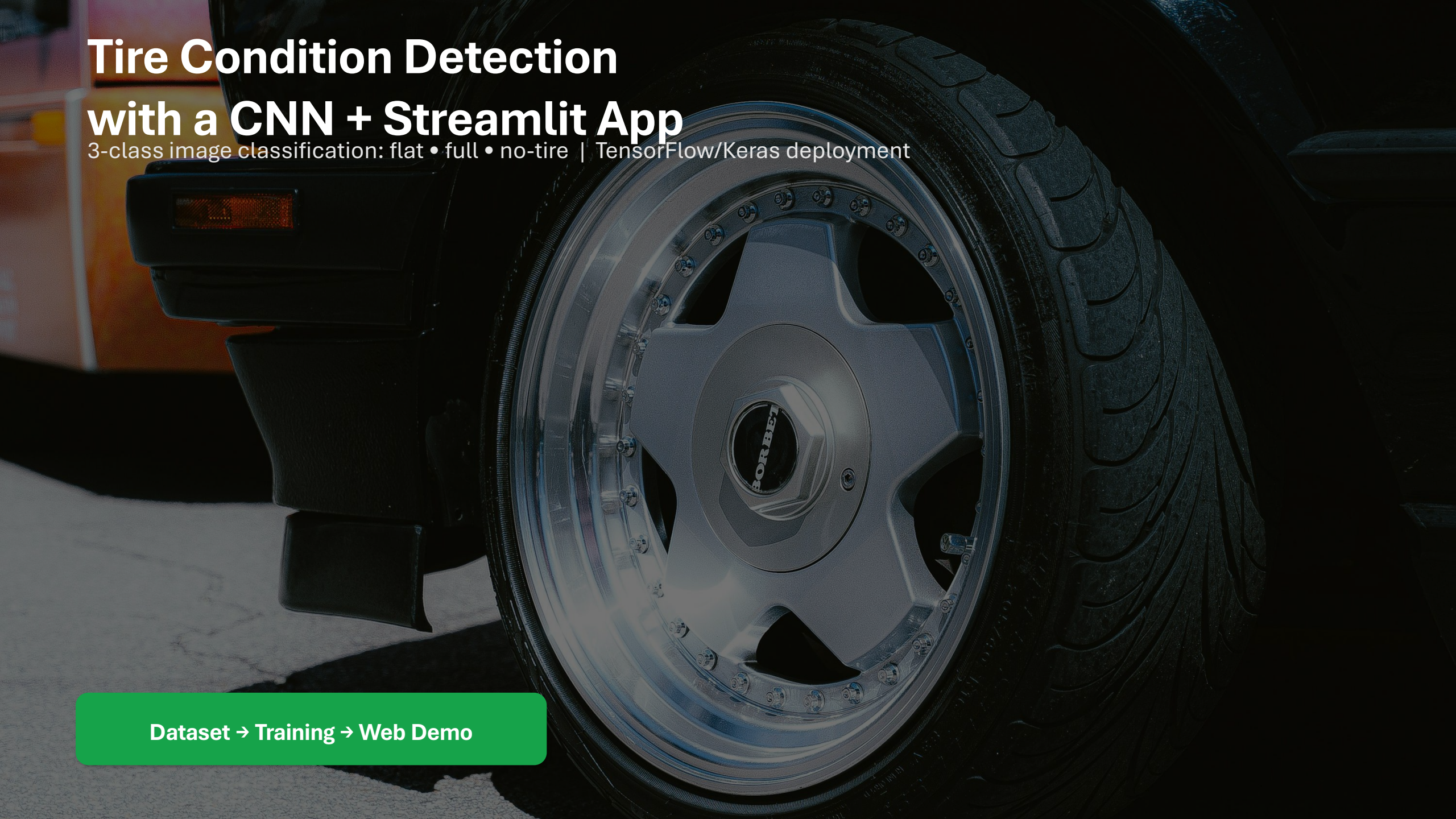


Tire Condition Detection with a CNN + Streamlit App

3-class image classification: flat • full • no-tire | TensorFlow/Keras deployment

Dataset → Training → Web Demo

Why detect tire conditions from images?

