Dog Breed Classifier

REVIEW

HISTORY

Meets Specifications

Congratulations

Great submission!

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

Suggesting some further readings:

- Using Convolutional Neural Networks to Classify Dog Breeds
- How To Improve Deep Learning Performance.
- A Guide to TF Layers: Building a Convolutional Neural Network.

Suggesting some further actions to make your project even better!

1. **Augment the Training Data**: Augmenting the training and/or validation set might help improve model performance.
2. **Turn your Algorithm into a Web App**: Turn your code into a web app using Flask or web.py!
3. **Overlay Dog Ears on Detected Human Heads**: Overlay a Snapchat-like filter with dog ears on detected human heads. You can determine where to place the ears through the use of the OpenCV face detector, which returns a bounding box for the face. If you would also like to overlay a dog nose filter, here some nice tutorials for facial keypoints detection.
4. **Add Functionality for Dog Mutts**: Currently, if a dog appears 51% German Shephard and 49% poodle, only the German Shephard breed is returned. The algorithm is currently guaranteed to fail for every mixed breed dog. Of course, if a dog is predicted as 99.5% Labrador, it is still worthwhile to round this to 100% and return a single breed; so, you will have to find a nice balance.
5. **Experiment with Multiple Dog/Human Detectors**: Perform a systematic evaluation of various methods for detecting humans and dogs in images. Provide improved methodology for the face_detector and dog_detector functions.

Files Submitted

✓

The submission includes all required, complete notebook files.

Your code was functional, well-documented, and organized.
Well done!

Suggested reading:

- Google Python Style Guide
- Python Best Practices

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.

Well done!
Percentage of human images with detected human faces: **98%**
Percentage of dog images with detected human faces: **17%**
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).

Good job!
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.

Great!
Percentage of first 100 images in human images with detected dog: **1%**
Percentage of first 100 images in dog images with detected dog: **100%**

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 job creating the train_loader, valid_loader and test_loader data loaders and also resizing and cropping the images to a square size of 224.

It is suggested to use PyTorch's transforms and DataLoader class

Suggested reading about the discussion of what machine learning framework should be used.

- TensorFlow vs. Pytorch.
- Tensorflow or PyTorch : The force is strong with which one?
- What is the best programming language for Machine Learning?

✓

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

Nice work describing your preprocessing steps in question 3.
Good job with data augmentation!

✓

The submission specifies a CNN architecture.

Overall you did a fine job implementing your CNN architecture!

Comments about you code:

- You chose to use maxpooling layers to decrease the spatial size.
- Adding some dropout layers to reduce the risk of overfitting was a very sensible decision.
- You used ReLU activations between convlayers to introduce non-linearity and to allow gradients to flow backwards through the layer unimpeded.
- You could have used Batch normalization to transform the input to zero mean/unit variance distributions.

Batch normalization is becoming very popular to further improve the performance of the model. Batch normalization layers avoid covariate shift and accelerate the training process

Look at these suggested readings:

- Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift
- Understanding the backward pass through Batch Normalization Layer

When you have any time look at this discussion on how do you decide the parameters of a Convolutional Neural Network for image classification and another one, about how can you decide the kernel size, output maps and layers of CNN.

✓

Answer describes the reasoning behind the selection of layer types.

Nice work describing the reasoning behind the selection of layer types in question 4.

✓

Choose appropriate loss and optimization functions for this classification task. Train the model for a number of epochs and save the "best" result.

Good job using the **CrossEntropyLoss()** loss function as it is suitable for classification tasks such as this.

Comments about you code:

- You achieved a good convergence with 25 iterations (epochs). A very high number of iterations must yield overfitting. One way to define the number of iterations would be to look at loss stabilization showed by the graph. You must choose the number of epochs such that the loss on the training set is low and the loss on the validation set isn't increasing
- You chose learning rate of 0.001. You should avoid using learning rates lower than enough to get the network to converge, there really isn't a benefit in a really low learning rate.
- Good choice using **Adam optimizers** for updating the weights of the discriminator and generator. Here Sebastian Ruder explains Adam optimizers a little bit.

✓

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

Test Accuracy: 16% (138/836)

This is pretty high 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.

VGG16 was a good choice.
You could try some of the other pre-trained models provided by torchvision.models to see which one can give you the highest accuracy.

✓

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

A good justification has been given as to why the architecture succeeded.
Good use of transfer learning.
Well done!

Transfer learning is a machine learning method where a model developed for a task is reused as the starting point for a model on a second task.

Here are some good documents you can check out for more insight on transfer learning:

- CS231n Convolutional Neural Networks for Visual Recognition
- Transfer Learning Introduction

✓

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

Well done training the model with **Adam optimizer** and **CrossEntropyLoss()** function.

This is a nice article that would help enrich your knowledge about them:
Loss Functions and Optimization Algorithms. Demystified. may help you understand the uses of these functions and

algorithms.

✓

Accuracy on the test set is 60% or greater.

Test Accuracy: 75% (627/836)

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

✓

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.

A function **predict_breed_transfer()** was correctly defined to take a file path as input and return the breed predicted by the CNN.
Great job!

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.

The implemented algorithm in **run_app()** uses the CNN (**predict_breed_transfer()** function) to predict dog breeds.

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.

Nice job testing the algorithm on several human and dog images.
The predicted dog breeds for the images are excellent. Impressive results!

You should also test it on images with neither a dog nor human.

The algorithm was sufficiently tested

✓

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

DOWNLOAD PROJECT