PetFace Two-Stage ResNet Pipeline: Training, Inference, and Accuracy Evaluation Summary

1 System Overview

The PetFace system is a two-stage pipeline designed for large-scale species and attribute recognition on pet images.

Stage 1 – Species Classifier

Stage 1 is a single ResNet-18 model trained from scratch to predict one of 13 species from an input image (cat, dog, rabbit, etc.). It outputs a species label and associated probabilities and serves as the entry point for the full system. (Trainer: `train_resnet_minority_aug_dynamic.py`)

Stage 2 – Per-Species Attribute Specialists

Stage 2 consists of one ResNet-18 *per species*, each with multiple heads to predict attributes such as breed, gender, and color features (Color, Color1, Color2) for that species. Each specialist is trained with a multi-head architecture that reflects the annotation schema of that species. (Trainer: `stage2_resnet_multi.py`)

Inference and Evaluation

A unified inference engine runs Stage 1, selects the correct Stage 2 specialist based on the predicted species, and then predicts attributes. It can also evaluate the full dataset and dump per-image and summary metrics. (Script: `inference_engine.py`)

The `eval.zip` produced comes from this engine's `evaluate_full_dataset` run over the entire PetFace image tree, which contains approximately 1.01 million images. The evaluation aggregates both species-level and attribute-level performance.

2 Stage 1 Training – Species ResNet-18

Script

Stage 1 is implemented in the script `train_resnet_minority_aug_dynamic.py`. It encapsulates dataset construction, minority-aware sampling, dynamic difficulty adaptation, training, and checkpointing.

Data and Splitting

Images are assumed to live under:

`IMAGES_ROOT/<species>/<record_id>/<image>.png`.

The trainer:

- Enumerates all species directories to define a `class_to_index` mapping.
- Splits the data *per record folder* (not per image) into:

$$80\% \text{ train}, \quad 10\% \text{ validation}, \quad 10\% \text{ test},$$

  which guarantees zero record leakage across splits.

This yields a heavily imbalanced species distribution:

- Cats and dogs have hundreds of thousands of images.
- Smaller species (e.g., ferret, chinchilla, javasparrow) typically have only a few thousand images each.

Model and Augmentations

The Stage 1 model is:

- Backbone: ResNet-18 with randomly initialized weights (`weights=None`) to match the baseline.
- Final fully connected layer: replaced with a `Linear(512, num_species)` layer.
- Input: images are treated as 224×224 RGB with no explicit normalization, to stay consistent with earlier experiments.

Augmentations applied during training include:

- Random horizontal flip.
- Color jitter (brightness, contrast, saturation, hue).
- Random affine (small rotations, translations, scaling, shearing).
- Random erasing.

There are two augmentation regimes:

- *Base transform* for head (frequent) classes.
- *Tail transform* with stronger geometric and color jitter and higher random-erasing probability for minority classes.

Minority-Aware and Difficulty-Aware Sampling

The training script implements a sophisticated rebalancing strategy.

Minority Definition. Minority status is configurable.

$$\texttt{TAIL\_MODE} = \text{``}\texttt{head\_exclusion}\text{''}.$$

In this mode:

- Some "head" classes (specifically {`cat`, `dog`}) are never considered minority.
- All other classes with count $\leq$ `HEAD_EXCLUSION_MAX` are treated as minority.

Base Class Weights. Let c_i be the training count for class i, and let $\tilde{c}$ be the median count over classes with positive support. Base class weights are computed as:

$$w_i \propto \left(\frac{\tilde{c}}{c_i}\right)^{\alpha},$$

then clipped to:

$$w_i \in [1, \texttt{MAX\_UPWEIGHT\_PER\_CLASS}].$$

These weights feed into a `WeightedRandomSampler`, upsampling minority species while controlling variance.

Dynamic Difficulty Adaptation. After each epoch, the trainer:

- Computes a confusion matrix and per-class recall on a specific split (validation by default).
- Defines difficulty for each class as:

$$\text{difficulty}_i \approx 1 - \text{recall}_i.$$

- Normalizes difficulties by the median difficulty and raises to a power `PERF_ALPHA` to create a difficulty multiplier.
- Multiplies the base weights by the difficulty multiplier and clips again.