Problem & Motivation

Real-world checks are hard

Manual inspections are slow; sensor solutions add hardware + maintenance. Vision-based checks can scale from simple cameras.

⚠

Safety impact

Flat or missing tires reduce stability and braking performance → higher risk incidents.

$

Cost & coverage

Cameras are cheaper and easier to deploy than specialized sensors (especially at scale).

🧠

ML opportunity

CNNs learn robust visual features (edges → textures → parts) directly from pixels, outperforming brittle hand-crafted rules in varied lighting/backgrounds.

What we built

Goal, Scope, and Deliverables

Goal

- Classify a user-uploaded RGB image into one of 3 classes: flat, full, or no-tire.
- Return a predicted label and confidence score (softmax probability).
- Deploy as a lightweight web app for interactive demos.

Deliverables

- Trained CNN model (.h5)
- Training curves (loss/accuracy)
- Streamlit inference UI
- Reproducible config (YAML)

Output example:
Prediction: full
Confidence: 89.44%

How data is organized

Dataset

Classes and split

- 3 classes: flat, full, no-tire
- Total images: 900
- Split: 80% train (720), 20% val (180)

Directory structure

```
data/raw/
  flat/
  full/
  no-tire/
```

Why a hold-out validation set?

Validation estimates generalization to unseen images and guides model selection (e.g., best epoch).

Why 3 classes (flat/full/no-tire)?

Covers the most actionable states for a demo: under-inflation, normal, and missing tire (data sanity).

Example tire image (visual variability)

Make the model more robust

Preprocessing & Augmentation

Pipeline

- Resize to 240×240 to standardize input and control compute.
- Normalize pixels with rescale = 1/255 to stabilize training.
- Augment only the training split: rotation (±10°), zoom (±10%), horizontal flip.
- Keep validation un-augmented to measure true generalization.

Reasoning for these choices

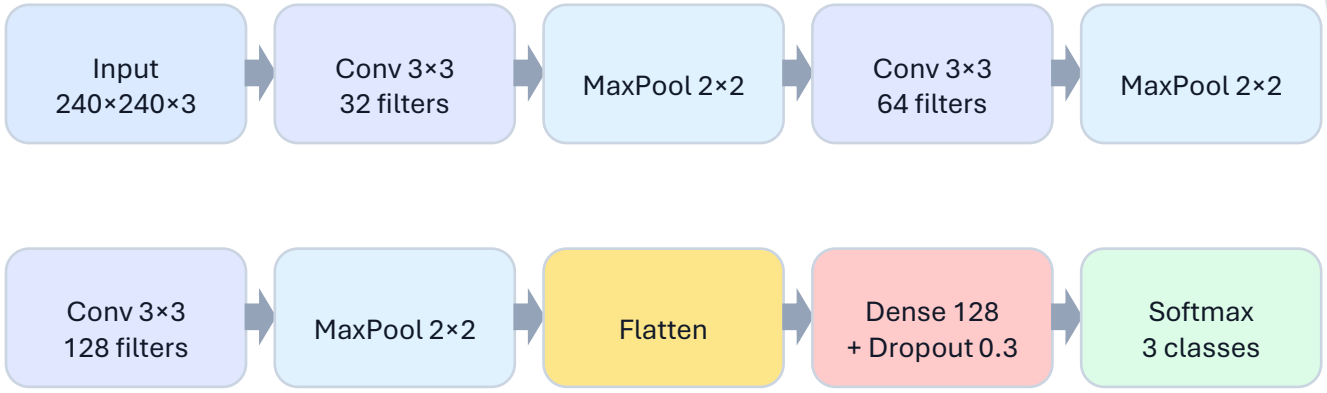
- Small geometric transforms simulate viewpoint variance
- Flip helps when left/right orientation is irrelevant
- Mild ranges reduce risk of generating unrealistic tires
- Data augmentation is a practical regularization tool

Implementation: Keras ImageDataGenerator

Simple CNN baseline (custom)

