**Avacado Ripeness level classification using CNN**



Gayathree Basarahalli

EC Utbildning

Data Science Project

202510

# Abstract

In this project, a Streamlit application was deployed for predicting the ripeness level of avocados from uploaded images. The model is based on CNN, trained on a dataset of approximately 14000 images labelled into five ripeness classes. The images used for training and testing and validation were uploaded to and retrieved from Microsoft Azure Cloud Storage. The Streamlit app allows users to upload an avocado image, and the model provides a prediction of its ripeness stage along with a corresponding message based on the predicted stage.

Table of Contents

[Abstract 2](#_Toc212832612)

[1 Introduction 4](#_Toc212832613)

[2 Theory 5](#_Toc212832614)

[2.1 Data Exploration 5](#_Toc212832615)

[2.2 Convolutional Neural Network(CNN): 6](#_Toc212832616)

[2.3 Microsoft Azure Cloud Storage 6](#_Toc212832617)

[2.4 Weights and Biases 6](#_Toc212832618)

[2.5 Data Augmentation 7](#_Toc212832619)

[2.6 Image Data Generator 7](#_Toc212832620)

[2.7 Training and Optimization 7](#_Toc212832621)

[2.8 Confusion Matrix, Classification Report, and Accuracy : 7](#_Toc212832622)

[3 Method 8](#_Toc212832623)

[3.1 Collection and Exploration of Data 8](#_Toc212832624)

[3.2 CNN model Construction and training 8](#_Toc212832625)

[3.3 Preprocessing of Uploaded Images 9](#_Toc212832626)

[3.4 Deployment via Streamlit 9](#_Toc212832627)

[4 Result and Discussion: 10](#_Toc212832628)

[4.1 Data Exploration 10](#_Toc212832629)

[4.2 Model Performance on Test Data 10](#_Toc212832630)

[4.3 Prediction on Uploaded Images 12](#_Toc212832631)

[4.4 Discussion 12](#_Toc212832632)

[5 Conclusions 13](#_Toc212832633)

[Appendix A 14](#_Toc212832634)

[Module 1: upload\_images\_to\_azure.py 14](#_Toc212832635)

[Module 2: retrive\_data.py 15](#_Toc212832636)

[Module 3: model.ipynb 17](#_Toc212832637)

[Module 4: app.py(Streamlit app) 25](#_Toc212832638)

[References 27](#_Toc212832639)

# Introduction

In recent years, artificial intelligence (AI) has received increasing attention and is expected to play an even more significant role in our daily lives in the near future.

A central concept in AI is deep learning (DL), a subset of machine learning (ML), which focuses on algorithms inspired by the structure and function of the human brain, particularly neural networks. Convolutional Neural Networks (CNNs) are a type of deep learning model specifically designed for image analysis and have shown remarkable performance in tasks like object recognition and classification.

In practice, a CNN model is trained on a dataset of images with known labels and deployed to make predictions on new data. Successful deployment requires careful consideration of several aspects, including data collection, preprocessing, model architecture, and computational resources. Each image classification task has unique challenges, and the design choices made during development often determine the model's accuracy and efficiency.

The current report focuses on predicting the ripeness level of avocados from images. Avocado is a highly perishable fruit, and improper handling leads to substantial food waste and economic loss. Automating ripeness prediction can help consumers, retailers, and suppliers make informed decisions about storage, transportation, and consumption. To develop this system, a CNN was trained on a dataset of approximately 14,000 avocado images, labelled into five ripeness stages. The images for training, validation, and testing were stored in and retrieved from Microsoft Azure Cloud Storage.

