SmartStock AI

An intelligent inventory management system leveraging FastAPI, Next.js, and EfficientNetB0.

FastAPI

Next.js 14

TensorFlow

MongoDB

Backend Architecture

Framework

FastAPI (Python)

- High Performance (ASGI)
- Async ML Operations

BackgroundTasks

Offloads heavy training logic without blocking the main thread.

AI Integration

Inference Engine → loaded → **PetClassifier** Singleton → `/ml/classify`

Loads model once. Handles real-time prediction from raw bytes.

Model Surgery `routes_train.py`

Orchestrates incremental learning.

Existing APIs

Auth:
`POST /auth/login (JWT)`

Products:
`GET /products/shop/published`
`POST /products/ (Image Upload)`

Train:
`POST /train/add-breed (Zip Upload)`
`GET /train/status (Polling)`

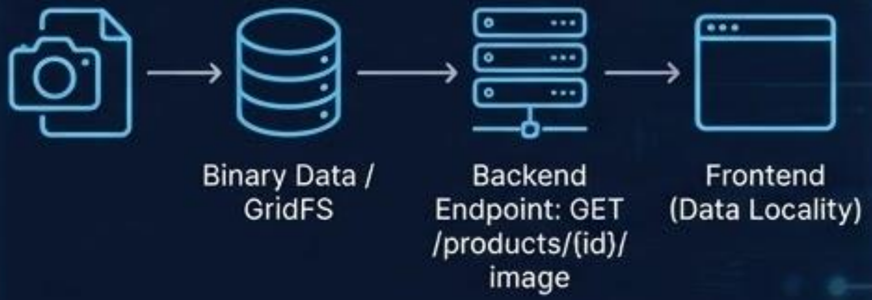
ML Stub:
`POST /ml/classify`

Database Architecture

Strategy: MongoDB (NoSQL)

Justification: Pet products vary wildly (e.g., 'water_type' for fish vs. 'breed_size' for dogs). MongoDB's schema-less nature handles this polymorphism efficiently without sparse SQL tables.

Image Storage Process



Images are stored as **binary data** directly in the database or via GridFS. The backend provides a dedicated endpoint to serve raw bytes to the frontend, ensuring data locality.

Product Schema (JSON Preview)

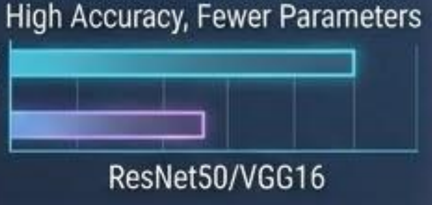
```
{
  "_id": "ObjectId(...)",
  "name": "Premium Dog Food",
  "type": "Dog",
  "breed": "Golden Retriever",
  "price_predicted": 45.99,
  "price_modified": null,
  "attributes": { // Flexible schema
    "weight_kg": 12.5,
    "vaccinated": true
  }
}
```

Classification Model: EfficientNetB0

Why EfficientNetB0?

Efficiency

Achieves high accuracy with significantly fewer parameters than ResNet50 or VGG16.