Model Architecture

Feature extractor → classifier head



Reasoning for choosing this CNN

- 1) Baseline-first: easy to train & debug
- 2) 3 conv blocks capture increasing complexity
- 3) Small 3×3 kernels are standard for locality
- 4) Dropout reduces overfitting with limited data
- 5) Softmax fits multi-class output

What a Conv2D layer computes

Forward Pass: Convolution

Convolution (with activation)

$$Y_{i,j,k} = \sigma\Big(\sum_{u=1..r} \sum_{v=1..s} \sum_{c=1..C} W_{u,v,c,k} \cdot X_{i+u,\, j+v,\, c} + b_k \Big)$$

Interpretation

- X is the input feature map (pixels or previous layer activations).
- W is the learnable kernel (filter) weights; b_k is the bias for filter k.
- The triple sum aggregates local neighborhoods across channels.
- σ is ReLU in this project: σ(x)=max(0,x).

Why it works for tires

- Early filters learn edges/curves (wheel boundary).
- Deeper filters learn textures (tread) and deformation cues (flat).
- Pooling adds translation tolerance (position shifts).

How predictions are made and optimized

Decision Rule & Training Objective

Softmax → class probability

$$p_i = \exp(z_i) / \Sigma_{j=1..3} \exp(z_j)$$

$$\hat{y} = \text{argmax}_i\ (p_i) \quad ; \quad \text{confidence} = \max_i(p_i)$$

Categorical cross-entropy loss

$$L = -\Sigma_{i=1..3}\ y_i \cdot \log(p_i)$$

Minimizing L encourages high probability on the true class.

How this maps to your code

In Streamlit: pred = model.predict(tensor)[0] returns softmax probabilities.
idx = argmax(pred) selects the class; confidence = pred[idx].
During training: model.compile(loss="categorical_crossentropy", optimizer="adam") minimizes cross-entropy over batches.

Training Setup

Core hyperparameters

Image size: 240×240
Batch size: 32
Epochs: 10
Classes: 3
Split: 80/20 via validation_split

Why Adam + Dropout?

Adam adapts learning rates per-parameter → strong default for CNN training.
Dropout (0.3) reduces co-adaptation and helps when Dense layer is large relative to dataset size.

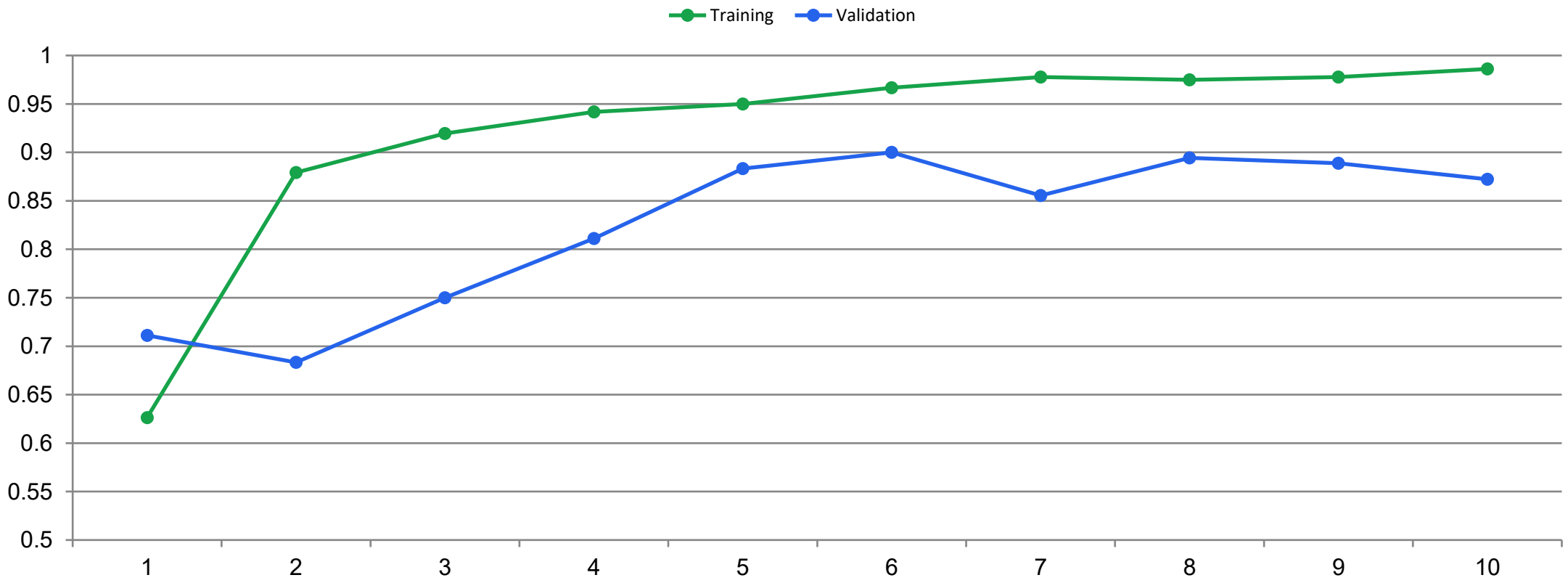
Practical tip: select best epoch by validation accuracy (early stopping).

