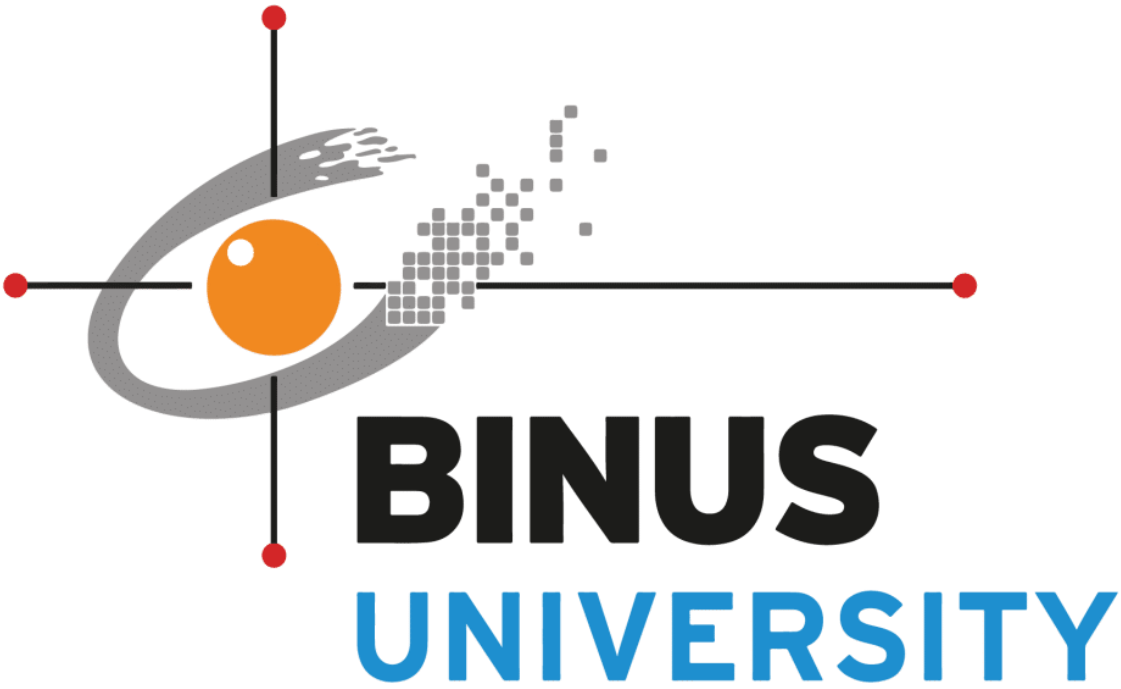
Tire Condition Detection Using Convolutional Neural Networks (CNN) and Streamlit Deployment



Team Members: Owen Figo – 2702358664 Aufar Rizkullah B - 2702379694

Course Code: LD01

Semester: 5

Abstract

Tire condition monitoring is a critical factor for road safety and vehicle maintenance. Conventional approaches rely on manual inspection or sensor-based systems such as tire-pressure monitoring, but these methods can be time-consuming, subjective, or dependent on specialized hardware. This project proposes an image-based tire condition classification system using a Convolutional Neural Network (CNN) trained to recognize three states: flat, full, and no-tire. The dataset used in this project contains 900 labeled RGB images, split into 720 training images and 180 validation images using a directory-based generator with validation_split=0.2. The training pipeline applies standard normalization (rescale=1/255) and augmentation (rotation, zoom, horizontal flip) to improve robustness to viewpoint and illumination changes. The CNN architecture consists of three convolutional blocks (32, 64, 128 filters) followed by a dense classifier with dropout regularization. Training is performed for 10 epochs using the Adam optimizer and categorical cross-entropy loss. Experimental results show fast convergence: training accuracy improves from 62.6% to 98.6%, while validation accuracy reaches 90.0% at its best epoch and finishes at 87.2%. The trained model is deployed in a Streamlit web application that allows users to upload an image and receive a predicted class label and confidence score. This report provides a detailed methodology, explicit mathematical formulation of the CNN decision function (softmax + argmax), the training objective and optimization process, and an extended discussion of limitations and future improvements.

Keywords

Tire condition classification; convolutional neural network; softmax decision; categorical cross-entropy; data augmentation; Streamlit deployment; TensorFlow; Keras; model serialization; reproducible ML.