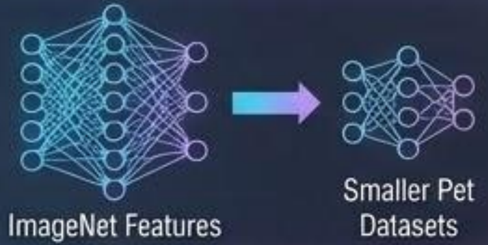


Speed

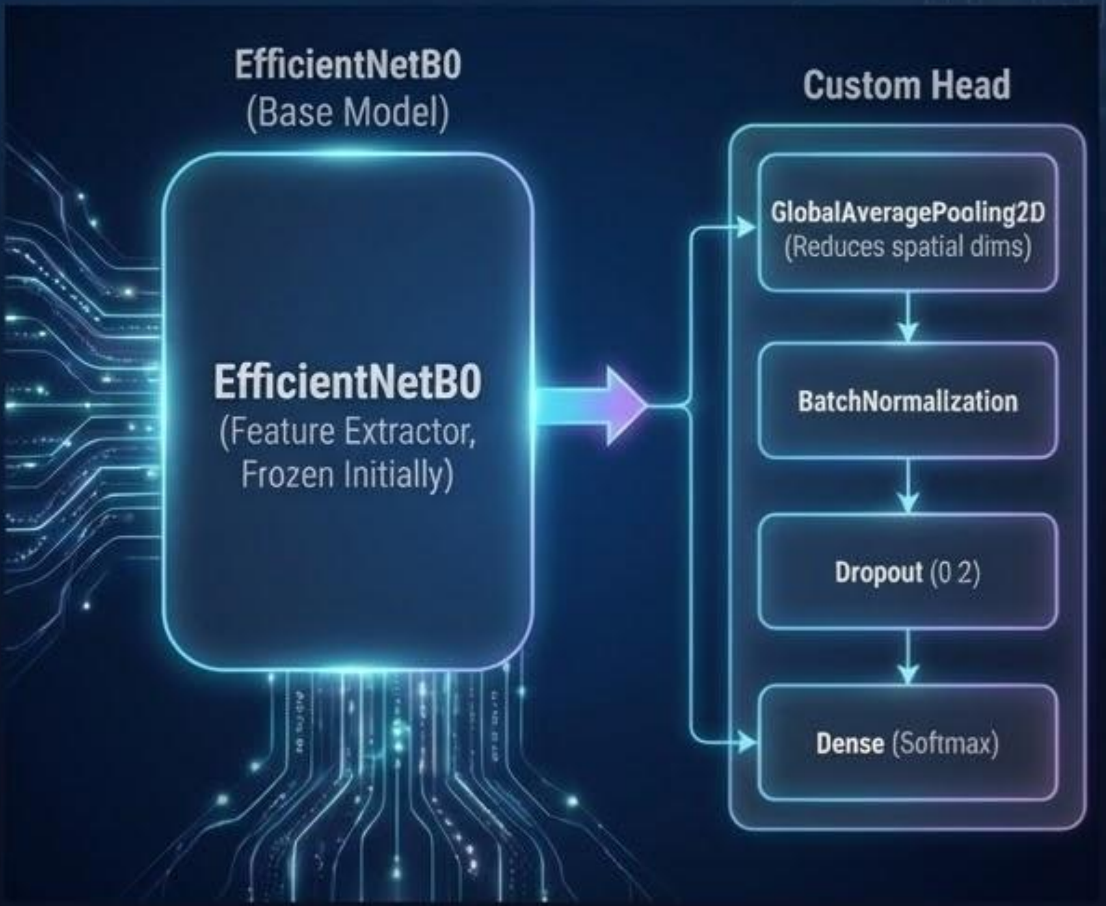
Faster inference time is critical for the real-time web application experience.

Transfer Learning

Leverages features learned from ImageNet to train effectively on smaller pet datasets.



Architecture



Training Data Strategy

Data Directory Structure

Images are organized hierarchically: Category > Breed.
The metadata used for the price regression model is stored separately in CSV format.

Augmentation

Techniques: Rotation, Zoom, Horizontal Flip.

Increases dataset diversity and prevents overfitting by ensuring the model learns features (fur, ears) rather than specific lighting or poses.

Hyperparameters

Batch Size: 32

Phase 1: 50 Epochs (Head Only) 50

Phase 2: 10 Epochs (Fine-Tune) 10

Dynamic Training Strategies

Model Surgery

Goal: Add a *new* breed (e.g., Poodle) without retraining from scratch.

1. Prune & Expand: Remove output layer (N neurons). Create new layer with N+1 neurons.

2. Weight Transfer: Copy old weights. Initialize only the new neuron.

Fine-Tuning

Goal: Improve accuracy for an *existing* breed using new photos.

Low Learning Rate: Uses 1e-5 to gently adjust weights without destroying pre-learned features.

Rehearsal Strategy: Mixes in data from *other* breeds to prevent "Catastrophic Forgetting" of old knowledge.

Frontend Engineering

Framework & Stack

Next.js 14

Utilizes App Router for Server-Side Rendering (SSR) and superior SEO.

Styling

Tailwind CSS combined with Shadcn/UI for rapid, accessible component development.

State

React Context API (AuthContext) and custom Hooks (use-toast, use-mobile).