Implementation stack

- TensorFlow / Keras (model, training)
- ImageDataGenerator (augmentation)
- Streamlit (web UI)
- YAML config (paths & class names)

Training and validation accuracy

Results



Best validation accuracy

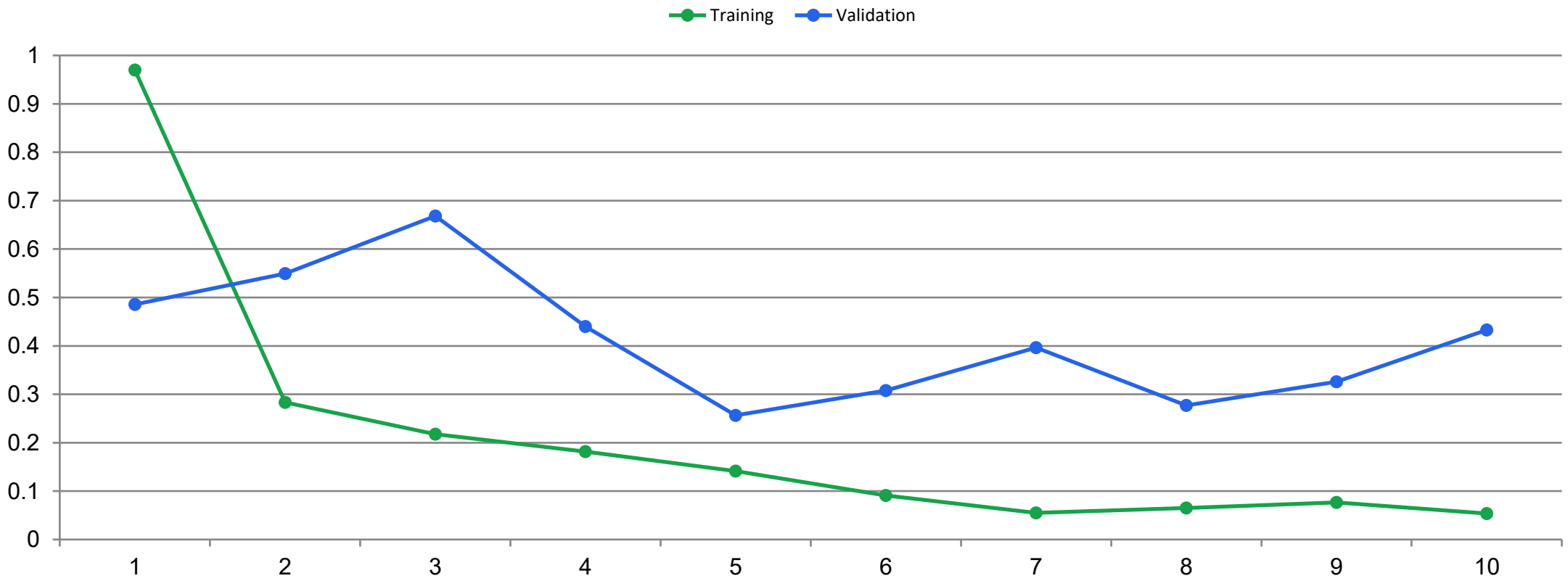
Peak val_acc ≈ 0.90 at Epoch 6 (candidate checkpoint).