1. Introduction

Tires are a fundamental component of road vehicles, providing traction, stability, and load distribution. A tire in poor condition can reduce braking performance, increase hydroplaning risk, and compromise steering control. In fleet operations, parking facilities, and transportation safety contexts, tire inspection is often treated as a routine task; however, inspection procedures vary widely and depend heavily on human attention and expertise.

Image-based inspection offers a practical alternative because cameras are low-cost and ubiquitous. A single photo of the wheel region can contain strong visual cues about tire inflation state (e.g., sidewall deformation), tire presence (wheel missing), and related conditions. Deep learning, particularly Convolutional Neural Networks (CNNs), has proven effective at extracting such cues automatically from pixel data and mapping them to a discrete class label.

This project addresses a simplified yet useful task: classify an input image into one of three categories—flat, full, or no-tire. Although this task is narrower than industrial defect detection (e.g., crack localization or tread-depth measurement), it provides a complete, end-to-end workflow from dataset loading and model training to deployment in a user-facing web application.

In addition to model training, the project demonstrates best practices for small-scale ML systems: configuration management via YAML, consistent preprocessing across training and inference, and caching of expensive resources (model loading) in the web application layer.

1.1 Problem Statement and Objectives

Given an RGB image x showing a vehicle wheel region, the goal is to predict the tire condition $y \in \{\text{flat, full, no-tire}\}$. The objectives are: (1) construct a supervised dataset loader and preprocessing pipeline; (2) design and train a CNN classifier; (3) evaluate training behavior and validation performance; (4) deploy the trained model using Streamlit for interactive inference; and (5) document the system with explicit mathematical formulations of the decision rule and training objective.

1.2 Scope and Assumptions

The scope is limited to image-level classification. The model does not perform object detection, segmentation, or localization of tire defects. Each image is assumed to contain sufficient visual information for one of the three classes. Because the dataset source is not the

focus of this report, the methodology emphasizes reproducible training and deployment given an existing labeled image directory.

1.3 Organization of the Report

Section 2 reviews related work. Section 3 explains CNN theory and the mathematical decision rule. Section 4 details the dataset interface, preprocessing, augmentation, and architecture. Section 5 describes experimental setup and metrics. Section 6 reports results. Section 7 explains Streamlit deployment. Section 8 discusses limitations and validity threats. Section 9 concludes and outlines future work. Appendices provide configurations and code excerpts.

2. Related Work and Literature Review

Research on tire inspection spans multiple modalities and objectives: tread defect detection in industrial settings, crack detection on sidewalls, radiographic internal defect classification, and consumer-focused tire state monitoring. Many studies treat the task as detection or segmentation rather than classification, especially when the objective is to locate cracks or foreign objects. Nevertheless, classification remains an important subproblem when the output is a discrete state or quality grade.

Deep learning has become dominant in visual inspection due to its ability to learn discriminative features end-to-end. In tire defect detection, studies have employed lightweight CNN backbones, transfer learning, and detection frameworks (YOLO/Faster R-CNN) to support real-time inspection and localization. The present project focuses strictly on RGB image classification, which is simple to deploy because it requires only a camera and a lightweight inference application.

Compared to large industrial datasets, coursework datasets can be small and biased. Therefore, augmentation and careful validation are important to avoid inflated performance estimates.

2.1 CNN-Based Tire Image Inspection

CNNs are frequently applied to tire quality assessment because they learn hierarchical representations of edges, textures, and object parts. Research in tire inspection includes crack detection, tread defect analysis, and tire quality grading; these problems often share challenges such as varying illumination, rubber texture, and complex backgrounds.

2.2 Detection and Segmentation Approaches

When the objective includes defect localization, object detection (Faster R-CNN, YOLO) and instance segmentation (Mask R-CNN) are common choices. These methods require bounding-box or pixel-level labels, increasing annotation effort. Classification models like ours are cheaper to develop but do not provide defect location.

2.3 Baseline vs Transfer Learning

A simple CNN trained from scratch can perform well on modest datasets if the visual cues are clear. However, transfer learning using ImageNet-pretrained backbones often improves generalization, especially under dataset shifts. A MobileNet/EfficientNet backbone is a strong candidate due to good accuracy-efficiency trade-offs.

