**Image Classification Using Convolutional Neural Networks (CNNs)**

**1. Introduction**

Image classification is a fundamental problem in the field of computer vision, where the goal is to categorize images into predefined classes based on their visual content. It has applications in various domains such as autonomous vehicles, healthcare diagnostics, retail, and robotics.

With the advent of deep learning, Convolutional Neural Networks (CNNs) have become the most effective approach for image classification tasks. Unlike traditional machine learning methods that rely on hand-crafted features, CNNs automatically learn hierarchical features directly from raw image data.

This project focuses on classifying images of three common objects: **shoes, clips, and toothbrushes**. The classification system is designed to take an input image, preprocess it,augment it and predict the object class along with a confidence score. The model architecture is based on **MobileNetV2**, a lightweight CNN known for its efficiency and suitability for smaller datasets and limited computational resources.

**2. Objectives**

The primary objectives of this project are:

1. **Dataset Preparation and Preprocessing:**
   * Collect and organize images for each object class.
   * Convert all images to a uniform format (JPEG) and resize them for consistent input to the neural network.
   * Encode categorical labels into numerical form suitable for training.
2. **Model Development:**
   * Implement a CNN using MobileNetV2 as a feature extractor.
   * Build a sequential model with additional layers for global pooling, dropout, and dense classification.
3. **Model Training and Validation:**
   * Train the model on the preprocessed training dataset.
   * Validate the model using a separate validation set to monitor overfitting and generalization.
4. **Fine-Tuning:**
   * Unfreeze select layers of the pre-trained network to fine-tune the model and improve classification accuracy.
5. **Prediction Pipeline:**
   * Deploy a system that can take a new image as input, process it, and output the predicted class along with a confidence score.

**3. Methodology**

**3.1 Data Collection and Organization**

The images were collected and stored in separate folders for each class:

* /TRAINING/shoes
* /TRAINING/clips
* /TRAINING/toothbrush

All images were converted to **JPEG format** to ensure compatibility with the image loading functions. The dataset consisted of multiple images per class to provide the model with sufficient examples for training.

**3.2 Image Preprocessing**

Each image was resized to **128x128 pixels**. This step ensures that all images have a uniform input size for the CNN. Pixel values were normalized by dividing by 255 to scale them to a range of 0–1, which improves training stability. Labels were encoded using **LabelEncoder** to convert the textual class names into numerical values.

**3.3 Train-Validation Split**

The dataset was divided into **training (80%)** and **validation (20%)** sets. Stratified sampling was applied to maintain the same class distribution in both sets. This helps in accurate evaluation of the model’s performance on unseen data.

**3.4 Model Architecture**

The CNN model is based on **MobileNetV2**, a deep learning architecture optimized for speed and efficiency. Key components include:

* **MobileNetV2 base model:** Used as a feature extractor with convolutional layers pre-trained (or initialized randomly) for efficient feature learning.
* **GlobalAveragePooling2D:** Reduces the spatial dimensions of feature maps and helps in summarizing feature information.
* **Dropout layer:** Applied to prevent overfitting by randomly disabling neurons during training.
* **Dense layer with softmax activation:** Outputs probabilities for each class.

**3.5 Training**

The model was compiled with:

* **Optimizer:** Adam, which adapts the learning rate during training.
* **Loss function:** Sparse categorical cross-entropy, suitable for multi-class classification.
* **Metrics:** Accuracy to monitor model performance.

The model was trained for **15 epochs** initially.

**3.6 Fine-Tuning**

After initial training, some layers of the MobileNetV2 base were unfrozen (keeping the early layers frozen) to allow the model to adjust pre-learned features for the specific dataset. Fine-tuning was done for **10 additional epochs** with a smaller learning rate (1e-5) to avoid overfitting.

**4. Packages and Libraries Used**

* **Python 3.x** – Primary programming language.
* **TensorFlow / Keras** – For building and training CNN models.
* **NumPy** – Numerical operations and array handling.
* **scikit-learn** – For train-test splitting and label encoding.
* **OS** – To handle file paths and directory traversal.
* **Matplotlib (optional)** – For visualizing images and training progress.

