Automated Dog Breed Identification from User-Uploaded 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. 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
Stanford Dogs Dataset — https://www.kaggle.com/datasets/jessicali9530/stanford-dogs-dataset
PetFinder— https://www.petfinder.com/developers/v2/docs/

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 created 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

In this section, we first describe our convolutional neural network–based dog breed classifier, which leverages a frozen XceptionNet backbone and a lightweight classification head to assign fine-grained breed labels to cropped canine images. We then turn to our dog detector, built on the YOLOv11 architecture, that locates and crops dog instances in images with high precision and recall. Finally, we present the end-to-end application that unifies these two components: upon uploading, the YOLOv11 module identifies if an image is a dog, and the breed classifier immediately predicts the specific breed, all within a responsive web interface.

Dog breed classifier model

We implemented a convolutional neural network (CNN) pipeline for dog breed classification using the Stanford Dogs Dataset (Khosla et al. 2011). Using the powerful feature-extraction capabilities of the XceptionNet architecture (Chollet and François 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 generator (with shuffling enabled), validation generator (shuffle disabled), and a test generator (shuffle disabled) each targeting 299×299 pixels (the XceptionNet 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 XceptionNet convolutional neural network as a frozen feature extractor. Input images of size 299 × 299 × 3 are fed directly into the XceptionNet base pretrained on ImageNet (Deng et al. 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 accelerate convergence and mitigate overfitting on the relatively small Stanford Dogs dataset.

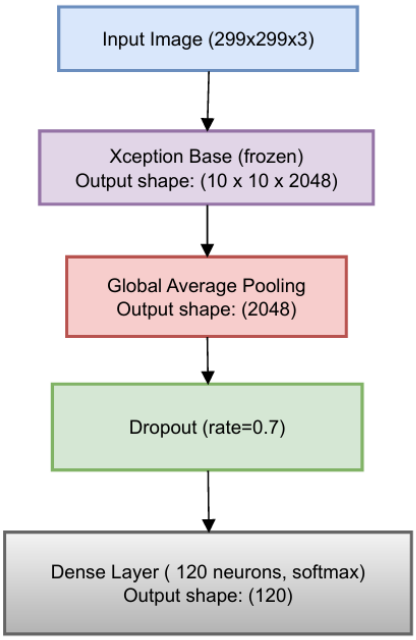

Figure 1: Proposed CNN architecture for dog-breed classification, comprising a frozen XceptionNet backbone (output 10 × 10 × 2048), global average pooling (2048), dropout (rate = 0.7), and a final dense-SoftMax layer (120 classes).

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.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \quad (1)$$

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.

$$\text{Precision}_i = \frac{TP_i}{TP_i + FP_i} \quad (2)$$

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.

$$\text{Recall}_i = \frac{TP_i}{TP_i + FN_i} \tag{3}$$

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.

$$F_{1,i} = 2 \times \frac{\text{Precision}_i \times \text{Recall}_i}{\text{Precision}_i + \text{Recall}_i} \tag{4}$$

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 Fig. 2. This 3-point gap suggests that the learned features generalize well to unseen images. The training loss drops sharply during the initial epochs and continues to decline, while the validation loss remains relatively flat with a slight downward trend, indicating effective learning with maintained generalization Fig. 3.

A closer inspection of the per-breed metrics Fig. 4, Fig. 5, Fig. 6 shows that most classes had a high Precision, Recall, and F_1-scores. In the Precision bar chat Fig. 4, over half of the breeds cluster above 0.95, with a long righthand tail reaching 1.00 indicating zero false positives for many classes. Similarly, Recall values Fig. 5 are predominantly above 0.90, and the F_1-distribution Fig. 6 mirrors this trend, with most 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.

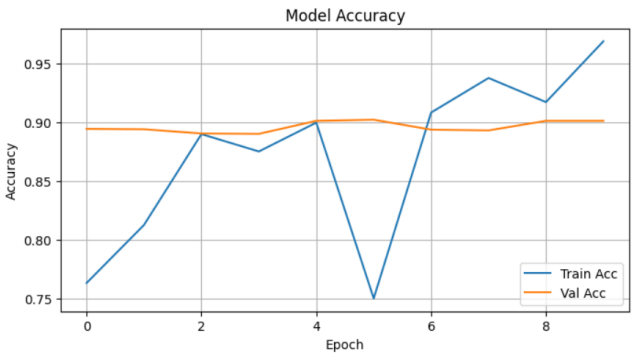

Figure 2: Epoch-wise curves of training (blue) and validation (orange) accuracy, illustrating convergence and generalization trends over 10 epochs.

