Automated Fish Health Monitoring for Aquaculture using Deep Learning

- GOKUL

Agenda

INTRODUCTION

PROBLEM STATEMENT

PROJECT OBJECTIVE

CODING

OUTPUTS

Introduction

In fish farming, monitoring fish health is very important because dead fish can quickly affect water quality and spread disease to healthy fish. In large fish farms, manual monitoring is inefficient. So, I built an AI-based system using deep learning that can automatically detect whether a fish is alive or dead using images and video streams.

Problem Statement

In fish farming environments, dead fish are often not detected immediately. This delayed detection can:

- Contaminate water quality
- Spread diseases to healthy fish
- Reduce overall farm productivity
- Increase operational costs

Project Objectives

- To develop a **Deep Learning-based system** for detecting fish health status
- To classify fish as **Alive or Dead** using image data
- To extend the model for **real-time video-based detection**
- To achieve reliable performance under real fish farm conditions
- To reduce manual effort and enable early intervention in fish farming

Dataset Overview

Dataset collected manually from **random internet sources**

Images related to fish farming and aquarium conditions

Two classes:

- **Alive Fish**
- **Dead Fish**

Total images used:

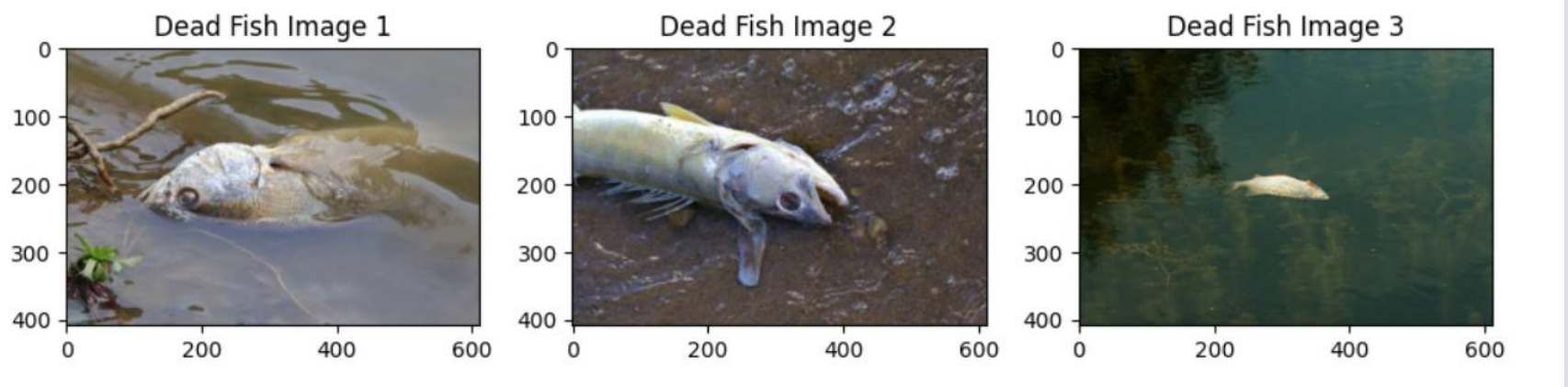
- **Alive Fish: 52**
- **Dead Fish: 52**

Dataset prepared specifically for this project

Packages

```
# importing the packages
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import pathlib
from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.preprocessing.image import ImageDataGenerator, load_img, img_to_array
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing.image import load_img, img_to_array
import cv2
from google.colab.patches import cv2_imshow
from tensorflow.keras.applications import MobileNetV2
```

📷 Sample Dataset Images



Data Augmentation

```
# -------------------------
# DATA AUGMENTATION
# -------------------------
datagen = ImageDataGenerator(
    rescale=1/255.0,
    validation_split=0.2,
    rotation_range=15,
    width_shift_range=0.1,
    height_shift_range=0.1,
    zoom_range=0.1,
    horizontal_flip=True
)

train_data = datagen.flow_from_directory(
    data_dir,
    target_size=(img_size, img_size),
    batch_size=batch_size,
    class_mode="categorical",
    subset="training"
)

val_data = datagen.flow_from_directory(
    data_dir,
    target_size=(img_size, img_size),
    batch_size=batch_size,
    class_mode="categorical",
    subset="validation"
)
```