3. Theoretical Background

This section summarizes the theoretical foundations required to understand the implemented CNN classifier and its training dynamics. We review convolution, pooling, nonlinear activations, softmax-based multi-class decisions, the categorical cross-entropy objective, and optimization with Adam.

3.1 Convolution and Feature Learning

CNNs exploit local spatial correlations by applying learned kernels across the image. Parameter sharing reduces the number of free parameters and encourages translation-equivariant features.

3.1.1 Convolution Operation

$$Yi,j,k = \sigma(\sum u = 1r\sum v = 1s\sum c = 1CWu,v,c,k \cdot Xi + u,j + v,c + bk)$$

The project uses padding='valid', so spatial dimensions shrink after each convolution. This is reflected in the layer output shapes in Table 1.

3.1.2 Max Pooling

Max pooling downsamples the feature maps, reducing compute in later layers and providing modest invariance.

3.2 Softmax Decision Function

The network outputs logits z and converts them into probabilities p using softmax: $p\_i = exp(z\_i)/\Sigma\_j\ exp(z\_j)$. The predicted class is $\hat{y} = argmax\_i\ p\_i$, and the confidence shown in Streamlit is $Conf = max\_i\ p\_i$

3.3 Cross-Entropy and Backpropagation

For one-hot labels y, categorical cross-entropy is $L(y,p) = -\Sigma\_i\ y\_i\ log(p\_i)$. With softmax, the gradient simplifies to ∂L/∂z = p - y

3.4 Adam Optimization (Update Equations)

For gradient g_t: $m\_t = \beta 1\ m\_\{t-1\} + (1-\beta 1)\ g\_t, v\_t = \beta 2\ v\_\{t-1\} + (1-\beta 2)\ g\_t\text{^}2;\ \hat{m}\_t = m\_t/(1-\beta 1\text{^}t), \hat{v}\_t = v\_t/(1-\beta 2\text{^}t);\ \theta\_\{t+1\} = \theta\_t - \alpha \cdot \hat{m}\_t/(sqrt(\hat{v}\_t) + \varepsilon)$.

3.5 Regularization

Dropout and data augmentation reduce overfitting. Dropout randomly drops units during training; augmentation synthetically increases training diversity.

4. Methodology

This section describes the dataset interface, preprocessing, augmentation policy, model architecture, and training procedure implemented in the project.

4.1 Configuration Management

Hyperparameters and paths are stored in a YAML configuration file. At inference time, Streamlit loads the same config to guarantee consistent IMG_SIZE and CLASS_NAMES.

4.2 Dataset Structure and Class Mapping

The dataset contains 900 RGB images organized into three folders (one per class). Keras flow_from_directory loads images and assigns class indices. Class names are explicitly defined in YAML as [flat, full, no-tire] to prevent mismatch during deployment.

4.3 Train/Validation Split

The project uses validation_split=0.2 to hold out 20% of images for validation. Logs show 720 training images and 180 validation images. For future work, use an explicit split and image deduplication to minimize leakage.

4.4 Preprocessing and Normalization

Each image is resized to 240×240 pixels and normalized by dividing by 255 to map values to [0,1]. Inference preprocessing must match training preprocessing.

```
def preprocess_image(img_pil, img_size):
    img = img_pil.resize((img_size, img_size))
    arr = np.array(img).astype("float32") / 255.0
    arr = np.expand_dims(arr, axis=0)
    return arr
```

4.5 Data Augmentation

Augmentation includes rotation_range=10, zoom_range=0.1, and horizontal_flip=True to simulate minor camera rotations and distance changes.

4.6 CNN Architecture

The model architecture is: Conv2D(32)-MaxPool, Conv2D(64)-MaxPool, Conv2D(128)-MaxPool, Flatten, Dense(128), Dropout(0.3), Dense(3,softmax).

Table 1. Model architecture and parameter count summary.

Layer	Output shape	Kernel/Units	Parameters
Conv2D	(238, 238, 32)	3×3, 32	896
MaxPool	(119, 119, 32)	2×2	0
Conv2D	(117, 117, 64)	3×3, 64	18,496
MaxPool	(58, 58, 64)	2×2	0
Conv2D	(56, 56, 128)	3×3, 128	73,856
MaxPool	(28, 28, 128)	2×2	0
Flatten	(100,352)	-	0
Dense	(128)	128	12,845,184
Dropout	(128)	p=0.3	0
Dense (Softmax)	(3)	3	387

4.7 Parameter Count Derivation

Conv2D parameters: $(r \cdot s \cdot C_{in} + 1) \cdot C_{out}$. Dense parameters: $(d + 1) \cdot h$. Example: first Conv2D has $(3 \cdot 3 \cdot 3 + 1) \cdot 32 = 896$ parameters; Dense(128) has $(100{,}352 + 1) \cdot 128 = 12{,}845{,}184$ parameters.