The main purpose of this report is to develop a Streamlit application that allows users to upload an image of an avocado and receive an accurate prediction of its ripeness stage. To achieve this purpose, the following questions are addressed:

1. How accurately can a CNN classify avocado ripeness into five stages using a dataset of approximately 14,000 images?
2. How effectively can cloud services, such as Microsoft Azure, be used for storage and access to training and testing data?
3. Which image preprocessing techniques are most effective in improving model performance on uploaded user images?
4. How effectively can the Streamlit application process uploaded images and return predictions in a user-friendly and timely manner?
5. What is the potential impact of this avocado ripeness prediction system in reducing food waste and improving consumer and supplier decision-making?

# Theory

## Data Exploration

The avocado ripeness dataset used in this project contains approximately 14,000 images of avocados, labelled into five ripeness stages: unripe, slightly unripe, ripe, slightly overripe, and overripe. Each image is represented as a three-channel (RGB) array, capturing the colour and texture of the avocado surface, which are key indicators of ripeness.

Important characteristics of the dataset and preprocessing considerations include:

* Image size varies; images were resized to a standard dimension (e.g., 128 × 128) for uniform input to the CNN model.
* Pixel intensity values range from 0–255 for each RGB channel.
* Colour features (green to brown spectrum) are critical, so maintaining true colour representation during preprocessing is important.
* Images were stored and retrieved from Microsoft Azure Cloud Storage, ensuring centralized access for training, validation, and testing.

Although the CNN model can learn features automatically, proper preprocessing of user-uploaded images is essential to match the conditions of the training dataset. Preprocessing steps applied include resizing, normalization (scaling pixel values to 0–1), and, in some cases, minor brightness adjustments to handle variation in image capture. Ensuring that uploaded images are similar in format and quality to the training data is crucial for accurate prediction of avocado ripeness.

A close up of an avocado

AI-generated content may be incorrect.

*Figure 1: Example images of each class.*

## Convolutional Neural Network(CNN):

CNNs are a type of deep learning model designed for analysing visual data. They automatically extract hierarchical features from images, starting with simple features like edges and textures and progressing to more complex patterns. The main components of a CNN include:

* **Convolution layers:** Apply filters to the input image to extract features.
* **Pooling layers:** Reduce spatial dimensions and retain important features, which helps decrease computational cost and prevent overfitting.
* **Activation functions (ReLU):** Introduce non-linearity to allow the model to learn complex patterns.
* **Fully connected layers:** Combine extracted features to make final predictions.

In this project, a CNN model was used to classify avocado images into five ripeness stages. The model learns colour, texture, and shape patterns that correspond to each ripeness level.

## Microsoft Azure Cloud Storage

In this project, **Microsoft Azure Blob Storage** was used as a centralized platform for managing the large avocado image dataset. All image folders corresponding to the five ripeness classes were uploaded to an Azure Blob Storage container. This setup ensured secure, scalable, and easily accessible storage for all images used during training, validation, and testing.

The images were retrieved from Azure and converted into **NumPy arrays** for efficient model training and processing. Both the feature arrays (**X**) and corresponding labels (**y**) were then saved back to Azure Blob Storage. This approach significantly reduced local memory usage and improved data handling speed, especially when working with a large dataset of approximately 14,000 images.

Storing pre-processed NumPy arrays in Azure also made the **deployment of the Streamlit application** more efficient, since the model could access clean, structured data directly from the cloud. This ensured smoother scalability and allowed faster data loading during inference without relying on local file systems.

## Weights and Biases

During CNN training, the model learns a set of parameters known as weights and biases.

* Weights determine the importance of different input features (such as colour and texture patterns) during convolution operations.
* Biases help shift the activation function, allowing the network to learn more complex relationships.

These parameters are continuously updated through backpropagation to minimize the loss function, enabling the CNN to make more accurate predictions over time. The trained weights are later saved and loaded during deployment to ensure that the Streamlit application uses the same learned patterns without retraining.

## Data Augmentation

To improve model generalization and prevent overfitting, data augmentation techniques were applied to the training images. Augmentation artificially increases the size and diversity of the dataset by applying random transformations such as:

* Rotation (to simulate different orientations of avocados)
* Horizontal and vertical flips
* Zooming and shifting
* Brightness adjustment

These transformations help the model become more robust to variations in lighting, positioning, and background, improving its ability to classify unseen images accurately.