## What I Have Done in This Project

In this project, I developed a **Convolutional Neural Network (CNN)** model to classify images into three categories: **shoes, clips, and toothbrush**. The entire process involved careful preparation of the dataset, building a deep learning model, applying data augmentation, and testing the predictions. Below is a detailed explanation of the work I carried out:

### 1. Dataset Preparation

* I collected images of **shoes, clips, and toothbrushes** and organized them into separate folders.
* Each folder represented one class so that the model could learn to recognize unique features of each object.
* The images were preprocessed by resizing them into a uniform size of **128 × 128 pixels**, making them suitable for input to the CNN model.
* The pixel values were normalized to a scale of **0 to 1**, which speeds up training and improves performance.

### 2. Data Splitting

* I divided the dataset into **training and validation sets**.
* Training data was used to train the model, while the validation data was used to evaluate the performance during training and avoid overfitting.
* I used a **stratified split** to ensure each class (shoes, clips, toothbrush) was fairly represented in both sets.

### 3. Data Augmentation

* To make the model more robust and improve its ability to generalize, I applied **data augmentation techniques**.
* The augmentations included:
  + **Rotation** of images by small angles.
  + **Shifting** images horizontally and vertically.
  + **Zooming** in and out.
  + **Shearing** transformations.
  + **Flipping** images horizontally.
* These transformations helped increase the variety of the dataset without the need to collect more images.
* I also saved some of the augmented images into a separate folder so they could be visually inspected.

### 4. Building the CNN Model

* I used **MobileNetV2** as the base architecture, which is a powerful CNN model.
* The base model was initially kept **frozen** to train only the top layers for my dataset.
* I added layers on top of it including:
  + A **Global Average Pooling layer** to reduce dimensions.
  + A **Dropout layer** to avoid overfitting.
  + A **Dense layer with softmax activation** to classify the images into three categories.
* Later, I performed **fine-tuning** by unfreezing some of the deeper layers of MobileNetV2 and retraining them at a lower learning rate to improve accuracy.

### 5. Training the Model

* I trained the model using the **Adam optimizer** with categorical cross-entropy loss.
* The training process involved feeding both **original and augmented images**.
* During training, I monitored the **accuracy and loss values** on both the training and validation sets to ensure good learning.

### 6. Prediction on Test Images

* After training, I tested the model on **new unseen images**.
* The model produced predictions with a **class label** (shoes, clips, toothbrush) and a **confidence score**.
* For example, when I tested the model on a toothbrush image, it correctly predicted the label as **“toothbrush” with 34% confidence**.

**6.SOURCE CODE**

import os

import numpy as np

import tensorflow as tf

from tensorflow.keras import layers, models

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

from sklearn.preprocessing import LabelEncoder

from tensorflow.keras.preprocessing.image import ImageDataGenerator

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

# 1. Set paths

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

clips\_path = "/Users/sincymol/Downloads/CNN\_PROJECT/TRAINING/clip"

toothbrush\_path = "/Users/sincymol/Downloads/CNN\_PROJECT/TRAINING/toothbrush"

shoes\_path = "/Users/sincymol/Downloads/CNN\_PROJECT/TRAINING/shoes123"

class\_paths = {

"shoes": shoes\_path,

"clips": clips\_path,

"toothbrush": toothbrush\_path

}

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

# 2. Load images and labels

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

IMG\_SIZE = (128, 128)

images, labels = [], []

for label, folder in class\_paths.items():

if not os.path.exists(folder):

print(f"⚠️ Folder not found: {folder}, skipping this class")

continue

for file in os.listdir(folder):

if file.lower().endswith((".jpg", ".jpeg", ".png")):

try:

img\_path = os.path.join(folder, file)

img = tf.keras.utils.load\_img(img\_path, target\_size=IMG\_SIZE)

img\_array = tf.keras.utils.img\_to\_array(img)

images.append(img\_array)

labels.append(label)

except Exception as e:

print(f"Skipped {file}: {e}")

images = np.array(images)

labels = np.array(labels)

if len(images) == 0:

raise ValueError("No images loaded! Check your folder paths and image files.")