Convolutional Neural Network

```python
model = models.Sequential([
    layers.Conv2D(32, (3,3), activation='relu',
                  kernel_regularizer=regularizers.l2(0.001),
                  input_shape=(img_size, img_size, 3)),
    layers.MaxPooling2D(2,2),
    layers.Dropout(0.2),

    layers.Conv2D(64, (3,3), activation='relu',
                  kernel_regularizer=regularizers.l2(0.001)),
    layers.MaxPooling2D(2,2),
    layers.Dropout(0.3),

    layers.Conv2D(128, (3,3), activation='relu',
                  kernel_regularizer=regularizers.l2(0.001)),
    layers.MaxPooling2D(2,2),
    layers.Dropout(0.4),

     layers.Conv2D(256, (3,3), activation='relu',
                  kernel_regularizer=regularizers.l2(0.001)),
    layers.MaxPooling2D(2,2),
    layers.Dropout(0.5),

    layers.Flatten(),
    layers.Dense(128, activation='relu',
                 kernel_regularizer=regularizers.l2(0.001)),
    layers.Dropout(0.6),

    layers.Dense(train_data.num_classes, activation='softmax')
])
```

```
Epoch 19/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 617ms/step - accuracy: 0.8060 - loss: 0.8900 - val_accuracy: 0.7500 - val_loss: 1.0045
Epoch 20/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 611ms/step - accuracy: 0.7974 - loss: 0.8844 - val_accuracy: 0.7500 - val_loss: 1.0005
Epoch 21/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 724ms/step - accuracy: 0.8434 - loss: 0.8232 - val_accuracy: 0.7500 - val_loss: 0.9901
Epoch 22/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 3s 928ms/step - accuracy: 0.8669 - loss: 0.8174 - val_accuracy: 0.7000 - val_loss: 1.0151
Epoch 23/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 628ms/step - accuracy: 0.7684 - loss: 0.8829 - val_accuracy: 0.7500 - val_loss: 0.9925
Epoch 24/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 609ms/step - accuracy: 0.7534 - loss: 0.8741 - val_accuracy: 0.7500 - val_loss: 0.9865
Epoch 25/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 646ms/step - accuracy: 0.8551 - loss: 0.7854 - val_accuracy: 0.7000 - val_loss: 0.9960
Epoch 26/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 611ms/step - accuracy: 0.7574 - loss: 0.9225 - val_accuracy: 0.7500 - val_loss: 0.9836
Epoch 27/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 700ms/step - accuracy: 0.8216 - loss: 0.8047 - val_accuracy: 0.7500 - val_loss: 0.9837
Epoch 28/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 869ms/step - accuracy: 0.8650 - loss: 0.8112 - val_accuracy: 0.7500 - val_loss: 0.9699
Epoch 29/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 757ms/step - accuracy: 0.8454 - loss: 0.8300 - val_accuracy: 0.7500 - val_loss: 0.9579
Epoch 30/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 606ms/step - accuracy: 0.8728 - loss: 0.7851 - val_accuracy: 0.7500 - val_loss: 0.9515
Model saved successfully!
```

