Predict Breed from Dog Images

Ethan Cloin, John Butoto, Max Pilot
Dept. of Computing and Information Science University of North Florida, Jacksonville, FL, USA

Abstract

Our application offers a prediction of dog breed on an image uploaded by the user. The application was developed in two main stages. First, we collected data from a public API, PetFinder (https://www.petfinder.com/developers/v2/docs/). We collected data for dogs in the Jacksonville area, including images, names, and breeds. This data was stored in a series of JSON files, and later parsed into a CSV using Python.

We used this dataset and the Stanford Breed dataset to fine-tune an Xceptionet Convolutional Neural Network. This fine-tuned model generates a class prediction for the provided dog image.

We exported the best performing fine-tuned model to a file and connected it to a Flask web application. The application includes a form to submit an image and some relevant information about the pet.

Code — https://github.com/EthanCloin/adoption-blurb-generator

Datasets — https://github.com/EthanCloin/adoption-blurb-generator, [Add Stanford Breed]

Introduction

In recent years, the use of machine learning and computer vision techniques for image classification has seen rapid growth, finding applications in areas ranging from healthcare to pet adoption services. This project focuses on building an application capable of predicting a dog's breed from an uploaded image. Our goal was to create an accessible and efficient tool that could assist users, including pet owners and adoption centers, in identifying dog breeds quickly and accurately. We aimed to have a high-performing machine learning model, alongside a functional web interface.

The application was developed through a two-stage process. Initially, we curated a dataset by gathering real-world dog images and breed information from the PetFinder API. The API allowed us to send a request for information on pets based on a number of parameters. The data we collected was primarily on dogs in the Jacksonville area. This data was supplemented with the well-established Stanford Dog Dataset to ensure a robust training set. We then fine-tuned a pre-trained deep learning model, XceptionNet, to perform the breed classification task. The resulting model was integrated into a Flask web application that allows users to submit an image and receive a breed prediction.

Methodologies/Algorithms/Approaches

Proposed Approach

Dog breed predictor model
We implemented a convolutional neural network (CNN) pipeline for dog breed classification using the Stanford Dogs Dataset (Khosla, 2011). Using the powerful feature-extraction capabilities of the Xception architecture (Chollet, 2017), the model was trained end-to-end (with a frozen base) on 120 dog breeds, achieving high accuracy through careful data preparation, training strategies, and evaluation.

Dataset Acquisition & Preparation
We acquired the Stanford Dogs Dataset using the Kagglehub API, ensuring a reproducible and up-to-date data pull. Each class folder originally adhered to the naming convention (e.g., “n02085620-Chihuahua”); thus, we applied a simple regular expression to strip the numeric prefix and yield human-readable breed labels (e.g., “Chihuahua”). To support model evaluation, we partitioned the renamed images into training, validation, and test subsets on a per-breed basis, using a 70 %/15 %/15 % split with scikit-learn’s train_test_split and a fixed random seed (42) to ensure deterministic results.

All images were uniformly rescaled by a factor of 1/255 before ingestion into the network. Three Keras ImageDataGenerator pipelines were instantiated, train_gen (with shuffling enabled), val_gen (shuffle disabled), and test_gen (shuffle disabled) each targeting 299×299 pixels (the Xception default) and operating with a batch size of 32. This setup facilitated efficient, on-the-fly data loading and ensured consistent preprocessing across training and evaluation phases.

Model Architecture
The core of our model employs the Xception convolutional

neural network as a frozen feature extractor. Input images of size 299 × 299 × 3 are fed directly into the Xception base pretrained on ImageNet (Deng, 2009) with its classification head removed, ensuring that the rich, hierarchical feature representations learned on large-scale data are used without further modification. By freezing all layers of the base model, we dramatically reduce the number of trainable parameters, which both accelerates convergence and mitigates overfitting on the relatively small Stanford Dogs dataset.

On top of the frozen backbone, we append a lightweight classification head tailored for breed classification. Global average pooling condenses the spatial feature maps into a fixed-length feature vector, preserving channel-wise activations while reducing parameter count. A dropout layer with a rate of 0.7 introduces stochastic regularization, preventing co-adaptation of features and further combating overfitting. Finally, a dense layer with softmax activation produces per-class probability estimates across all dog breeds, enabling end-to-end training of the classification head while retaining the expressive power of the pretrained convolutional base.

Training Setup

We trained the network for 10 epochs using the Adam optimizer with an initial learning rate of 0.001, minimizing categorical cross-entropy and tracking accuracy as our primary metric. To ensure convergence, we employed three callbacks: a ModelCheckpoint that saves the best weights based on validation accuracy, an EarlyStopping criterion with a patience of five epochs on validation loss (restoring the best weights upon termination), and a ReduceLROnPlateau scheduler that reduces the learning rate by half if validation loss fails to improve over three consecutive epochs, with a floor of 1×10^{-6}.

