UDACITY

Back to Deep Learning Nanodegree

Dog Breed Classifier

REVIEW

CODE 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.

If you want to dive deeper into CNNs/image classification and challenge yourself after finishing this project:

1. I would recommend first to read up on how to generate the bottleneck features yourself. See https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html under the heading " Using the bottleneck features of a pre-trained network: 90% accuracy in a minute" on how to do this with Keras.
2. After that you should be ready to take part in one of the playground competitions on Kaggle like https://www.kaggle.com/c/dog-breed-identification and https://www.kaggle.com/c/plant-seedlings-classification. Check out the kernels to learn techniques and tricks from other people and see if you can get a top leaderboard score!

Files Submitted

The submission includes all required files.

Step 1: Detect Humans

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

```
human detected in human pics: 100 %, should be 100%
human detected in dog pics: 11 %, should be 0%
```

Good! 👌

The submission opines whether Haar cascades for face detection are an appropriate technique for human detection.

Step 2: Detect Dogs

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

```
dog detected in human pics: 0 %, should be 0%
dog detected in dog pics: 100 %, should be 100%
```

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)

The submission specifies a CNN architecture.

Good choice for the architecture:

1. Multiple convolutional layers with an increasing number of filters and small filter sizes. Note that it's more common to use 3x3 filters (see for example VGG), resulting in an increased receptive field but still a low number of parameters.
2. Global average pooling and max pooling to reduce the dimensionality before the dense layers to prevent overfitting and reduce the number of parameters of the model.
3. Dense layers to link the features to the 133 dog breed classes with a `softmax` activation function for multi class classification.
4. Additional dropout to further reduce overfitting.

This is a good resource discussing he rationale behind various basic CNN architectures: http://cs231n.github.io/convolutional-networks/#architectures.

Tip: add batch normalization before every convolutional or dense layer, this will normalize the features before every layer and will result in a faster training time and a boost in accuracy, see https://keras.io/layers/normalization/ for the Keras documentation.

The submission specifies the number of epochs used to train the algorithm.

To get everything out of your network and data, the epochs should be chosen such that the validation accuracy is no longer increasing. You can do this manually or, better, use the early stopping callback function of Keras (https://keras.io/getting-started/faq/#how-can-i-interrupt-training-when-the-validation-loss-isnt-decreasing-anymore).

The trained model attains at least 1% accuracy on the test set.

Step 5: Create a CNN to Classify Dog Breeds

The submission downloads the bottleneck features corresponding to one of the Keras pre-trained models (VGG-19, ResNet-50, Inception, or Xception).

To get the highest accuracy I would recommend to try all bottleneck features out as transfer learning doesn't take a lot of time, compared to training from scratch.

Currently ResNet and Xception are very popular networks producing state of the art results, see:

- ResNet (original) https://arxiv.org/abs/1512.03385
- WideResNet https://arxiv.org/abs/1605.07146
- SENet https://arxiv.org/abs/1709.01507
- Xception https://arxiv.org/abs/1610.02357

The submission specifies a model architecture.

The submission details why the chosen architecture succeeded in the classification task and why earlier attempts were not as successful.

The submission compiles the architecture by specifying the loss function and optimizer.

Good work!

Instead of using `rmsprop` you might want to try out `adam`, it's usually a better choice as it's easier to configure and gives superior results. Check out https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/ for more information.

The submission uses model checkpointing to train the model and saves the model weights with the best validation loss.

The submission loads the model weights that attained the least 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 6: Write Your Algorithm

The submission uses the CNN from Step 5 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 (the `.predict()` function of the Keras model returns the probabilities) and showing an (example) image of the predicted dog breed.

Step 7: Test Your Algorithm

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

DOWNLOAD PROJECT

RETURN TO PATH

Rate this review