## Image Data Generator

The **ImageDataGenerator** from TensorFlow’s Keras library was used to preprocess and feed the images into the CNN in batches. This generator automatically performs:

* Image resizing and normalization (scaling pixel values between 0 and 1)
* Application of data augmentation techniques
* Real-time loading of images during training, reducing memory usage.

Using the ImageDataGenerator allowed efficient handling of large image datasets and ensured that each training epoch received a slightly varied version of the dataset, improving model robustness.

## Training and Optimization

The CNN model was trained using the categorical cross-entropy loss function, suitable for multi-class classification problems. The Adam optimizer was used due to its adaptive learning rate and efficiency in handling large datasets. Model training was monitored over several epochs, with key metrics such as accuracy and validation loss tracked to ensure convergence and prevent overfitting.

Callbacks such as ModelCheckpoint and EarlyStopping were implemented:

* **ModelCheckpoint** saved the best-performing model during training.
* **EarlyStopping** halted training if no improvement in validation accuracy was observed, preventing unnecessary computation.

After training, the final model was evaluated on the test dataset to assess its accuracy and generalization ability before being deployed in the Streamlit application.

## Confusion Matrix, Classification Report, and Accuracy :

**Confusion Matrix:** Shows correct predictions along the diagonal and misclassifications off-diagonal. Most errors occurred between adjacent ripeness stages, which are visually similar.

**Classification Report:** Provides precision, recall, and F1-score for each class. All classes showed good values, indicating that the model accurately identifies different ripeness levels.

**Accuracy:** Accuracy measures the proportion of correctly predicted samples out of the total samples. For multi-class problems like avocado ripeness classification, accuracy gives an overall idea of model performance across all classes. However, it may not capture class-wise nuances, especially if some classes have fewer samples. Hence, it is complemented by the confusion matrix and classification report to better understand the model’s performance for each ripeness stage.

# Method

## Collection and Exploration of Data

The first part involved exploring the avocado image dataset according to the following steps:

* Uploading the five folders of avocado images (corresponding to each ripeness stage) to Microsoft Azure Blob Storage.
* Retrieving images from Azure and converting them into NumPy arrays for efficient handling (X for features, y for labels).
* Inspecting image dimensions, pixel intensity ranges, and colour distributions across ripeness stages.
* Resizing all images to a uniform size (128 × 128) and normalizing pixel values to the 0–1 range.
* Performing visual inspection of sample images from each class to ensure quality and consistency.

## CNN model Construction and training

* Dividing the dataset into training, validation, and test sets to evaluate model performance.
* Initializing the **weights and biases** of the CNN layers. The weights were either randomly initialized or loaded from previously saved training to accelerate convergence and improve performance.
* Applying **data augmentation** using TensorFlow’s ImageDataGenerator to increase dataset diversity and improve generalization. Transformations included random rotations, horizontal and vertical flips, zooming, and brightness adjustments.
* Designing the CNN architecture with convolutional layers, pooling layers, ReLU activations, and fully connected layers to automatically extract colour and texture features.
* Using **categorical cross-entropy** as the loss function and **Adam optimizer** for training.
* Implementing callbacks such as **ModelCheckpoint** (to save the best-performing model) and **EarlyStopping** (to prevent overfitting).
* Training the model over multiple epochs, monitoring accuracy and loss on training and validation sets.
* Evaluating the final trained model on the test set to estimate generalization performance before deployment.

## Preprocessing of Uploaded Images

The third part ensured that the Streamlit application could process user-uploaded avocado images correctly:

* Loading the uploaded image and resizing it to 128 × 128 pixels to match the training set.
* Normalizing pixel intensity values to the 0–1 range.
* Optionally adjusting brightness or contrast if the image was captured under unusual lighting conditions.
* Converting the image into a NumPy array and reshaping it to fit the CNN input.
* Passing the pre-processed image to the trained CNN model for prediction.
* Returning the predicted ripeness stage along with a user-friendly message describing the avocado’s condition.