Evaluation Metrics

We assess model performance using standard classification metrics:

Overall Accuracy

This tells us "Across all test images, what fraction did the model classify correctly?". It's the total count of correct predictions (sum of all TP's) divided by the total number of samples.

$$TP + TN + FP + FN$$

Where:

True Positives (TP) is the count of positive instances correctly identified

False Positives (FP) is the count of negative instances incorrectly labelled positive

True Negatives (TN) is the count of negative instances correctly identified

False Negatives (FN) is the count of positive instances missed by the model

Precision n

This tells us "Of everything the model labelled as breed n, what fraction was actually breed n?". High precision means few false alarms: when the model calls a dog breed n, it's usually right.

$$TP_i + FP_i$$

Recall n

This tells us "Of all the real instances of breed n, what fraction did the model successfully detect as n?". High recall means the model misses very few true n's.

$$TP_i + FN_i$$

F_1-score n

This is the harmonic mean of precision and recall for breed n. It balances the trade-off: a high F_1 only occurs if both precision and recall are high.

$$\text{Precision}_i + \text{Recall}_i$$

Application Design

To create an interface for the project, we leveraged the Flask micro-framework for web applications in Python. Our application exposes endpoints which render HTML templates to support interactivity with the application.

Our primary endpoint is an HTML form including a file input element which accepts a JPEG image upload. After form submission, the application runs the image through our trained model. The application is lightweight, including a single 'main' Blueprint which exposes two endpoints, one for the form, and one for the result.

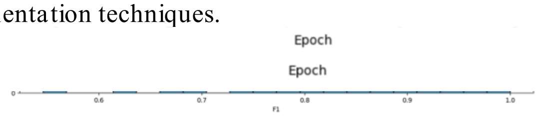
The result view includes the provided image and the breed prediction with confidence percentage. We utilize a lightweight JavaScript library, HTMX, to provide some simple interactivity on the client side. Part of the result template is an additional form which asks for user feedback on the result.

Results

The final classifier exhibits strong overall performance while retaining good per-breed consistency. After training for 10 epochs on our curated dog-breed dataset, the model converged to a training accuracy of 93.20 % and achieved 90.20 % validation accuracy on the held-out set. This 3-point gap indicates only slight overfitting and suggests that the learned features generalize well to unseen images.

A closer inspection of the per-breed metrics Fig. 1 shows that most classes had a high Precision, Recall, and F_1-scores. In the Precision bar chat Fig. 1, over half of the breeds cluster above 0.95, with a long right-hand tail reaching 1.00 indicating zero false positives for many classes. Similarly, Recall values are predominantly above 0.90, and the F_1-distribution mirrors this trend, with the vast majority of breeds scoring above 0.92. Only a handful of rarer breeds fall into the 0.55–0.70 range on any one metric, highlighting those specific classes as candidates for targeted data augmentation or architectural refinement.

Taken together, these results demonstrate that our model not only learns discriminative representations for the majority of dog breeds but also maintains balanced Precision and Recall at scale. The small disparity between training and validation accuracy (≈ 3 %) confirms good generalization, while the tight clustering of F_1-scores shows consistent performance across all breed categories. In future work, we will further improve the low-performing tail by adding more examples of those underrepresented classes and applying data augmentation techniques.



If the user indicates that the prediction was incorrect, we allow them to provide the correct breed in the feedback form. The responses are stored in a local SQLite database file. Since our form includes the classes we trained the model on, this allows potential to improve the performance by determining patterns in incorrect predictions and building a targeted dataset to retrain.

Results

Data Collection:
After completing our 1000 requests to the PetFinder API, we collected 100 JSON files each containing 100 animal objects. After parsing these scripts, we had 7,366 unique IDs with pet images. We also wrote the attributes and IDs to a combined CSV file for easy portability.

Training the Model:

We found success with this structure, storing globally relevant information, like the path to the image 'uploads' folder, in the config.py file. This made the information readily available both the main.py route and the breed_classifier.py module.

The speed of our model response is faster than expected. Using a local instance of a finetuned model and referencing the same image file instead of sending an HTTP request with image data contribures to the snappy response.

Conclusions

References

Reference citations in the text should appear in author-year format, for example (Smith 1975). References of the same year by the same author(s) should be distinguished by small letters following the year, for example (Smith 1977c) and ordered alphabetically by title. Use a narrative citation form when referring to a paper in a narrative context. For example, say "In his paper, Michael Youngblood (2017) refers to ...." instead of (Youngblood 2017) refers to...."

All entries in the reference list must be cited in the text. In-text citations of four or more authors should be shortened to "first author et al." For example, the university technical report reference in the sample that follows would be cited as (Vattam et. al. 2013) in the text because it contains four authors.

