UDACITY

< Return to "Deep Learning" in the classroom

# Dog Breed Classifier

REVIEW

CODE REVIEW

HISTORY

## Requires Changes

1 SPECIFICATION REQUIRES CHANGES

Good first submission on this project! 👍

You have shown a good understanding of CNNs, both in training them from scratch (step 3) and using transfer learning (step 5). There are few things left to do, please see my comments below for the details.

**I tried with data augmentation but was getting array dimension mismatch error so i had to comment it out.**

You tried to apply data augmentation on the bottleneck features, but that's not possible. The features are of the size 1x1x2048 and are not image-like, so rotating, resizing etc don't make sense. If you want to apply data augmentation I would recommend doing it in step 3 (from scratch model).

### Files Submitted

| |
|---|
| **The submission includes all required 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 with a detected human face.**

```
Human face accuracy is 100.00%
Dog face accuracy is 11.00%
```

Perfect!

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

It seems you missed to answer this question:

- **Question 2**: This algorithmic choice necessitates that we communicate to the user that we accept human images only when they provide a clear view of a face (otherwise, we risk having unneccessarily frustrated users!). In your opinion, is this a reasonable expectation to pose on the user? If not, can you think of a way to detect humans in images that does not necessitate an image with a clearly presented face?

Note that this question is not optional because it is a rubric item (implementing an alternative face detector is optional). Please include an answer, where in you discuss whether Haar cascades are an appropriate technique for human detection, to meet the specifications.

## 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.**

```
Human face accuracy is 0.00%
Dog face accuracy is 100.00%
```

Great! Note that the CNN dog detector has many fewer false positives as the face detector: 0 vs 11. Therefore, in the algorithm (step 5) it's better to place the dog detector before the face detector to get more accurate results.

## Step 3: Create a CNN to Classify Dog Breeds (from Scratch)

**The submission specifies a CNN architecture.**

Good choice of the architecture and well explained!

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

The number of epochs you used for training is sufficient to get the required accuracy, good job!

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.**

```
Test accuracy: 3.9474%
```

You easily beat the required 1% accuracy. 👍

Some tips to improve the accuracy further:

- Be sure to train for sufficient epochs, it helps to plot training and validation accuracy to see when your model stops improving and starts overfitting.
- Add more convolutional layers (this will require a longer training time though).
- Using multiple max pooling layers to reduce the size to about 10x10 and apply global average pooling.
- Add batch normalization after every convolutional or dense layer see https://keras.io/layers/normalization/ for the Keras documentation.
- Augment the training data, see https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html.

## 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).**

```
bottleneck_features = np.load('/data/bottleneck_features/DogResnet50Data.npz')
```

ResNet-50 is a good choice! Currently ResNet-based models 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.**

Good description of your approach and excellent explanation of why the architecture is suitable for the current problem. 👌

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

Good work!

Instead of using `rmsprop` as optimizer 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.**

```
Test accuracy: 81.8182%
```

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://machinelearningmastery.com/using-learning-rate-schedules-deep-learning-models-python-keras/

**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! 👍

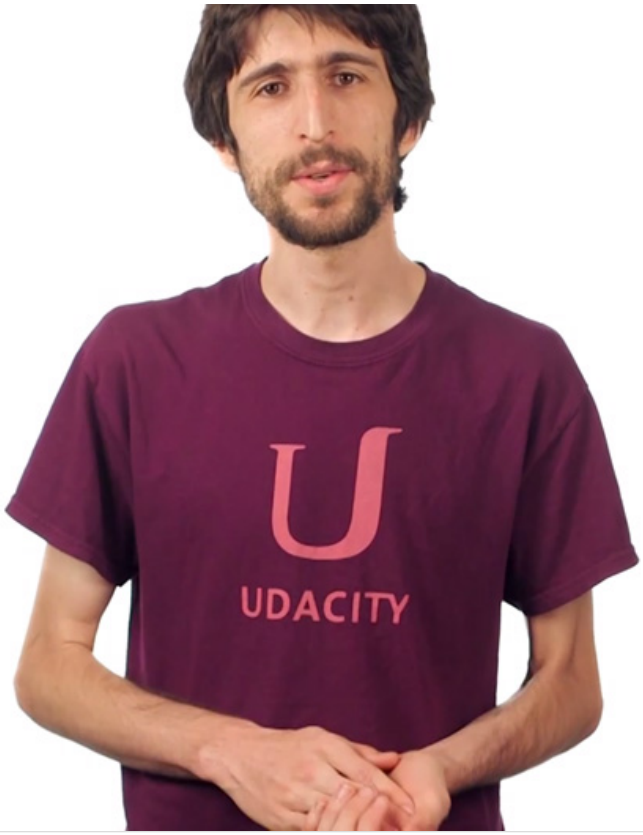
To give some extra information to the user you could think about adding the predicted probability of a dog breed (the `.predict()` function of the Keras model returns the probabilities) and showing an (example) image of the predicted dog breed.

## Step 7: Test Your Algorithm

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

RESUBMIT

DOWNLOAD PROJECT




## Best practices for your project resubmission

Ben shares 5 helpful tips to get you through revising and resubmitting your project.

Watch Video (3:01)

RETURN TO PATH