Deep Learning for Dog Breed Prediction

Jackson Thetford, Mihiro Suzuki

Spring 2024
Software Design For Responsible Intelligent Systems
COE 379L, The University of Texas at Austin

https://github.com/jthet/machine-learning-projects/tree/main/dog-breed-prediction

Abstract

In this paper, we explore the application of convolutional neural networks (CNNs) for dog breed identification. This has proven to be a challenging task given the subtle inter-breed variations and intra-breed similarities among dogs, and many owners often not knowing the true breed of their dog, with the only way to find out being a DNA test. This project proposes a much quicker and similarly accurate way to identify a dog's breed with CNNs. Using the Stanford Dogs dataset, which consists of 20,580 images spanning 120 breeds, we implement and evaluate several neural network architectures to determine their effectiveness in classifying the breed of a dog from images. Our approach focused on transfer learning and fine-tuning techniques on well-known architectures such as Inception V3, LeNet-5, and VGG16.

The best-performing model, a modified Inception V3 network, achieved around 80% accuracy on testing data. This model has been deployed in a Flask-based server as a RESTful API that allows users to upload an image of their dog and receive a prediction of the dog's breed. The deployment is containerized using Docker to help installation and ease the reproducibility of the environment. With more time and resources we are confident a model with a much higher accuracy could be achieved as this model was trained on only 20 epochs. In future interactions of this project we would like to train several other Inception models with varying layers succeeding the base model, as well as training the model for longer.

After the model was made it was discovered that the model performs in an intriguing way when given images of humans. This discovery led the project to take a twist and turn into a way to find out which dog breed a human most resembles.

Introduction

The goal of this project is to develop a machine learning model that can accurately classify images of dogs by their breed. We expanded the scope of this model capability beyond simply categorizing dogs by their breed to, allowing people to see what breed of dog they look like. Furthermore, this technology has various other applications, such as breed-specific medical research, veterinarian breed identifications, and as an educational tool for children in schools.

Data

For this project we used the Stanford Dogs dataset, which was originally released for fine-grained image categorization. It originally comes from the ImageNet dataset and contains 20,580 images of 120 different dog breeds. Each image in the dataset is labeled by breed, helping our supervised learning process.

The dataset includes 120 dog breeds ranging from common breeds like Golden Retrievers and Poodles to more exotic breeds such as the Norwegian Lundehund and the Xoloitzcuintli. The wide range of breeds and the subtle differences and similarities between these breeds make this classification project much more challenging than a binary classification problem, such as making a model that classifies an image as a "dog" or "not a dog".

The images also differ significantly in terms of pose, size, lighting, background, and accessories (e.g., collars, hats). Some images are focused on the dog's face while others have humans interacting with the dog, the dog wearing clothes and accessories, and even multiple dogs.

The data can be found here: http://vision.stanford.edu/aditya86/ImageNetDogs/

CNN Architectures

This project focused on the following architectures. Note that the ANN is shown in the jupyter notebook, but was shut down after long training times, memory limitations, and poor performance.

- **ANN:** A dense, fully-connected network with varying layers and perceptrons.
- **LeNet-5 CNN:** A convolutional neural network based on the classical Lenet-5 architecture, adjusted for our image dimensions.
- **VGG:** A custom model with the VGG-16 architecture. https://arxiv.org/pdf/1409.1556.pdf
- **Inception V3:** A custom model using Google's Inception model architecture and weights. https://arxiv.org/abs/1512.00567

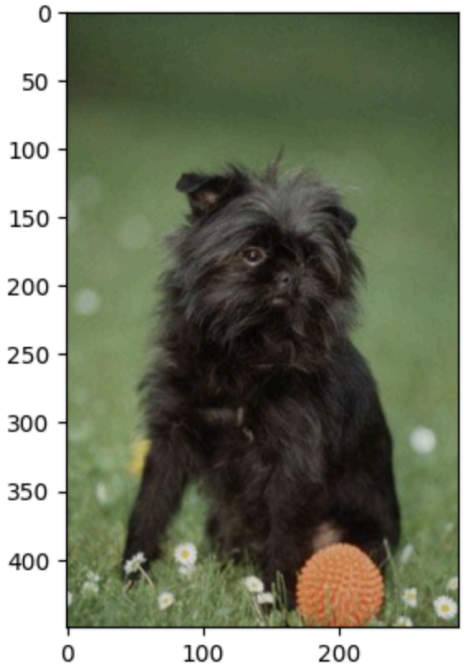
Methods employed

Data Preprocessing

1. **Data Loading:** The data was downloaded from the Stanford source and was loaded into the image folder in the repository. This allowed for the data to be pulled directly to the ipynb file. The images were original contained in individual folders by breed and were named inconspicuously, so there images were renamed by their dog breed followed by and index and then combined into one folder named "Images." This process can be seen in the GitHub repository at data/data_cleaning.ipynb.

2. **Data Investigation/Preprocessing:** The data was first manually investigated to recognize patterns. Then details such as image size, file type, color patterns were explored. In this step, the data was resized to 128, 128, 3 for consistency. The images were then normalized so they were ready for training. An example of an unprocessed image is shown below.

```
Image Format:  JPEG
Image Mode:  RGB
Image Size:  (289, 450)
datatype:  uint8
shape:  (450, 289, 3)
```



3. **Data Splitting:** The data was split into training and testing sets using the Scikit-learn train-test-split module.