CNN Output

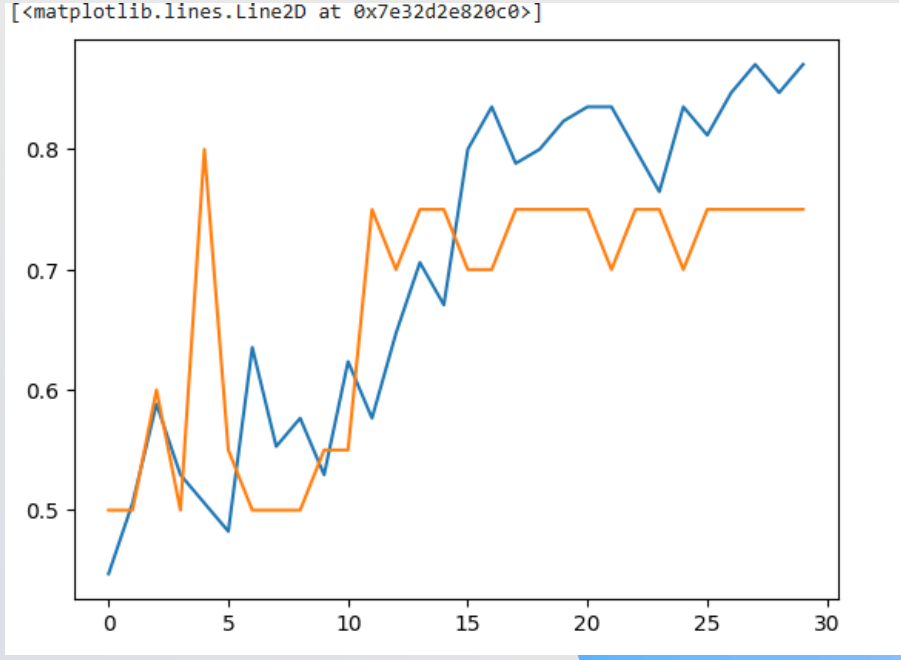
```
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 582ms/step
Alive: 0.66 | Dead: 0.34 → Prediction: ALIVE
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step
Alive: 0.63 | Dead: 0.37 → Prediction: ALIVE
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step
Alive: 0.40 | Dead: 0.60 → Prediction: DEAD
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step
Alive: 0.23 | Dead: 0.77 → Prediction: DEAD
```

```
[<matplotlib.lines.Line2D at 0x7e32d2e820c0>]
```



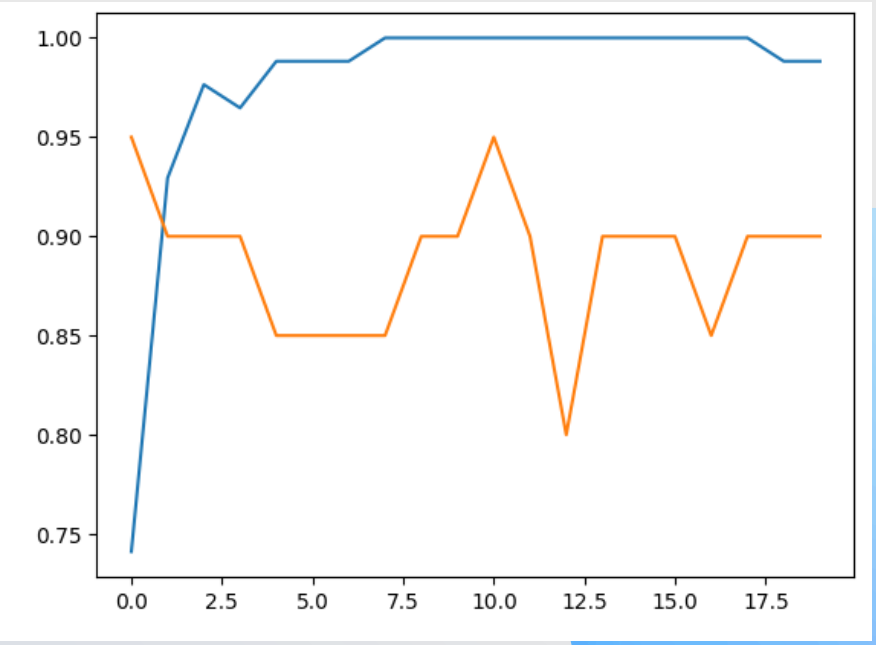
Training Accuracy vs Validation Accuracy

