UDACITY

PROJECT

Dog Breed Classifier

A part of the Deep Learning Nanodegree Program

PROJECT REVIEW

NOTES

SHARE YOUR ACCOMPLISHMENT!

Meets Specifications

Fantastic submission and I hope you had lots of fun doing this. Good luck and happy learning 😄

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.

Well done!

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

Good Answer, and great comparison with full_body features. But faces are still the distinct features for us humans, that's why it's sometimes difficult for us to recognize even familiar people without seeing their face.

If you want to build a robust and state of the art face detector, augmentations, landmark estimations are a perfect way to start. A great article to get started if you are interested. This article is a summary of various models.

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.

Pretty impressive huh!

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

The submission specifies a CNN architecture.

Good job explaining your architecture and your chosen architecture is a solid choice.

1. Your convolution pyramid is solidly constructed!
2. Great use of GAP instead of Flatten. This adds a lot of regularization to the model in addition to the points that you've identified!

Suggestions

1. Also you could stagger multiple layers of convolutions before a single max pool (like how VGG does). This is a great way to extract more crisper features.
2. You could try adding dropouts to curb overfitting
3. You can try adding more fully connected layers. This will also increase the accuracy of your model.

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

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

Excellent! I hope it makes you appreciate just how hard training a CNN from scratch is!

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).

Fantastic job of choosing ResNet. After seeing it's performance earlier, it's an obvious choice! 😄

The submission specifies a model architecture.

A simple classifier is more than sufficient since the pretrained network would be doing most of the heavy lifting. So in that essence, your chosen architecture is perfect! Your addition of Dropouts is definitely required, as it's very easy to overfit while transfer learning. Good work 👏🏽

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

Good job explaining why this works! This gives us insight that you understand transfer learning.

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

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.

Awesome job reaching and crossing the rubric! 👏🏽

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.

That's a fantastic answer!

Some further steps you can take to debug why certain test images fail are as follows

- Try taking the softmax output and plot out the top 5 results and see if the correct class falls in any of the top 5 values.
- You can visualize the CNN to understand what aspects of the image is making the model predict a certain class, to see what's wrong or biased in your input data set.

Few other techniques to improve could be (but in most cases, you'll not be able to use the bottleneck data)

- Use augmentation and provide variations in your training set
- Consider retraining a small part of your pretrained network (you can fine tune the parameters of the CNN too)
- Regarding multiple objects in the image, you could turn your system into an object detection (detects bounding boxes for the objects) along with the class rather than a simple classifier.
- Regarding computational speed, other than trying to get a simpler model to work at the same accuracy, a GPU or a set of GPUs are the only solution. You can look into model compression and how most face detectors work on mobile today.

Suggestions to make the app more interesting

1. You can make your app even more interesting, by rather than just giving out the top class, sample all the classes based on the softmax probabilities (especially when it comes to humans). This way, each time the human uses the app, there is a chance he will get a different dog breed based on probability!
2. Showing the photo of the dog breed is a great idea and something we recommend to most students. Makes the app much more interactive and fun. You can even run some similarity tests and show the dog image that most resembles the human from the training dataset.

DOWNLOAD PROJECT

RETURN TO PATH

Rate this review

Student FAQ