Reading the curve

Training rises steadily; validation fluctuates → mild

Training and validation loss

Results



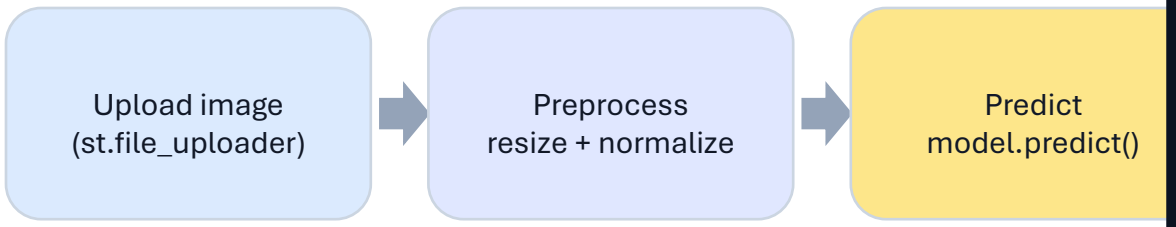
Interpretation

Training loss keeps dropping; validation loss bottoms out around Epoch 5–6 then rises → consider early stopping or stronger regularization.

Streamlit inference app

Deployment

Inference flow



Key implementation choices

- • YAML config keeps paths/class names centralized
- • st.cache_resource loads the model once (fast reruns)
- • preprocess_image mirrors training normalization
- • Output: label + confidence (softmax)

```
@st.cache_resource
def load_model():
    return tf.keras.models.load_model(path)

img = Image.open(uploaded).convert("RGB")
tensor = preprocess_image(img, IMG_SIZE)
pred = model.predict(tensor)[0]
idx = np.argmax(pred)
label = CLASS_NAMES[idx]
conf = pred[idx]
```

What the results suggest

Discussion

Learning behavior

Training accuracy rises to ~0.99, while validation peaks at ~0.90 then fluctuates. This is typical when model capacity is high relative to dataset size.

Model capacity hotspot

Flatten → Dense(128) dominates parameters (~12.8M). With limited images, the Dense layer can overfit. Using GlobalAveragePooling can cut parameters dramatically and often improves generalization.

Actionable improvements

- • Early stopping on best val_acc (≈ Epoch 6)
- • Replace Flatten with GlobalAveragePooling
- • Stronger augmentation (color jitter) if lighting varies
- • Add confusion matrix + per-class precision/recall
- • Calibrate probabilities (temperature scaling) if needed

Where this prototype can fail

Limitations

Key limitations

- Dataset size and diversity: 900 images may not cover night scenes, rain, motion blur, or extreme angles.
- Background bias: model can learn correlated background cues (garage vs roadside) rather than tire shape.
- Ambiguity: flat vs full can be subtle from some viewpoints.
- Confidence ≠ calibrated probability: softmax scores can be overconfident without calibration.
- Real deployment needs robustness testing (cross-device/camera variation).

Next steps to reach production quality

Future Work

Training & evaluation

- Early stopping + checkpointing
- Learning-rate scheduling
- Confusion matrix + macro F1
- Cross-validation (if data small)

Model improvements

- GlobalAveragePooling to reduce parameters
- Transfer learning (e.g., MobileNet/EfficientNet)
- Grad-CAM for explainability
- Quantization for edge devices

Deployment upgrades

- • Save model in native Keras format (.keras)
- • Add batch inference + caching for throughput
- • Add input validation + OOD detection
- • Track experiments (MLflow/W&B)
- • Package as Docker for reproducible demos

Conclusion

- A simple 3-block CNN can classify tire conditions with strong validation performance (peak ~90%).
- Data augmentation + dropout help generalization on limited datasets.
- Streamlit provides an easy path to an interactive demo (upload → preprocess → predict).

Questions?

Thank you