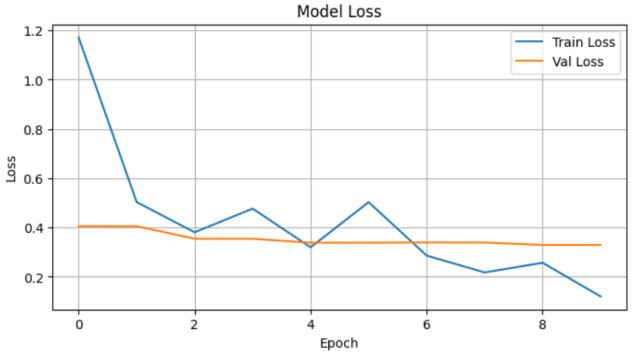

Figure 3: Epoch-wise curves of training (blue) and validation (orange) loss, demonstrating effective reduction in training loss alongside stable validation performance.

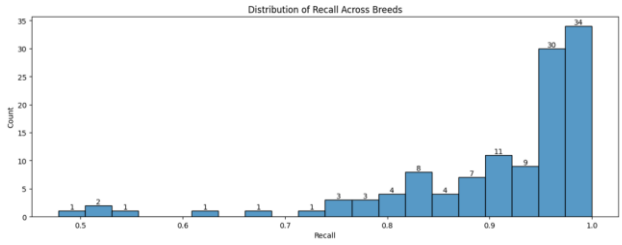

Figure 4: Bar chart showing per-class precision scores across 120 dog breeds, highlighting the distribution and concentration of high-precision classifications.

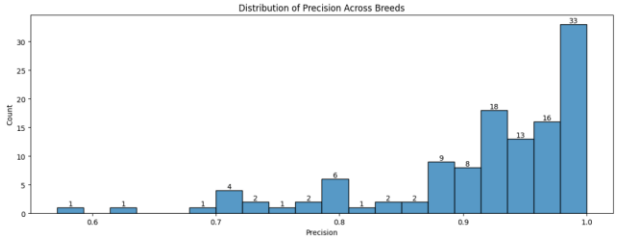

Figure 5: Bar chart showing per-class recall scores across 120 dog breeds, illustrating the range of completeness in breed detection.

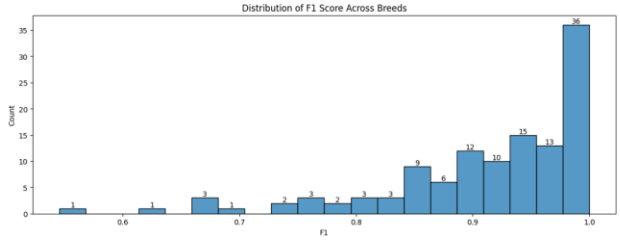

Figure 6: Bar chart showing per-class F_1 scores across 120 dog breeds, summarizing the balance between precision and recall for overall classification performance.

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.

Dog detector

In addition to the breed classifier model, we also developed a second, more simplistic model that would detect if an image contains a dog, how accurately it predicted it. The model was trained on the images provided by the Stanford Dogs dataset. We created a main YAML file that points to the image folder used for training and validation, followed by the number of classes that it will use. For this project, only one class was used, dog, due to the model only detecting if the image contains a dog or not.

For each image, we also had to create text files containing labels for the images. These labels included:
Class ID: 0 for "dog"
y-center: The horizontal center of the bounding boxes
x-center: The vertical center of the bounding boxes
Width: The normalized width of the images
Height: The normalized height of the images
If the result returns true, the model gives a confidence value score, saying that it does contain a dog. Otherwise, it will return a confidence score of zero if the result returns false, and there is no dog in the image. The result then gets console logged into our main application file.