The weights of the `WeightedRandomSampler` are then updated *in-place* each epoch. As a result:

- Under-represented and low-recall species are sampled more frequently.
- Easy head classes are kept under control.

This yields an *active curriculum* that adapts sampling based on where the model currently struggles.

Training and Checkpoints

- Optimizer: AdamW with learning rate LR $= 10^{-3}$ and weight_decay $= 10^{-4}$.
- Mixed precision training: enabled on CUDA via `torch.amp.GradScaler`.
- Loss: standard cross-entropy.
- Validation: run every epoch; early stopping is based on validation accuracy with a configurable patience.

Checkpoints:

- `last.pt`: full training state including model weights, optimizer state, scaler, `class_to_index`, and epoch.
- `best.pt`: an inference-ready checkpoint containing:
  - `"model"`: state dict of the trained ResNet-18.
  - `"class_to_index"`: mapping from string species names to indices.

The `best.pt` checkpoint is exactly what the inference engine loads as Stage 1.

3 Stage 2 Training – Per-Species Multi-Head ResNet-18

Script

Stage 2 training is implemented in `stage2_resnet_multi.py`. Each run trains a specialist for a single species, chosen via the `SPECIES` constant.

Per-Species Configuration

A global `SPECIES_CONFIG` dictionary defines, for each species:

- The annotation CSV file (e.g. `cat.csv`, `dog.csv`, `javasparrow.csv`).
- The list of attribute heads to train. Examples:
  - **Cat**: `["gender", "breed", "color1", "color2"]`.
  - **Dog**: `["gender", "breed"]`.
  - **Hamster, Guinea pig, Rabbit**: `["gender", "breed", "color1", "color2"]`.
  - **Javasparrow**: `["gender", "color", "color1", "color2"]`.
- A mapping from each head to its CSV column and type:
  - `type="gender"`: mapped via the gender map `GENDER_MAP` $= \{"female" : 0, "male" : 1\}$.
  - `type="categorical"`: vocabulary built from unique non-unknown values in the corresponding column.
- Which heads have their vocab saved in the checkpoint (via `head_vocabs`) so inference can reconstruct semantic labels.

Data Processing and Split

For each species:

- The trainer reads the annotation CSV, normalizes fields, and builds:
  - Per-head vocabularies for categorical attributes.
  - Per-image targets as integer IDs, using -1 for unknown or missing labels.
- Each record folder (identified by `Name`) may contain multiple images; all images share the same targets.

- Each image is represented by a `Sample(path, record_id, targets)` object.

The split is done at the record level with attribute-aware stratification:

- The primary attribute is chosen as the first non-gender head in `primary_attr_priority` (e.g. breed or color1).
- Records are grouped by this primary attribute where possible, and the train/validation/test split is made to preserve the attribute distribution.
- As in Stage 1, this ensures zero leakage across splits.

Model Architecture

The Stage 2 model uses:

- Backbone: ResNet-18 with ImageNet pretraining if available; otherwise, `pretrained=True` fallback.
- A custom `_MultiHeadModel`, consisting of:
  - Shared backbone (`conv1` through `layer4`) plus global average pooling.
  - One fully connected head (`Linear`) per attribute, stored in an `nn.ModuleDict`.
  - A forward pass that returns a tuple of logits in a fixed head order.

As a result, each species obtains an independent ResNet-18 backbone with a small set of attribute heads configured for that species.

Training Losses and Sampling

Treatment of different heads:

- **Gender head**:
  - Loss: masked cross-entropy with inverse-frequency class weights.
  - Unknown labels (-1) are masked out; they do not contribute to loss or accuracy.
- **Non-gender heads** (breed, color, color1, color2):
  - Loss: masked focal loss plus class weighting.
  - Focal gamma:
    - `FOCAL_GAMMA_COLOR` $= 2.0$ if there are color heads.
    - `FOCAL_GAMMA_IMBAL` $= 2.5$ if only breed exists (heavily imbalanced).
  - This focuses learning on rare breeds/colors and harder examples.