Key UI Features

Smart Upload

Drag & Drop interface with real-time inference feedback

Real-time Console

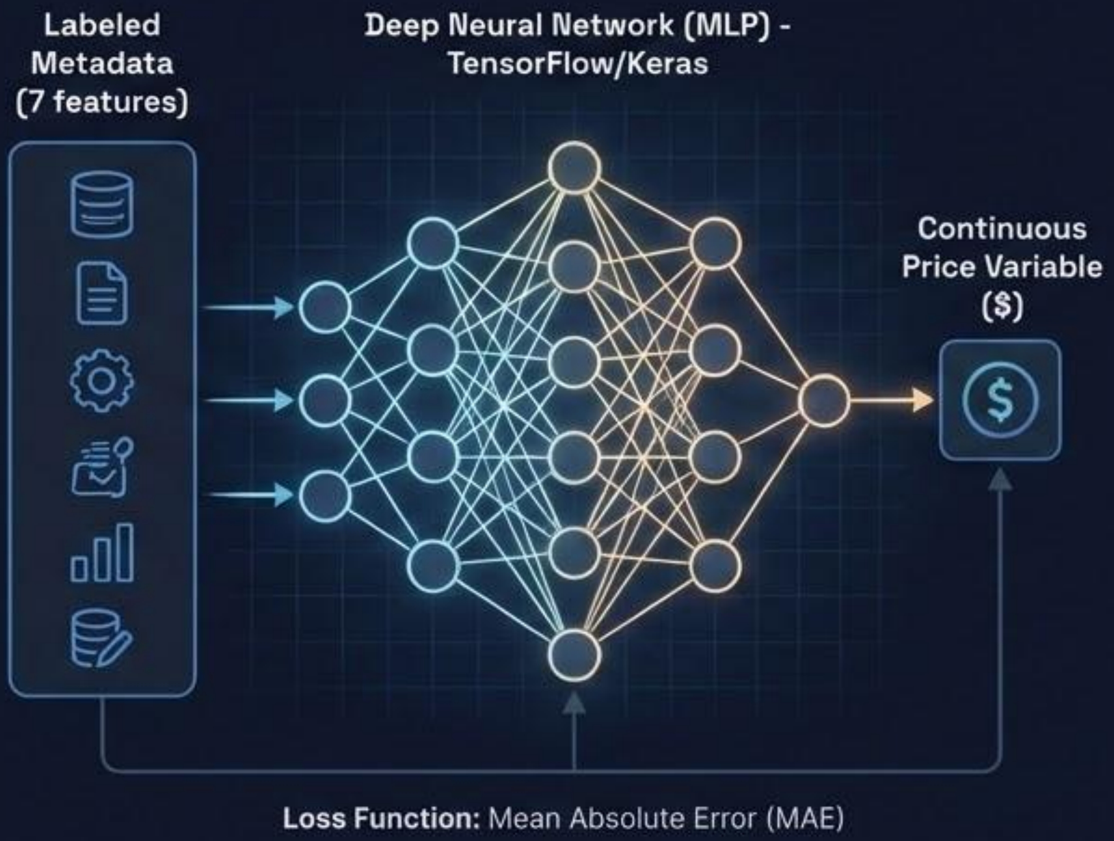
Live streaming of training logs to the browser via polling status endpoints.

Admin Dashboard

Visual analytics tracking MAE accuracy and inventory revenue.

Model Selection: Price Regression

1. Chosen Technique & 2. Expected Outcome



3. Justification: Why Neural Network?

Why use a Neural Network over XGBoost or Random Forest?

Criterion	Random Forest	Neural Network (Chosen)
Non-Linearity	Good/Trees)	Excellent/Activations) ✓
Ecosystem	Scikit-Learn	TensorFlow(Unified) ✓
Complexity	Lower	High Capacity

Key Reason: Unified Stack

Streamlined deployment pipeline (single library dependency) and easier future integration (e.g., multi-modal learning).

Price Model: Synthetic Data Engine

1. The Logic Engine

Since real-world pet price data is unstructured, we use a Python script (generate_price_dataset.py) to apply market rules to existing images.

Base Characteristics

Lookups per breed: Base Price (e.g., $1500) and Avg Weight (e.g., 30kg).

Price Formula

Final Price = Base × (Age Factor) + Health Bonus + Origin Premium + Random Noise

2. Generated Output Schema

The script outputs a CSV file with 7 features per sample:

Type / Breed	Age	Weight	Health	Origin	Target Variable:
"Dog", "Golden Retriever"	12 Months (Younger = $$$)	29.5 kg	2 (Excellent)	"Scotland" (Premium)	**Price: ↑ $1,850.50**

Preparing Data for AI

1. Feature Engineering

A. Normalization (Scaling) Numerical

Ensures large numbers don't dominate the model.

`Age (30 mo)` → `÷ 60 (Max)` → `0.50`

`Weight (10 kg)` → `÷ 50 (Max)` → `0.20`

B. Encoding (LabelEncoder) Categorical

Converts text labels into unique ID numbers.

`Breed: "Poodle"` → `Lookup ID` → `28`

`Country: "USA"` → `Lookup ID` → `1`

2. Network Architecture

Input Layer (7 Neurons)

↓

Deep Hidden Layers

↓

Output Layer (1 Neuron)

↓

Linear Activation = Predicted Price ($)

Technical Q&A: Core Decisions

Frontend Architecture

Why Next.js (SSR) instead of standard React?

We chose **Server-Side Rendering** to improve SEO and initial load times. For an e-commerce platform, ensuring product pages are indexable by search engines is critical, which a client-side SPA (Single Page App) often struggles with.

Price Regression

Why separate the Price Model from the Image?

Signal-to-Noise Ratio. A photo doesn't show "Vaccination Status" or "Pedigree." A dedicated MLP using structured metadata (Age, Health, Origin) yields far higher regression accuracy than attempting to estimate value from pixels alone.