Full web application

The trained model was successfully deployed as part of a Flask application. Flask is a 'micro-framework' designed to help developers quickly stand-up APIs with minimal boilerplate. The simplest Flask application instantiates a Flask class instance and uses the decorator pattern to transform a Python method into an HTTP endpoint. Our application instantiates the Flask application upon startup and adds a Blueprint to that app for our main controller. The Blueprint behaves very similarly to the app instance, allowing us to use a decorator to assign routes. The main benefit of using

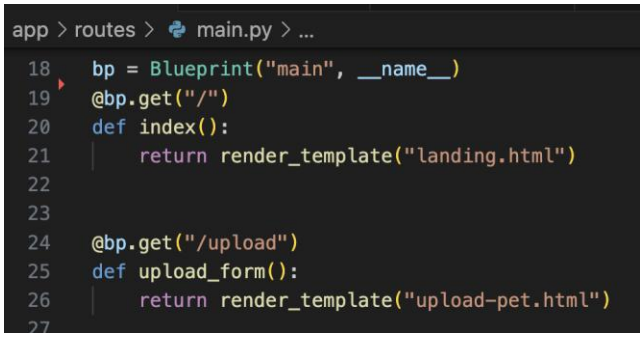

Figure 7: Flask route definitions in main.py, illustrating the use of a Blueprint to register the root ("/") and upload ("/upload") endpoints.

this blueprint is keeping routes which differ in purpose compartmentalized to separate, logical locations in the application structure.
As seen above, our application exposes an "/upload" endpoint, returning an HTML template which included a form with an input field accepting a JPEG image. Upon upload, the image is assigned a UUID and stored in the app instance's static folder. The model file references the file directly from that location.

After the Dog Breed Classifier model performs a prediction on the supplied image, the application creates a new record and inserts it into the SQLite database. SQLite is another lightweight tool, allowing us to use SQL syntax to persist data into a file instead of writing to a database in a separate server. SQLite would struggle to support multiple clients as it does not support concurrent writing on but neatly suits our purposes for this demo application.

The results from our prediction model are formatted into a string response which is included as context for the "/view-result" template. The result view displays the uploaded image, the breed prediction and confidence, and also includes a feedback form. The purpose of the feedback form is to allow users to correct inaccurate classifications and ultimately support crowd-sourced improvements on our model. The feedback form accepts input for whether the prediction was correct and a text input for the correct, with options powered by the same class list our model trained on.

This feedback form makes use of another compact and powerful piece of web technology, HTMX. HTMX is a JavaScript library which extends the capabilities of HTML. After installing the library, there are new attributes available in

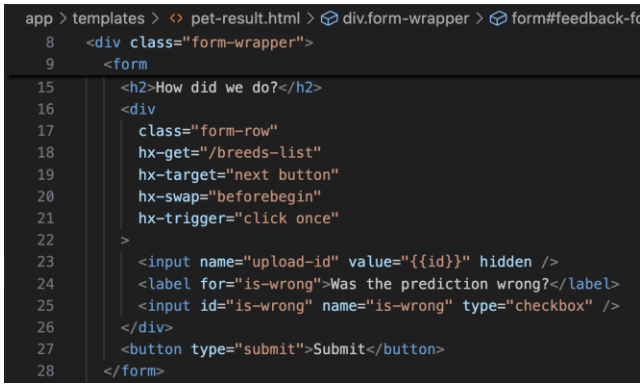

Figure 8: Snippet of the HTMX-enhanced feedback form in pet-result.html, showing dynamic breed–list loading upon checkbox interaction.

HTML which enhance the hypermedia to more flexibly interact directly with our Flask server. In our case, we wrap the form checkbox input in a div element, which includes hx-* attributes provided by HTMX. These attributes roughly translate to “the first time this element is clicked, GET the HTML from the ‘/breeds-list’ and insert it before the next button element”. The HTML stored at that endpoint includes another input element, along with a datalist element with all the human-readable dog breeds upon which the model was trained. The full application code is available in the zip and on GitHub.

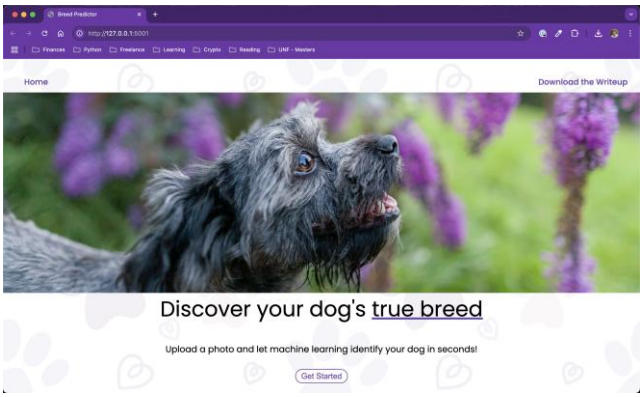

Figure 9: Landing page of the Breed Predictor application, featuring a prominent call-to-action for uploading a dog image.

