UDACITY

PROJECT

Dog Breed Classifier

A part of the Deep Learning Nanodegree Program

PROJECT REVIEW

CODE REVIEW

NOTES

SHARE YOUR ACCOMPLISHMENT!

Meets Specifications

You did great on this 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 and image classification:

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 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.

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

Got your answer, it was actually meant to think about what you could potentially do - but great you made this improvement by making some small changes. Two more things you could think about:

1. Combine various algorithms (Haar cascades and CNNs) to improve overall accuracy.
2. Let the app check realtime if a face is detected, similar to how QR code scanner apps work.

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.

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.
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.

Check out http://cs231n.github.io/convolutional-networks/#architectures for more on the rationale behind various CNN 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.

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).

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 architecture of your network for transfer learning is good, but actually best results (I have seen so far) have been obtained by just having one dense layer after the pretrained features. The bottleneck features apparently contain enough information to distinguish the dog breed classes, adding more layers overcomplicates the classification.

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

Well done here!

Instead of using `rmsprop` you might want to try out `adam`, it's usually a better choice for more complex problems 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.

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

Student FAQ