Model Design, Training, and Evaluation

1. **Model Selection and Design:** While all the models in the "CNN Architectures" section were tested, the best 3 candidates were Inception V3, LeNet-5 CNN, VGG-16. (The ANN model was extremely computationally extensive, and took too long). In all cases the input layer expected a 3-dimensional vector of size (128, 128, 3), as the images were all resized to fit the architectures. The output layer for each model contained 120 perceptrons, the same as the number dog breeds of which we are using to classify a dog.

2. **Training:** Here is a summary of the model parameters that were used. In all cases the input was 128,128, 3 (the image size) and the output was 120 since there were 120 breeds to categorize.

3. **Evaluation:** The results will be explained in depth in a later section.

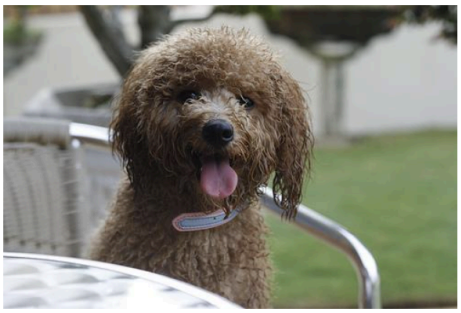
Model Deployment

The models were saved as .keras files using the supported tensorflow.keras api. This ensures that the models are portable and reproducible. A flask server was developed to serve the model over HTTP. The server includes the following endpoints.

Endpoint	Method	Description
`/model/info`	GET	Provides basic information about the currently loaded TensorFlow model, including its parameters.
`/model/models`	GET	Lists the available TensorFlow models that users can switch to, indicating the default model.
`/model/predict`	POST	Predicts a dog's breed, given a jpg picture of the dog.
`/model/change`	POST	Changes the TensorFlow model used by the server to a specified model.
`/model/summary`	GET	Provides a textual summary of the currently loaded TensorFlow model's architecture.
`/help`	GET	Provides an overview and usage examples for the available API endpoints.

Here is an example prediction on a human.

```
$ curl -X POST -F 'image=@./assets/jt-2-34.JPG' http://localhost:5000/model/predict
{
  "result": "miniature poodle"
}
```

More specific information on how to deploy the server, as well as example data is included in the readme file.

Results

LeNet-5 CNN: The LeNet-5 model performed very poorly, likely due to the insufficient data available per breed. Since there are only about 150 images per breed, the model had limited examples to learn from. Therefore, it lacked the ability to learn sufficiently.

This model, on our testing set, recorded a test loss of 4.4757 and a test accuracy of 0.0534

To fit this model, a validation split of 0.2 was used to maximize the amount of data to train on. The epochs were 15, with a batch size of 32, and custom callbacks. This model trained faster per epoch due to a smaller batch size but did not perform well.

The callbacks used were

- Checkpoint: saves the model or weights at specified intervals
- Earlystop: Stops the training process early if the validation loss stops improving
- Csvlog: keeps a log of training progress
- reduceLR: reduces the learning rate when the validation loss is not improving.

Shown below are the first 9 images in the testing set, along with the model's predicted value and the actual value.

Inception V3 (Best Model): The Inception V3 model showed significantly better results than the LeNet-5 CNN and an accuracy of approximately 78%. We strongly believe that with access to a GPU, and additional time for fine-tuning and training, the accuracy of this model could approach 90% and potentially up to 97%, as described in the paper "SR-GNN: Spatial Relation-aware Graph Neural Network for Fine-Grained Image Categorization" (A. Bera, et al. https://arxiv.org/pdf/2209.02109). The higher accuracy of Inception V3 is because of the weights it inherited from being trained on the ImageNet dataset. Therefore, although the model only had 150 images of each breed to train on, the pretrained weights were significant in the success of fine-tuning the model.

To fit this model, a validation split of 0.3 was used, and the epochs were 20, with a batch size of 128, and custom callbacks similar to before were used. This model used a higher batch size but can still perform the best.

This model, on our testing set, recorded a test loss of 1.1743 and a test accuracy of 0.7806

As we can see from the first 9 images of the testing data, the model got 7/9 correct. Additionally, the images it missed were somewhat unclear. Most notable, image 7 has a human in the picture and the dog is partially covered.

VGG - 16: The VGG model did not perform as well as we expected. While it's known for being a strong image classification model with pre-trained ImageNet weights, it struggled with our dataset. This could be because of 2 main reasons. First, the VGG model likely requires more data to accurately train it, more than the Inception 3 model. Second, the way VGG handles image features may not have been suited for the images in our dataset.

To fit this model, a validation split of 0.3 was used, and the epochs were 20, with a batch size of 64, and custom callbacks similar to before were used. This model likely has the best balance between batch size and epochs but performed very poorly.

This model, on our testing set, recorded a test loss of 3.4104 and a test accuracy of 0.2218

It only got 2 out of 9 correct, which is better than the LeNet CNN but significantly worse than the Inception V3.

References

Simonyan, K., & Zisserman, A. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv:1409.1556. https://arxiv.org/pdf/1409.1556

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2015). Rethinking the Inception Architecture for Computer Vision. arXiv:1512.00567.https://arxiv.org/abs/1512.00567

Asish Bera, Zachary Wharton, Yonghuai Liu, Nik Bessis, Ardhendu Behera (2022). SR-GNN: Spatial Relation-aware Graph Neural Network for Fine-Grained Image Categorization https://arxiv.org/pdf/2209.02109