5. Experimental Setup

This section documents training configuration, metrics, and protocol.

5.1 Training Hyperparameters

Hyperparameters: img_size=240, batch_size=32, epochs=10, num_classes=3. Optimizer: Adam. Loss: categorical cross-entropy. Metric: accuracy.

Table 2. Training configuration.

Parameter	Value
Training samples	720
Validation samples	180
Batch size	32
Epochs	10
Optimizer	Adam
Loss	Categorical cross-entropy
Augmentation	rotation=10°, zoom=0.1, horizontal_flip

5.2 Evaluation Metrics

Accuracy = correct/total. Confusion matrix M counts predictions by true vs predicted classes. Per-class precision, recall, and F1-score are derived from M.

5.3 Protocol and Reproducibility

Train for 10 epochs using model.fit(). Best validation accuracy occurs at epoch 6. Recommended: set seeds, pin package versions, and save best checkpoints.

5.4 Recommended Enhancements

Use EarlyStopping, ModelCheckpoint, and learning rate scheduling (ReduceLROnPlateau) to improve stability and select the best-performing model.

6. Results

This section summarizes metrics across epochs and interprets learning behavior.

6.1 Training Log Summary

Table 3 lists recorded metrics per epoch.

Table 3. Training history metrics per epoch.

Epoch	Train Acc	Train Loss	Val Acc	Val Loss
1	0.6264	0.9699	0.7111	0.4857
2	0.8792	0.2834	0.6833	0.5495
3	0.9194	0.2177	0.7500	0.6683
4	0.9417	0.1818	0.8111	0.4400
5	0.9500	0.1411	0.8833	0.2565
6	0.9667	0.0912	0.9000	0.3078
7	0.9778	0.0552	0.8556	0.3965
8	0.9750	0.0649	0.8944	0.2771
9	0.9778	0.0768	0.8889	0.3258
10	0.9861	0.0537	0.8722	0.4327

6.2 Interpretation of Curves

Validation accuracy peaks at epoch 6 and decreases slightly afterward, indicating mild overfitting. Checkpointing the best epoch is recommended.

6.3 Confusion Matrix (Recommended)

A confusion matrix and per-class metrics should be reported in future work to identify which classes are most frequently confused.

7. Deployment and System Design

The trained model is integrated into a Streamlit application for interactive inference.

7.1 UI Workflow

Upload image → show preview → display predicted class and confidence.

7.2 Model Loading and Caching

@st.cache_resource caches the model across reruns, improving responsiveness.

7.3 Decision Rule (Formula)

$$p = f\_\theta(\tilde{x}), \hat{y} = argmax\_i\, p\_i, Conf = max\_i\, p\_i.$$

Algorithm 1. Streamlit inference steps.

```
Input: uploaded image file
1) Read image as RGB (PIL) and display
```

```
2) Preprocess: resize, normalize, add batch dimension
3) Predict: p = model.predict(x)[0]
4) idx = argmax(p)
5) Output label and confidence
```

7.4 Project Structure

Recommended: config/, src/, notebooks/, outputs/models/, app.py.

7.5 Serialization Notes

Use .keras format for modern Keras; keep model artifacts versioned.

8. Discussion and Limitations

We discuss performance drivers, limitations, and threats to validity.

8.1 Performance Drivers

CNN learns absence cues (no-tire) and deformation cues (flat vs full) through hierarchical features.

8.2 Overfitting and Remedies

Use early stopping, stronger augmentation, reduced capacity, and more diverse data.

8.3 Domain Shift Limitations

Different cameras/lighting/weather can reduce accuracy; evaluate on representative test data.

8.4 Threats to Validity

Potential leakage via near-duplicates; label ambiguity between flat and full; limited external validity.

8.5 Responsible Use

Treat predictions as decision support; add uncertainty handling and manual review when needed.