## Deployment via Streamlit

* Integrating the trained CNN model into a Streamlit application.
* Enabling image uploads by users in the app interface.
* Preprocessing each uploaded image using the same pipeline as the training data.
* Generating and displaying ripeness predictions in real time.
* Ensuring smooth interaction and fast response by retrieving data and weights from Azure Blob Storage.

# Result and Discussion:

The results of this project demonstrate the effectiveness of a Convolutional Neural Network (CNN) in predicting avocado ripeness from images. All metrics were calculated on the test set, while qualitative observations were made on uploaded images through the Streamlit application.

## Data Exploration

During the exploration of the avocado dataset, several patterns were observed:

* Images for each ripeness stage showed distinct colour patterns, ranging from dark green (unripe) to brown (overripe).
* Texture became smoother and darker as ripeness increased.
* After resizing all images to **128 × 128 pixels**, pixel intensity values were normalized to 0–1, ensuring uniform input to the CNN.
* Visual inspection confirmed that images were of good quality and did not contain major noise or irrelevant background.

These observations highlighted the **importance of colour and texture features** in distinguishing ripeness stages, which the CNN model could learn automatically.

## Model Performance on Test Data

The CNN model was evaluated on the test set using various metrics:

**Accuracy**:  
The model attained a test accuracy of 74.31%, correctly classifying about three out of four avocado images into the proper ripeness category.

**Confusion Matrix**:

* Correct predictions are displayed along the diagonal.
* Most misclassifications occurred between adjacent ripeness stages (e.g., slightly unripe predicted as unripe or ripe).
* This is understandable due to the visual similarity of adjacent stages, where colour and texture differences are subtle.

A diagram of a confusion matrix

AI-generated content may be incorrect.

*Figure 2: Confusion matrix.*

**Classification Report**:

* **Precision**: Indicates how many predicted samples for a class were correct.
* **Recall**: Indicates how many actual samples of a class were correctly predicted.
* **F1-score**: Combines precision and recall into a single metric.

precision recall f1-score support

0 0.92 0.87 0.90 535

1 0.65 0.79 0.72 334

2 0.75 0.57 0.65 414

3 0.62 0.75 0.68 494

4 0.79 0.71 0.75 430

accuracy 0.74 2207

macro avg 0.75 0.74 0.74 2207

weighted avg 0.76 0.74 0.74 2207

All five classes achieved **acceptable precision and recall values**, confirming that the model could distinguish ripeness stages effectively, though borderline cases remained challenging.

## Prediction on Uploaded Images

The Streamlit application allowed **users to upload avocado images**, and predictions were returned in real time. Observations from uploaded images include:

* Model predictions were **consistent with visual assessment**, particularly for clearly unripe and overripe avocados.
* Minor inaccuracies occurred in images with unusual **lighting, shadows, or partially obscured avocados**.
* **Preprocessing steps** (resizing, normalization, optional brightness adjustment) were crucial to maintain prediction accuracy for uploaded images.

This confirms that the preprocessing pipeline successfully aligns user-uploaded images with the training dataset characteristics.

Upload an image :

A green avocado on a white background

AI-generated content may be incorrect.

*Figure 3: Avacado image .*

**Result: Unripe**

**Message : Need more than 1 week to ripen**

## Discussion

* The model achieved moderate accuracy (~74%) on the test set, which is reasonable given the subtle differences between ripeness stages.
* Data augmentation helped improve generalization by introducing variability in rotation, flip, zoom, and brightness.
* Misclassifications mostly occurred between adjacent stages, suggesting that increasing dataset size or including additional features (e.g., colour histograms) could further improve accuracy.
* The CNN model proved effective for practical use, as evidenced by successful predictions on uploaded images.