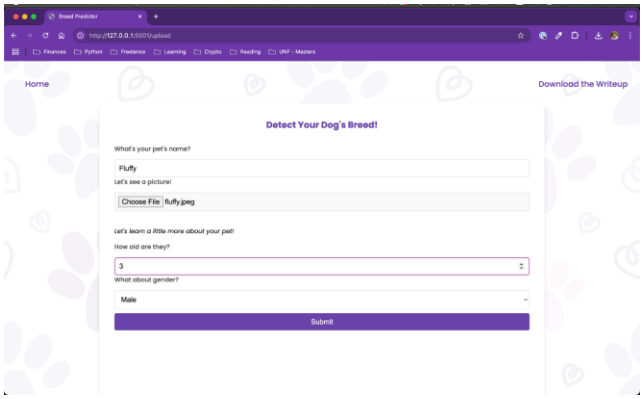

Figure 10: Image upload form, which collects the pet’s name, age, gender, and photo before submission to the server.

In Figures 9–11, we illustrate the end-user workflow of our web application. The landing page Fig. 9 invites users to discover their dog’s breed via a simple “Get Started” button. Upon initiation, the upload form Fig. 10 asks for basic pet metadata; name, age, gender and a JPEG image, which the server assigns a UUID and stores for model inference. Finally, the result view Fig. 11 presents the YOLOv11 dog detection with a confidence score (e.g., 91.25% certainty) of how strongly the model thinks the image is a dog alongside the breed classifier’s prediction and corresponding confidence scores (e.g., 98.85 % for “Samoyed”). It also includes an inline feedback form that dynamically loads the full breed list to capture user-provided corrections. This seamless integration of detection, classification, and crowd-sourced feedback supports continuous model improvement in a user-friendly interface.

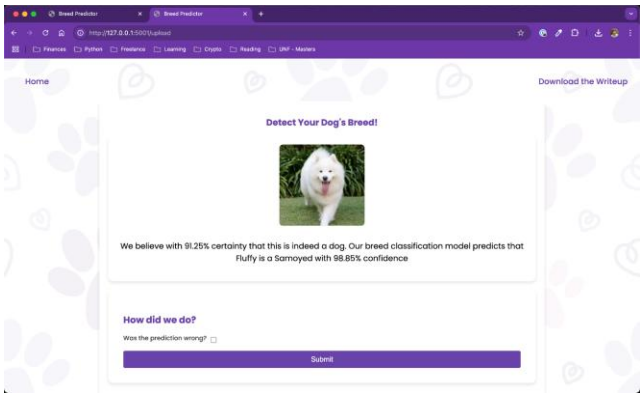

Figure 11: Prediction result view, displaying the detected dog crop, YOLOv11 detection confidence, breed classification with SoftMax confidence, and an HTMX-powered feedback form for user correction.

Conclusion

This project presents a comprehensive and effective approach to automated dog breed classification. By leveraging a frozen XceptionNet backbone with a custom classification head, the model achieved high predictive performance, attaining 93.20% training accuracy and 90.20% validation accuracy.

Performance analysis across breed classes indicated strong generalization, with precision, recall, and F_1-scores exceeding 0.90 for the majority of breeds. A small subset of underrepresented classes exhibited lower performance, suggesting potential avenues for future work in data augmentation and class balancing.

The system was deployed as an end-to-end application using a Python-based technology stack. The backend was implemented with Flask and SQLite, ensuring lightweight data persistence and seamless model integration. The frontend utilized HTMX to facilitate responsive image submissions, allowing real-time inference and display of results. Overall, the project demonstrates the viability of combining modern deep learning architectures with web technologies to deliver accessible and accurate breed identification tools for public use.

References

Khosla, Aditya, Nityananda Jayadevaprakash, Bangpeng Yao, and Fei-Fei Li. "Novel dataset for fine-grained image categorization: Stanford dogs." In *Proc. CVPR workshop on fine-grained visual categorization (FGVC)*, vol. 2, no. 1. 2011.

Chollet, François. "Xception: Deep learning with depthwise separable convolutions." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1251-1258. 2017.

Deng, Jia, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. "Imagenet: A large-scale hierarchical image database." In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248-255. Ieee, 2009.

"htmx - high power tools for html," htmx.org. https://htmx.org/

Flask, "Welcome to Flask — Flask Documentation (3.0.x)," Palletsprojects.com, 2010. https://flask.palletsprojects.com/en/stable/

SQLite, "SQLite Home Page," Sqlite.org, 2019. https://sqlite.org/index.html