MobileNetV2 Transfer Learning

```python
# ------------------------------
# MobileNetV2 Transfer Learning
# ------------------------------
base_model = MobileNetV2(
    weights="imagenet",
    include_top=False,
    input_shape=(img_size, img_size, 3)
)

base_model.trainable = False  # freeze weights

# Unfreeze last 30 layers
for layer in base_model.layers[:-30]:
    layer.trainable = False
for layer in base_model.layers[-30:]:
    layer.trainable = True

model = models.Sequential([
    base_model,
    layers.GlobalAveragePooling2D(),
    layers.Dense(256, activation='relu'),
    layers.Dropout(0.4),
    layers.Dense(train_data.num_classes, activation='softmax')
])

model.compile(
    optimizer=tf.keras.optimizers.Adam(1e-3),   # VERY IMPORTANT
    loss="categorical_crossentropy",
    metrics=["accuracy"]
)
```

```
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 676ms/step - accuracy: 1.0000 - loss: 0.0022 - val_accuracy: 0.9000 - val_loss: 1.3775
Epoch 11/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 602ms/step - accuracy: 1.0000 - loss: 0.0059 - val_accuracy: 0.9500 - val_loss: 1.0431
Epoch 12/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 707ms/step - accuracy: 1.0000 - loss: 5.5990e-04 - val_accuracy: 0.9000 - val_loss: 1.7603
Epoch 13/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 606ms/step - accuracy: 1.0000 - loss: 9.5940e-04 - val_accuracy: 0.8000 - val_loss: 1.0398
Epoch 14/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 614ms/step - accuracy: 1.0000 - loss: 0.0014 - val_accuracy: 0.9000 - val_loss: 0.2115
Epoch 15/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 3s 928ms/step - accuracy: 1.0000 - loss: 9.4156e-04 - val_accuracy: 0.9000 - val_loss: 0.2310
Epoch 16/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 599ms/step - accuracy: 1.0000 - loss: 0.0027 - val_accuracy: 0.9000 - val_loss: 1.6547
Epoch 17/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 598ms/step - accuracy: 1.0000 - loss: 0.0014 - val_accuracy: 0.8500 - val_loss: 0.9920
Epoch 18/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 602ms/step - accuracy: 1.0000 - loss: 0.0014 - val_accuracy: 0.9000 - val_loss: 2.6324
Epoch 19/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 595ms/step - accuracy: 0.9816 - loss: 0.0261 - val_accuracy: 0.9000 - val_loss: 2.4306
Epoch 20/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 605ms/step - accuracy: 0.9894 - loss: 0.0810 - val_accuracy: 0.9000 - val_loss: 1.1903
Model saved successfully!
```

MobileNetV2 Output



Training Accuracy vs Validation Accuracy

📝 Model Comparison

Model	Correct Predictions (out of 10)	Accuracy	Observation
CNN	9 / 10	~90%	Stable & reliable predictions
MobileNetV2	6 / 10	~60%	Less effective for this dataset

Final Model Selection:
✅ **CNN was chosen** because it provided better accuracy and consistent predictions for fish health classification.

Processing the real time video

```python
def process_video(video_path, save_output=False):
    cap = cv2.VideoCapture(video_path)

    if save_output:
        fourcc = cv2.VideoWriter_fourcc(*"mp4v")
        out = cv2.VideoWriter("output_fish.mp4", fourcc, 20.0, (640, 480))

    frame_count = 0

    while True:
        ret, frame = cap.read()
        if not ret:
            print("Video finished!")
            break

        frame = cv2.resize(frame, (640, 480))
        label, conf = predict_frame(frame)

        text = f"{label.upper()} ({conf*100:.1f}%)"
        color = (0,255,0) if label == "alive" else (0,0,255)

        cv2.putText(frame, text, (20,40),
                    cv2.FONT_HERSHEY_SIMPLEX, 1, color, 2)

        # Show frame in Colab
        cv2_imshow(frame)

        if save_output:
            out.write(frame)

        frame_count += 1

        # Stop after showing few frames (optional)
        if frame_count > 50:
            print("Stopping early to avoid Colab lag.")
            break

    cap.release()
    if save_output:
        out.release()

    print("Done!")
```

🔍 Real time monitoring output

🔍 Real time monitoring output

Thank you

LINKEDIN: GOKUL MADHESHWARAN | LINKEDIN