Sampling strategy:

- Class weights are computed per head from training counts.
- The script selects the first non-gender head with $K > 0$ in `primary_attr_priority` and builds a `WeightedRandomSampler` over training images based on that head's class weights.
- This ensures emphasis on the most imbalanced attribute while all heads are trained jointly.

Two-Phase Training and Evaluation

Stage 2 training proceeds in two phases:

Phase 1 – Heads + layer4.

- Freeze the backbone except for `layer4`; train `layer4` and all heads.
- Optimizer: AdamW with learning rate 10^{-3}.
- Early stopping on validation loss with patience = 7.
- Checkpoints:
  - {`species`}`_last.pt`: full training state.
  - {`species`}`_best.pt`: inference-ready checkpoint storing:
    - `"model"`: state dict.
    - `"head_vocabs"`: mapping from head names to vocab lists.

Phase 2 – Fine-tune layer3.

- Unfreeze `layer3` so that `layer3`, `layer4`, and all heads are trainable.
- Reset the optimizer with a smaller learning rate 3×10^{-4}.
- Early stopping on validation loss with patience = 5.
- Whenever the validation loss improves, {`species`}`_best.pt` is updated.

Final Evaluation.

- Load {`species`}`_best.pt`.
- Evaluate on the species-specific test split and on all images (train + validation + test).
- Compute confusion matrices for each head on both:
  - Saved as `confusion_test_<head>.csv` and `confusion_all_<head>.csv`.
- Produce a human-readable `eval_summary_stage2.txt` reporting:
  - Per-head accuracy.
  - Macro precision, recall, and F1.
  - Per-class support and recall for the top-support classes.

  These per-species summaries feed into the global evaluation via the inference engine.

4 Inference Engine

Script

The inference logic is implemented in `inference_engine.py`. It defines wrappers around Stage 1 and Stage 2 models, and a high-level `TwoStageInference` class for single-image inference and full-dataset evaluation.

Stage 1 Wrapper

The `Stage1Model` class:

- Loads `runs/cnn_folder_224_raw_aug/best.pt`.
- Rebuilds a ResNet-18 with the correct number of output classes and loads its state dict.
- Uses the stored `class_to_index` mapping to translate indices back to string species names.
- Applies a simple evaluation transform: resize to 224×224, convert to tensor, cast to `float32`.

The method `predict(img, topk)`:

- Runs the model, applies softmax, and returns the top-k species with probabilities.
- Also returns the argmax species label.

Stage 2 Wrapper(s)

The `Stage2Model.load_for_species` method is designed to be robust to multiple checkpoint formats:

- **New multi-head layout**: checkpoints with keys like `feature_infos`, `head_vocabs`, `heads_meta`, or `vocabs + heads`; these use a `ChinchillaMultiHeadResNet18` backbone with a `ModuleDict` of heads.
- **Legacy layout**: checkpoints with flat `head_*` parameters in the state dict; these use `GenericMultiHeadResNe` and infer the number of classes per head from the shape of the corresponding weight matrices.

For each head, the loader builds or recovers a vocabulary in the following order:

1. Prefer checkpoint-stored vocabularies via `head_vocabs`.
2. Fall back to hard-coded breed and color vocab lists for that species (mirroring the annotation files).
3. Fall back to the default gender vocab `["female", "male"]` when appropriate.
4. As a last resort, use index labels `["0", "1", ..., "K-1"]`.

The method `predict(img, topk_per_head)`:

- Runs the model and applies softmax on each head's logits.
- For each head:
  - Returns the argmax label and its top-k probability list.
  - Skips heads with zero classes (e.g., where a color2 head has no valid labels for a species).

Additional logic:

- A generic `attr2` head can be remapped to a semantic name (e.g. breed or color) for some species.
- Predictions are filtered to the species-specific feature set defined in `SPECIES_FEATURES`, so only relevant heads are reported.

Two-Stage Inference and Dataset Evaluation

The `TwoStageInference` class:

- Caches the Stage 1 model.
- For `infer(image_path)`:
  1. Runs Stage 1 to get the predicted species and top-k probabilities.
  2. Loads the corresponding per-species Stage 2 specialist (if a checkpoint exists).
  3. Runs Stage 2 to get raw attribute predictions.
  4. Applies the `attr2` alias mapping and species-feature filtering.