Generally, references include the name of the author (surname first, followed by initials only for given names) and the date, followed by a period, then the title, presented in mixed case. For multiple authors, separate two names with a comma, and three or more authors with a semicolon. The place of publication (which is required for all book and proceedings publications) is followed by a colon, with the name of the publisher following. For journal articles and serial publications, provide the volume and issue numbers as well as the page numbers. DOIs are strongly for serial publications if they have been assigned. For conference papers, and book chapters, give inclusive page numbers. Provide the DOI if it is available. Do not use shorthand abbreviations (such as AAAI-19) — spell out the full title of the publication.

If you are citing an ephemeral or general page of a website (such as, but not limited to, the landing page of a company or product), please do not include the citation in the reference list. Instead, incorporate the URL into a footnote. All references must contain author, title, and date information.

To meet CrossRef requirements, all cited journal articles within a reference list should include a DOI if one has been assigned.

References should be listed alphabetically (by surname of the primary author or main entry) at the end of the article. Multiple references by the same author(s) should be listed in ascending chronological order with the earliest reference first (for example, Matthews 1979 precedes Matthews 1986). Information for each reference should be in the sequence illustrated by the following examples.

Dissertation or Thesis
(Note: Include department and university):

Clancey, W. J. 1979b. Transfer of Rule-Based Expertise through a Tutorial Dialogue. PhD dissertation, Department of Computer Science, Stanford University, Stanford, CA.

Forthcoming Book

Clancey, W. J. Forthcoming. *The Engineering of Qualitative* Models. Redwood City, CA: Addison-Wesley Publishing Company.

Preprint Server

Agrawal, A.; Batra, D.; and Parikh, D. 2016. Analyzing the Behavior of Visual Question Answering Models. arXiv preprint. arXiv:1606.07356v2 [cs.CL]. Ithaca, NY: Cornell University Library.

Published Book

Petroski, H. 1985. *To Engineer Is Human: The Role of Failure in Successful Design.* New York: St. Martin's Press.

Chapter in Published Book

Brown, J. S. 1977. Artificial Intelligence and Learning Strategies. In *Learning Strategies,* edited by J. O'Neil, 345–78. New York: Academic Press.

Forthcoming Journal Article

O'Connor, J. L. Forthcoming. Artificial Intelligence and Commonsense Reasoning. *AI Magazine* 44(3).

Published Journal or Magazine Article

Cox, M. T. 2007. Perpetual Self-Aware Cognitive Agents. *AI Magazine* 28(1): 32–45. doi.org/10.1609/aimag.v28i1.2027.

Paper Presented at Meeting

(Note: Use this format only if no published proceedings appeared):

Schoenfeld, A. H. 1981. Episodes and Executive Decisions in Mathematical Problem Solving. Paper presented at the 1981 AERA Annual Meeting. Boston, MA, September 24–30.

Zhou, S.; Suhr, A.; and Artzi, Y. 2017. Visual Reasoning with Natural Language. Paper presented at the AAAI 2017 Fall Symposium on Natural Communication for Human-Robot Collaboration. Arlington, VA, November 9–11.

Paper Presented at Meeting and Published in Proceedings

Lester, J.; Converse, S.; Kahler, S.; Barlow, T.; Stone, B.; and Bhogal, R. 1997. The Persona Effect: Affective Impact of Animated Pedagogical Agents. In Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems.New York: Association for Computing Machinery. doi.org/10.1145/258549.258797.

Company Technical Report

Carbonell, J. R. 1970. Mixed-Initiative Man-Computer Instructional Dialogues, Technical Report QW-19871. Marina del Rey, CA: USC/Information Sciences Institute.

Scholarly Society Technical Report

Lin, F. 2007. Finitely-Verifiable Classes of Sentences. In *Logical Formalizations of Commonsense Reasoning: Papers from the 2007 AAAI Spring Symposium*. Technical Report SS-07-05. Palo Alto, CA: AAAI Press.

University Technical Report

Vattam, S.; Klenk, M.; Molineaux, M.; and Aha, D. W. 2013. Breadth of Approaches to Goal Reasoning: A Research Survey. In *Goal Reasoning: Papers from the ACS Workshop,* edited by D. W. Aha, M. T. Cox, and H. Muñoz-Avila. Technical Report CS-TR-5029. College Park, MD: University of Maryland, Department of Computer Science.

ArXiv Paper

Bouville, M. 2008. Crime and punishment in scientific research. arXiv:0803.4058.

Website or online resource

NASA. 2015. Pluto: The 'Other' Red Planet. https://www.nasa.gov/nh/pluto-the-other-red-planet. Accessed: 2018-12-06.

You are required to use the above AAAI reference format in your paper. If you fail to do so, your paper will be returned to you for reformatting (and a resubmission fee will apply). In addition, you must apply the buildt-in References style (which will automatically format your references in 9 point Time Roman with 10 point line spacing, and 3 additional points of space between each entry.

For the most up to date version and complete version of the AAAI reference style, please consult the *AI Magazine* Author Guidelines at

aaai.org/ojs/index.php/aimagazine/about/submissions#authorGuidelines.