# Conclusions

The following conclusions regarding set out objectives and questions we mentioned before.

1. The CNN model achieved good accuracy to differentiate avocado images between five ripening stages. Misclassifications mostly occurred between adjacent classes.
2. Microsoft Azure blob storage is very effective in handling data which helped in easy retrieval using connection strings and prevents local storage issues and enables smooth deployment using Streamlit.
3. Resizing all images to a fixed size and normalizing pixel values to [0,1] improved model consistency and accuracy. Maintaining RGB colour information was crucial for distinguishing subtle ripeness differences.
4. The Streamlit app processed uploaded images within seconds and displayed clear, accurate predictions. Its simple interface made it easy for users to interpret ripeness results quickly.
5. The system helps consumers choose avocados at the right ripeness and supports suppliers in grading and distribution. This reduces food waste and promotes smarter, data-driven decisions across the supply chain.

# Appendix A

# Module 1: upload\_images\_to\_azure.py

from azure.storage.blob import BlobServiceClient

import os

connection\_string = "DefaultEndpointsProtocol=https;AccountName=avacadoimagesstorage;AccountKey=VigxdeaqLmrCXayQCgTFD0Lf7qEL2RVPRFwxSO6gy28u3SlZKXm153slxYmcobUs2YhCb87fdr84+AStjFMDeA==;EndpointSuffix=core.windows.net"

container\_name = "avocado-images"

local\_folder = r"D:\EC\_utbildning\avacado\_ripeness\_level\dataset"

# Connect to Azure Blob Storage

blob\_service\_client = BlobServiceClient.from\_connection\_string(connection\_string)

container\_client = blob\_service\_client.get\_container\_client(container\_name)

# Upload dataset folder with subfolders

print("Uploading dataset folder...")

total\_files = 0

for root, dirs, files in os.walk(local\_folder):

    for file in files:

        file\_path = os.path.join(root, file)

        # Preserve subfolder structure (1,2,3,4,5)

        blob\_path = os.path.relpath(file\_path, local\_folder)

        with open(file\_path, "rb") as data:

            container\_client.upload\_blob(name=blob\_path, data=data, overwrite=True)

        total\_files += 1

        if total\_files % 100 == 0:

            print(f"✅ {total\_files} files uploaded...")

print(f"Upload complete! {total\_files} files uploaded successfully.")

# Module 2: retrive\_data.py

import numpy as np

from azure.storage.blob import ContainerClient

from io import BytesIO

from PIL import Image

import os

container\_sas\_url = os.getenv("AZURE\_CONTAINER\_SAS\_URL")

# Connect to Azure container

container\_client = ContainerClient.from\_container\_url(container\_sas\_url)

print("Retrieving images from Azure...")

X, y = [], []

# Looping through all blobs in container

for i, blob in enumerate(container\_client.list\_blobs()):

    if blob.name.endswith(".jpg") or blob.name.endswith(".png"):

        try:

            # Extract label from folder name (1, 2, 3, 4, 5)

            label = int(blob.name.split("/")[0])

            # Download image as stream

            blob\_client = container\_client.get\_blob\_client(blob)

            stream = BytesIO(blob\_client.download\_blob().readall())

            # Convert image to RGB and resize

            img = Image.open(stream).convert("RGB")

            img = img.resize((128, 128))

            # Convert to NumPy array

            img\_array = np.array(img)

            X.append(img\_array)

            y.append(label)

            # Print progress

            if (i + 1) % 50 == 0:

                print(f"{i + 1} images processed...")

        except Exception as e:

            print(f"⚠️ Skipping {blob.name}: {e}")

print("✅ All images retrieved and converted to NumPy arrays.")

# Convert to NumPy arrays

X = np.array(X)

y = np.array(y)

print("X shape:", X.shape)

print("y shape:", y.shape)

# ✅ Upload X and y to Azure (not saving locally)

import io

print("Uploading X.npy and y.npy to Azure Blob Storage...")