The `evaluate_full_dataset` function:

- Traverses all images under `IMAGES_ROOT`.
- For each image:
  1. Derives the true species from the folder name and the record ID (`Name`) from the second path component.
  2. Runs Stage 1 and records whether the predicted species matches the true species.
  3. Only if Stage 1 is correct:
     - Loads the annotation row for that record from `<species>.csv`.
     - Loads (and caches) the Stage 2 specialist for that species.
     - Runs Stage 2.
     - For each semantic feature in {"breed","color","color1","color2","gender"}:
       * Maps it to the correct CSV column for this species.
       * Skips the example if the ground-truth label is unknown (blank, "unknown", "NA", etc.).
       * Otherwise, compares the predicted label with ground truth and accumulates totals.

Outputs:

- `petface_eval_per_image.csv`: per-image records with Stage 1 predictions and Stage 2 predicted/ground-truth attributes.
- `petface_eval_summary.json`: aggregate statistics for:
  - Stage 1 overall and per species.
  - Stage 2 per feature (global) and per species (per-feature).

5 Accuracy Results

All numbers in this section come from `petface_eval_summary.json` produced by the full-dataset evaluation.

Stage 1 – Species Classification

- Total images evaluated: 1,012,933.
- Correct: 1,011,103.
- Overall accuracy: **99.82%**.

Per-species accuracy (true species from folder name):

Species	Accuracy (%)
cat	99.96
dog	99.88
hedgehog	99.48
parakeet	99.23
chimp	99.15
chinchilla	99.00
rabbit	98.99
ferret	98.96
degus	98.62
hamster	98.08
pig	97.27
guineapig	97.59
javasparrow	97.14

Even the weakest species are around 97% accuracy, while cats and dogs are essentially saturated. Given the heavy imbalance (cats and dogs dominate), this is about as strong a species classifier as one can realistically expect without significantly altering the architecture.

Stage 2 – Attribute Accuracy (Conditional on Stage 1 Correct)

These statistics are *conditional* on:

- Stage 1 predicting the correct species.
- The relevant annotation for that feature being present and not an "unknown" token.

Global per-feature performance:

Feature	# Evaluated	# Correct	Accuracy (%)
breed	840,331	713,638	84.92
color	4,488	3,979	88.66
color1	717,202	596,300	83.14
color2	486,768	375,865	77.22
gender	940,701	614,625	65.34

In summary:

- Breed and color predictions are strong ($\approx$ 83%–89%).
- Color2 is harder (secondary or compound colors) at about 77%.
- Gender is clearly the hardest attribute at around 65%.

Per-species extremes:

Breed

- Best: hamster – 97.41% (2,318 examples).
- Weakest: chimp – 55.57% (1,168 examples).

Color1

- Best: hedgehog – 91.72% (2,271 examples).
- Weakest: degus – 62.72% (4,249 examples).

Color2

- Best: chinchilla – 95.63% (572 examples).
- Weakest: degus – 67.94% (705 examples).

Gender

- Best: javasparrow – 84.16% (202 examples).
- Weakest: degus – 50.36% (3,797 examples).

For the dominant species:

Cats

- Breed: 88.36% on 519,144 evaluated samples.
- Color1: 83.93%.
- Color2: 77.00%.
- Gender: 67.35%.

Dogs

- Breed: 76.83%.
- Gender: 60.99%.

Overall, Stage 2 specialists achieve very solid performance on breed and color, with gender emerging as the main soft spot.

6 Interpretation and Next Steps

High-Level Interpretation

From a practical standpoint:

- **Stage 1 is essentially solved.** With 99.8% species accuracy on more than one million images and robust handling of minority classes, there is very little to gain from further architectural complexity unless the dataset or requirements change.
- **Stage 2 is strong overall, but gender is the bottleneck.**
  - Breed and color performance in the mid-to-high 80% range is clearly usable.
  - Gender at around 65% means roughly one out of three labels is wrong, which may or may not be acceptable depending on the application.