Dog Breed Classifier

- Review
- History

Meets Specifications

Hello learner, I am very happy that you passed all the points of the project! You have understood the scope of the convolution, transfer learning and you made an app that classifies dog breeds. Congratulations! For more resources

- my webinars about the Deep Learning
- NNs CNNs and RNNs(next project!)
- How CNNs are working
- Filters, stride, analysis of CNNs
- Visual Recognition lecture of Stanford
- How to attack the CNNs
- A new approach based on CNNs, the caspule neural networks

Turn your code into a web app using Flask!
Good luck with the next projects. You are gonna learn a lot!
Please, don't forget to rate my work as project reviewer! Your feedback is very helpful and appreciated

Files Submitted

The submission includes all required, complete notebook files.

indeed

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.

The right function is used.
Face detection using OpenCV and Python: A beginner's guide

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).

It is done, with the right transformations, normalization and the max().
Right that CPU is used for the prediction.

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

It is done with the right classes and it seems that the model works fine!

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.

Shuffling is appropriate only for the training.
Nice dictionary for the data loaders.
My webinar for data preparation and training.

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

You are right! With RandomResizedCrop(), RandomRotation() and RandomHorizontalFlip() you have augmented the training data.
The PyTorch documentation is well read!
My webinar with section about how to load image data.

The submission specifies a CNN architecture.

In any case, if you put more layers, the model could make more calculations and it could gain more accuracy.
It could go "deeper". Although 5 convolution layers are enough.
Right output number.
Here is my webinar about Convolution Neural Networks

Answer describes the reasoning behind the selection of layer types.

The striding convolutions are better, so stride of 2 may have better performance.
The architecture for the CIFAR-10 is nice. I would recommend to you something like the AlexNet. It was one of the first CNNs that made deep learning a trend. More complicated networks are not in the scope of the project and it is up to the student!

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

Saving and loading models are in a right way.
Adam is used. Adam(Adaptive Moment Optimization) uses momentum that accelerates the gradients.
Intro to optimization in deep learning: Momentum, RMSProp and Adam
All the optimization algorithms are included inside the optim package

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

You did it!

Step 4: Create a CNN Using Transfer Learning

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

You could use the data loaders from the previous step.
ResNet18 is used in a very generic way! You have substituted the classifier layer by using the fc property.
Common architectures in convolutional neural networks

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

You are gonna see the ResNet in the next chapters
The ResNet uses residual blocks Residual Networks on PyTorch.

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

The same optimization is used. You could see the difference.

Accuracy on the test set is 60% or greater.

By using the right tuning, such as specific learning rate, momentum etc, it could be achieved after 2-3 epochs.

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.

It is encapsulated in a function.

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.

You should use the predict_breed_transfer() for all 3 image types to see that your model classifies everything and identify false positives.

Step 6: Test Your Algorithm

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

One person is misclassified.

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

You can still improve it with more transformations! Look at the idea about the FiveCrop. Be careful, because it returns a tuple.