Colab Notebook - Code and Output (Formatted)

```
>> from google.colab import files

--> # Upload the ZIP file manually
uploaded = files.upload()

>> import zipfile
>> import os

>> dataset_path = "/content/archive (4).zip"  # Replace with your uploaded filename
>> extract_path = "/content/extracted_data"

--> # Extract if not already extracted
>> if not os.path.exists(extract_path):
>>     os.makedirs(extract_path, exist_ok=True) >>     with
zipfile.ZipFile(dataset_path,     'r')     as     zip_ref:
zip_ref.extractall(extract_path) >>       print(f" Dataset
extracted to {extract_path}") else: >>     print(" Dataset
already   extracted")   -->   #     Dataset   extracted   to
/content/extracted_data

--> # Check for video devices
!ls /dev/video* --> # ls: cannot access '/dev/video*': No such

file or directory

--> # Install OpenCV
!pip install --upgrade opencv-python opencv-python-headless

--> # Requirement already satisfied: opencv-python in /usr/local/lib/python3.11/dist-packages
--> # Requirement already satisfied: opencv-python-headless in
/usr/local/lib/python3.11/dist-p --> # Requirement already satisfied: numpy>=1.21.2 in
/usr/local/lib/python3.11/dist-packages

>> import tensorflow as tf
>> from tensorflow.keras.models import Sequential >> from tensorflow.keras.layers
import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
>> from tensorflow.keras.preprocessing.image import ImageDataGenerator
>> from sklearn.metrics import confusion_matrix, classification_report
>> import numpy as np
>> import matplotlib.pyplot as plt
```

```
>> import seaborn as sns

--> # Data augmentation for training and testing >> train_datagen =
ImageDataGenerator(rescale=1./255, rotation_range=20, zoom_range=0.2, horizo
>> test_datagen = ImageDataGenerator(rescale=1./255)

--> # Set paths for training and testing data
>> train_data_dir = '/content/extracted_data'  # Replace with the actual path
>> test_data_dir = '/content/extracted_data'  # Replace with the actual path

--> # Load dataset into ImageDataGenerator
>> train_generator = train_datagen.flow_from_directory(train_data_dir, target_size=(150, 150),
>> test_generator = test_datagen.flow_from_directory(test_data_dir, target_size=(150, 150), ba

--> # Define a simple CNN model
>> model = Sequential([     Conv2D(32, (3, 3), activation='relu',
input_shape=(150, 150, 3)),
    MaxPooling2D((2, 2)),     Conv2D(64,
(3, 3), activation='relu'),
MaxPooling2D((2, 2)),
    Flatten(),
    Dense(128, activation='relu'),     Dense(train_generator.num_classes,
activation='softmax')  # Number of output classes
])

--> # Compile the model >> model.compile(optimizer='adam',
loss='categorical_crossentropy', metrics=['accuracy'])

--> # Train the model >> history = model.fit(train_generator, epochs=10,
validation_data=test_generator)

--> # Evaluate the model >> test_loss, test_acc =
model.evaluate(test_generator) >> print(f"Test
Accuracy: {test_acc:.2f}")

--> # Generate predictions
>> y_pred = model.predict(test_generator) >>
y_pred_classes = np.argmax(y_pred, axis=1)
y_true = test_generator.classes

--> # Compute confusion matrix >> conf_matrix =
confusion_matrix(y_true, y_pred_classes)
>> plt.figure(figsize=(8, 6)) >> sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
xticklabels=test_generator.class
```

```
>> plt.xlabel('Predicted')
>> plt.ylabel('Actual') >>
plt.title('Confusion Matrix')
>> plt.show()

--> # Print classification report
>> print("Classification Report:")
>> print(classification_report(y_true, y_pred_classes,
target_names=test_generator.class_indice



# import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix, precision_recall_curve, classification_report
import numpy as np

>> # Example Data: Replace with actual model outputs
y_true = np.array([0, 1, 0, 1, 1, 0, 1, 2, 2, 2, 0, 1, 2])  # True labels
y_pred = np.array([0, 1, 0, 1, 0, 0, 2, 2, 2, 1, 0, 1, 2])  # Predicted labels
classes = ["Biodegradable", "Non-Biodegradable", "Plastic"]  # Adjust based on your dataset



>> # Precision-Recall Curve**
precision, recall, _ = precision_recall_curve(y_true, y_pred, pos_label=1)
plt.figure(figsize=(6, 4))
plt.plot(recall, precision, marker='.', label="Precision-Recall Curve")
plt.title("Precision-Recall Curve")
plt.xlabel("Recall")
plt.ylabel("Precision")
plt.legend()
plt.grid(True)
plt.show()



>>  Accuracy & Loss Trends**
# Example data: Replace with actual model metrics
epochs = list(range(1, 11))  # Change based on the number of training epochs
train_acc = [0.70, 0.75, 0.80, 0.83, 0.86, 0.89, 0.91, 0.93, 0.94, 0.95]
val_acc = [0.68, 0.73, 0.78, 0.82, 0.85, 0.87, 0.89, 0.91, 0.92, 0.93]
train_loss = [0.8, 0.7, 0.6, 0.55, 0.5, 0.4, 0.35, 0.3, 0.25, 0.2]
val_loss = [0.9, 0.75, 0.65, 0.6, 0.55, 0.45, 0.4, 0.35, 0.3, 0.25]

plt.figure(figsize=(6, 4))
plt.plot(epochs, train_acc, label='Train Accuracy', marker='o')
```

```
plt.plot(epochs, val_acc, label='Validation Accuracy', marker='s')
plt.plot(epochs, train_loss, label='Train Loss', linestyle='--', marker='o')
plt.plot(epochs, val_loss, label='Validation Loss', linestyle='--', marker='s')
plt.title("Model Accuracy & Loss Trends")
plt.xlabel("Epochs")
plt.ylabel("Metrics")
plt.legend()
plt.grid(True)
plt.show()
```