# Convert arrays to bytes in memory

X\_buffer = io.BytesIO()

np.save(X\_buffer, X)

X\_buffer.seek(0)

y\_buffer = io.BytesIO()

np.save(y\_buffer, y)

y\_buffer.seek(0)

# Upload directly to Azure

X\_blob = container\_client.get\_blob\_client("X.npy")

y\_blob = container\_client.get\_blob\_client("y.npy")

X\_blob.upload\_blob(X\_buffer, overwrite=True)

y\_blob.upload\_blob(y\_buffer, overwrite=True)

print("Uploaded X.npy and y.npy to Azure Blob Storage successfully.")

# Module 3: model.ipynb

#Imports

import numpy as np

import matplotlib.pyplot as plt

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from tensorflow.keras.utils import to\_categorical

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, BatchNormalization, SpatialDropout2D,GlobalAveragePooling2D

from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau

from tensorflow.keras.optimizers import Adam

from sklearn.metrics import confusion\_matrix, classification\_report

from sklearn.utils.class\_weight import compute\_class\_weight

from tensorflow.keras import regularizers

from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau

import os

from azure.storage.blob import ContainerClient, BlobClient

from io import BytesIO

X\_blob\_url = os.getenv("AZURE\_X\_BLOB\_URL")

y\_blob\_url = os.getenv("AZURE\_Y\_BLOB\_URL")

print("📥 Downloading arrays directly from Azure...")

#Download directly into memory

X\_stream = BytesIO(BlobClient.from\_blob\_url(X\_blob\_url).download\_blob().readall())

y\_stream = BytesIO(BlobClient.from\_blob\_url(y\_blob\_url).download\_blob().readall())

# Load arrays

X = np.load(X\_stream)

y = np.load(y\_stream) - 1

num\_classes = 5

print("✅ Loaded X and y directly from Azure.")

print("X shape:", X.shape)

print("y shape:", y.shape)

print("X dtype:", X.dtype)

print("y dtype:", y.dtype)

# Visualize number of images per each class

unique, counts = np.unique(y, return\_counts=True)

for cls, cnt in zip(unique, counts):

print(f"Class {cls}: {cnt} samples")

plt.bar(unique, counts)

plt.title("Class Distribution")

plt.xlabel("Class Label")

plt.ylabel("Count")

plt.show()

# Get unique classes

classes = np.unique(y)

# Images of avacado per each class

fig, axes = plt.subplots(1, len(classes), figsize=(15, 3))

for i, cls in enumerate(classes):

# Find first index of this class

idx = np.where(y == cls)[0][0]

axes[i].imshow(X[idx])

axes[i].set\_title(f"Class: {cls}")

axes[i].axis("off")

plt.tight\_layout()

plt.show()

#Random image visualization with rgb colours

# Pick one image

idx = np.random.randint(0, len(X))

img = X[idx]

# Split into RGB channels

R = img[:, :, 0]

G = img[:, :, 1]

B = img[:, :, 2]

# Plot all together

fig, axes = plt.subplots(1, 4, figsize=(12, 3))

axes[0].imshow(img)

axes[0].set\_title(f"Original (Label: {y[idx]})")

axes[0].axis("off")

axes[1].imshow(R, cmap="Reds")

axes[1].set\_title("Red Channel")

axes[1].axis("off")

axes[2].imshow(G, cmap="Greens")

axes[2].set\_title("Green Channel")

axes[2].axis("off")

axes[3].imshow(B, cmap="Blues")

axes[3].set\_title("Blue Channel")

axes[3].axis("off")

plt.tight\_layout()

plt.show()

#Image size

heights = [img.shape[0] for img in X]

widths = [img.shape[1] for img in X]

print(f"Unique heights: {set(heights)}")

print(f"Unique widths: {set(widths)}")