9. Conclusion and Future Work

This project implements a CNN-based tire condition classifier and deploys it as a Streamlit web application. The model converges quickly and achieves strong validation performance, peaking at 90.0% validation accuracy. The pipeline demonstrates dataset loading, augmentation, CNN training, evaluation, and deployment.

Future work: create a test set; add early stopping/checkpoints; adopt transfer learning; add Grad-CAM; extend to detection/segmentation; migrate to .keras and add model versioning.

References

[1] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-Based Learning Applied to Document Recognition,” Proceedings of the IEEE, 1998.

[2] D. P. Kingma and J. Ba, “Adam: A Method for Stochastic Optimization,” arXiv:1412.6980, 2014.

[3] N. Srivastava et al., “Dropout: A Simple Way to Prevent Neural Networks from Overfitting,” Journal of Machine Learning Research, 2014.

[4] TensorFlow API Documentation: ImageDataGenerator, Softmax, and CategoricalCrossentropy.

[5] TensorFlow Tutorials: Saving and loading Keras models (.keras format).

[6] Streamlit Documentation: st.cache_resource and st.file_uploader.

[7] S. L. Lin et al., “Research on tire crack detection using image deep learning method,” Scientific Reports, 2023.

[8] R. A. A. Saleh et al., “End-to-end tire defect detection model based on transfer learning,” Neural Computing and Applications, 2024.

[9] Z. Wu et al., “Tire Defect Detection Based on Faster R-CNN,” 2020.

[10] X. Cui et al., “Tire Defects Classification with Multi-Contrast Images,” 2018.

Appendix A. Configuration File

```
img_size: 240
batch_size: 32
epochs: 10
num_classes: 3

data_dir: "data/raw"
model_path: "outputs/models/tire_cnn.h5"

class_names:
  - flat
  - full
  - no-tire
```

Appendix B. Streamlit Application (Core Logic)

```
import sys, os
sys.path.append(os.path.abspath("."))

import streamlit as st
import tensorflow as tf
import yaml
import numpy as np
from PIL import Image
from src.utils import preprocess_image

with open("config/config.yaml") as f:
    config = yaml.safe_load(f)

CLASS_NAMES = config["class_names"]
IMG_SIZE = config["img_size"]

@st.cache_resource
def load_model():
    return tf.keras.models.load_model(config["model_path"])

model = load_model()

st.title(" Tire Condition Detection (Deep Learning CNN)")
uploaded = st.file_uploader("Upload an image", type=["jpg", "jpeg",
"png"])

if uploaded:
    img = Image.open(uploaded).convert("RGB")
    st.image(img, use_column_width=True)
```

```
    tensor = preprocess_image(img, IMG_SIZE)
    pred = model.predict(tensor)[0]
    idx = np.argmax(pred)
    st.subheader(f"Prediction: {CLASS_NAMES[idx]}")
    st.write(f"Confidence: {pred[idx]*100:.2f}%")
```

Appendix C. Training Script (Core Logic)

```
datagen = ImageDataGenerator(
    rescale=1./255,
    validation_split=0.2,
    rotation_range=10,
    zoom_range=0.1,
    horizontal_flip=True
)

train_gen = datagen.flow_from_directory(
    DATA_DIR,
    target_size=(IMG_SIZE, IMG_SIZE),
    batch_size=BATCH_SIZE,
    class_mode="categorical",
    subset="training"
)

val_gen = datagen.flow_from_directory(
    DATA_DIR,
    target_size=(IMG_SIZE, IMG_SIZE),
    batch_size=BATCH_SIZE,
    class_mode="categorical",
    subset="validation"
)

model = Sequential([
    Conv2D(32, (3,3), activation="relu", input_shape=(IMG_SIZE,
IMG_SIZE, 3)),
    MaxPooling2D(),
    Conv2D(64, (3,3), activation="relu"),
    MaxPooling2D(),
    Conv2D(128, (3,3), activation="relu"),
    MaxPooling2D(),
    Flatten(),
    Dense(128, activation="relu"),
    Dropout(0.3),
    Dense(NUM_CLASSES, activation="softmax")
])
```

```
model.compile(optimizer="adam", loss="categorical_crossentropy",
metrics=["accuracy"])
history = model.fit(train_gen, validation_data=val_gen, epochs=EPOCHS)
model.save("../outputs/models/tire_cnn.h5")
```