pytorch.org/t/normalization-in-the-mnist-example/457/12

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

The submission specifies a CNN architecture.

Great model! Usually I use batch normalization because it trains faster and performs better (higher accuracy). I also tend to use ELU instead of ReLU.

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.

Nice work! Often the `adam` optimizer seems to work best: https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/
Here is a comparison of some different optimizers: http://ruder.io/optimizing-gradient-descent/

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

Step 4: Create a CNN Using Transfer Learning

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

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

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

Yep! You got it.

Part of the reason some of these nets work really well and better than others is because they were trained on more dog images. You can see the classes for VGG16 include only a handful of dogs, Xception and inception have many more. Of course our net trained from scratch did not have nearly enough time or complexity to learn the statistical differences between dog breeds, since the accuracy was so low.

This is a list of classes for VGG-16
http://image-net.org/challenges/LSVRC/2014/browse-synsets

Also, the deeper layers find more complex patterns: http://www.coldvision.io/2016/07/29/image-classification-deep-learning-cnn-caffe-opencv-3-x-cuda/

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.

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.

Step 6: Test Your Algorithm

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

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

DOWNLOAD PROJECT

RETURN TO PATH