#Pixel intensity

plt.figure(figsize=(6,4))

plt.hist(X.flatten(), bins=50, color='green', alpha=0.7)

plt.title("Pixel Intensity Distribution")

plt.xlabel("Pixel value")

plt.ylabel("Frequency")

plt.show()

#Splitting data

X\_train\_val, X\_test, y\_train\_val, y\_test = train\_test\_split( X, y, test\_size=0.15, random\_state=42, stratify=y )

X\_train, X\_val, y\_train, y\_val = train\_test\_split( X\_train\_val, y\_train\_val, test\_size=0.18, random\_state=42, stratify=y\_train\_val )

# Compute class weights (to make sure the model take equal amount of data from each class)

class\_weights\_values = compute\_class\_weight(

class\_weight='balanced',

classes=np.unique(y\_train),

y=y\_train )

class\_weights = {i: w for i, w in enumerate(class\_weights\_values)}

#Data augmentation (to make model more robust by fine tuning)

train\_datagen = ImageDataGenerator(

rescale=1./255,

rotation\_range=15,

width\_shift\_range=0.1,

height\_shift\_range=0.1,

zoom\_range=0.1,

horizontal\_flip=True,

fill\_mode='nearest' )

val\_datagen = ImageDataGenerator(rescale=1./255)

# Custom generator (creates batches of data for training instead of loading all images into RAM once)

def generator(X\_data, y\_data, batch\_size=32):

while True:

indices = np.arange(len(X\_data))

np.random.shuffle(indices)

for i in range(0, len(indices), batch\_size):

batch\_idx = indices[i:i+batch\_size]

batch\_X = X\_data[batch\_idx].astype('float32') / 255.0

batch\_y = to\_categorical(y\_data[batch\_idx], num\_classes)

sample\_weights = np.array([class\_weights[label] for label in y\_data[batch\_idx]])

yield batch\_X, batch\_y, sample\_weights

train\_gen = generator(X\_train, y\_train, batch\_size=32)

val\_gen = generator(X\_val, y\_val, batch\_size=32)

#CNN model

model = Sequential([

Conv2D(32, (3,3), activation='relu', padding='same', input\_shape=(128,128,3)),

BatchNormalization(),

MaxPooling2D(2,2),

Conv2D(64, (3,3), activation='relu', padding='same'),

BatchNormalization(),

MaxPooling2D(2,2),

Conv2D(128, (3,3), activation='relu', padding='same'),

BatchNormalization(),

Dropout(0.2),

MaxPooling2D(2,2),

Conv2D(256, (3,3), activation='relu', padding='same'),

BatchNormalization(),

Dropout(0.3),

MaxPooling2D(2,2),

GlobalAveragePooling2D(),

Dense(256, activation='relu', kernel\_regularizer=regularizers.l2(0.001)),

Dropout(0.4),

Dense(num\_classes, activation='softmax')

])

# Compile

model.compile(

optimizer=Adam(learning\_rate=1e-4),

loss='categorical\_crossentropy',

metrics=['accuracy']

)

# Callbacks

early\_stop = EarlyStopping(monitor='val\_loss', patience=10, restore\_best\_weights=True)

lr\_reduce = ReduceLROnPlateau(monitor='val\_loss', factor=0.5, patience=3, min\_lr=1e-6, verbose=1)

#Model fit

history = model.fit(

train\_gen,

steps\_per\_epoch=len(X\_train) // 32,

validation\_data=val\_gen,

validation\_steps=len(X\_val) // 32,

epochs=30,

callbacks=[early\_stop, lr\_reduce],

verbose=1

)

val\_accuracies = history.history['val\_accuracy']

best\_epoch = np.argmax(val\_accuracies) # index of highest val\_accuracy

best\_train\_acc = history.history['accuracy'][best\_epoch]

best\_val\_acc = val\_accuracies[best\_epoch]

model.save("avocado\_model.h5")

#Evaluate test accuracy

X\_test\_scaled = X\_test.astype('float32') / 255.0

y\_test\_cat = to\_categorical(y\_test, num\_classes)

test\_loss, test\_acc = model.evaluate(X\_test\_scaled, y\_test\_cat, verbose=1)

print(f" Best Epoch: {best\_epoch + 1}") # +1 because index starts at 0

print(f" Training Accuracy at Best Epoch: {best\_train\_acc:.4f}")

print(f" Validation Accuracy at Best Epoch: {best\_val\_acc:.4f}")

print(f" Test Accuracy:{test\_acc:.4f}")

# Predict class probabilities

y\_pred\_probs = model.predict(X\_test\_scaled)

# Convert probabilities → predicted class labels

y\_pred = np.argmax(y\_pred\_probs, axis=1)

#confusion matrix

cm = confusion\_matrix(y\_test, y\_pred)

print("\n🔹 Confusion Matrix:")

print(cm)

plt.figure(figsize=(6,5))

sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")

plt.title("Confusion Matrix")

plt.xlabel("Predicted Label")

plt.ylabel("True Label")

plt.show()

# Classification report

report = classification\_report(y\_test, y\_pred)

print("\n🔹 Classification Report:")

print(report)

# Module 4: app.py(Streamlit app)

import streamlit as st

import numpy as np

from tensorflow.keras.models import load\_model

from tensorflow.keras.preprocessing import image

import os

# Load Model

MODEL\_PATH = "avocado\_model.h5"

st.title("🥑 Avocado Ripeness Classifier")

st.write("Upload an image of an avocado to check its ripeness level.")

# Try loading model

try:

    model = load\_model(MODEL\_PATH)

    st.success("✅ Model loaded successfully!")

except Exception as e:

    st.error(f"⚠️ Error loading model:\n\n{e}")

    st.stop()

# Ripeness info

ripeness\_info = {

    0: {"label": "Unripe", "message": "Need more than 1 week to ripen"},

    1: {"label": "Early Ripening", "message": "Can eat in 2-3 days"},

    2: {"label": "Perfect", "message": "Perfect for salad"},

    3: {"label": "Overripe", "message": "Perfect for making guacamole"},

    4: {"label": "Spoiled", "message": "Do not eat"},

}

# -----------------------

# File Upload

# -----------------------

uploaded\_file = st.file\_uploader("📸 Upload an avocado image", type=["jpg", "jpeg", "png"])

if uploaded\_file is not None:

    # Show uploaded image

    st.image(uploaded\_file, caption="Uploaded Image", use\_container\_width=True)

    # Preprocess image

    img = image.load\_img(uploaded\_file, target\_size=(128, 128))

    img\_array = image.img\_to\_array(img)

    img\_array = np.expand\_dims(img\_array, axis=0)

    img\_array = img\_array / 255.0

    # Predict

    predictions = model.predict(img\_array)

    predicted\_class = np.argmax(predictions[0])

    # Show ripeness level

    info = ripeness\_info.get(predicted\_class, {"label": "Unknown", "message": "No info available"})

    st.subheader(f"🥑 Ripeness Level: {info['label']}")

    # Show message in red

    st.markdown(f"<p style='color:red; font-size:20px'>{info['message']}</p>", unsafe\_allow\_html=True)

    # Show prediction probabilities

    st.write("🔢 Model prediction probabilities:")

    st.bar\_chart(predictions[0])

# References

Mendeley Data (2023). *Avocado Ripeness Stages Image Dataset.* <https://data.mendeley.com>

Microsoft Azure Documentation. (2024). *Store and retrieve image data using Azure Blob Storage.* <https://learn.microsoft.com/azure/storage/blobs/>

OpenAI. (2025). *ChatGPT (GPT-5)* [Large language model]. <https://chat.openai.com>

Prgomet, A. (2023, July 1). Djupinlärning [Video]. YouTube. https://www.youtube.com/watch?v=NiBMq7M72mY&list=PLgzaMbMPEHEy33r5tgph8TFR\_\_UHuutBN&index=2