# Encode labels

le = LabelEncoder()

labels\_encoded = le.fit\_transform(labels)

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

# 3. Train / Validation split

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

X\_train, X\_val, y\_train, y\_val = train\_test\_split(

images, labels\_encoded, test\_size=0.2, random\_state=42, stratify=labels\_encoded

)

# Normalize images

X\_train = X\_train / 255.0

X\_val = X\_val / 255.0

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

# 4. Data Augmentation

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

datagen = ImageDataGenerator(

rotation\_range=25,

width\_shift\_range=0.1,

height\_shift\_range=0.1,

shear\_range=0.1,

zoom\_range=0.2,

horizontal\_flip=True,

fill\_mode="nearest"

)

datagen.fit(X\_train)

# Save some augmented images

augmented\_output = "augmented\_images"

os.makedirs(augmented\_output, exist\_ok=True)

save\_gen = datagen.flow(

X\_train, y\_train,

batch\_size=10,

save\_to\_dir=augmented\_output,

save\_prefix="aug",

save\_format="jpg"

)

for i in range(30): # Save 30 augmented images

next(save\_gen)

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

# 5. Build CNN model (MobileNetV2)

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

base\_model = tf.keras.applications.MobileNetV2(

input\_shape=IMG\_SIZE + (3,),

include\_top=False,

weights=None

)

base\_model.trainable = False

model = models.Sequential([

base\_model,

layers.GlobalAveragePooling2D(),

layers.Dropout(0.3),

layers.Dense(len(class\_paths), activation="softmax")

])

model.compile(

optimizer="adam",

loss="sparse\_categorical\_crossentropy",

metrics=["accuracy"]

)

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

# 6. Train the model

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

history = model.fit(

datagen.flow(X\_train, y\_train, batch\_size=8),

validation\_data=(X\_val, y\_val),

epochs=15,

verbose=1

)

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

# 7. Fine-tuning (optional)

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

base\_model.trainable = True

for layer in base\_model.layers[:100]:

layer.trainable = False

model.compile(

optimizer=tf.keras.optimizers.Adam(1e-5),

loss="sparse\_categorical\_crossentropy",

metrics=["accuracy"]

)

history\_ft = model.fit(

datagen.flow(X\_train, y\_train, batch\_size=8),

validation\_data=(X\_val, y\_val),

epochs=10,

verbose=1

)

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

# 8. Prediction Function

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

def predict\_image(img\_path):

if not os.path.exists(img\_path):

print(f"⚠️ File not found: {img\_path}")

return

img = tf.keras.utils.load\_img(img\_path, target\_size=IMG\_SIZE)

img\_array = tf.keras.utils.img\_to\_array(img)

img\_array = np.expand\_dims(img\_array / 255.0, axis=0)

preds = model.predict(img\_array)

class\_idx = np.argmax(preds)

confidence = preds[0][class\_idx]

class\_name = le.inverse\_transform([class\_idx])[0]

print(f"Prediction: {class\_name}, Confidence: {confidence:.2f}")

# Example usage

predict\_image("/Users/sincymol/Downloads/CNN\_PROJECT/TESTING/IMG\_4890 Medium.jpeg")

**7.IMAGE UPLOADED**



**8.OUTPUT**

**Python 3.13.3 (v3.13.3:6280bb54784, Apr 8 2025, 10:47:54) [Clang 15.0.0 (clang-1500.3.9.4)] on darwin**

**Enter "help" below or click "Help" above for more information.**

**========== RESTART: /Users/sincymol/Downloads/CNN\_PROJECT/try idle5.py =========**

**Warning (from warnings module):**

**File "/Library/Frameworks/Python.framework/Versions/3.13/lib/python3.13/site-packages/keras/src/trainers/data\_adapters/py\_dataset\_adapter.py", line 121**

**self.\_warn\_if\_super\_not\_called()**

**UserWarning: Your `PyDataset` class should call `super().\_\_init\_\_(\*\*kwargs)` in its constructor. `\*\*kwargs` can include `workers`, `use\_multiprocessing`, `max\_queue\_size`. Do not pass these arguments to `fit()`, as they will be ignored.**

**Epoch 1/15**

**[1m1/9[0m [32m━━[0m[37m━━━━━━━━━━━━━━━━━━[0m [1m11s[0m 1s/step - accuracy: 0.5000 - loss: 1.0986**

**[1m3/9[0m [32m━━━━━━[0m[37m━━━━━━━━━━━━━━[0m [1m0s[0m 28ms/step - accuracy: 0.5000 - loss: 1.0986**

**[1m5/9[0m [32m━━━━━━━━━━━[0m[37m━━━━━━━━━[0m [1m0s[0m 30ms/step - accuracy: 0.4563 - loss: 1.0987**

**[1m7/9[0m [32m━━━━━━━━━━━━━━━[0m[37m━━━━━[0m [1m0s[0m 31ms/step - accuracy: 0.4279 - loss: 1.0987**

**[1m9/9[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m 32ms/step - accuracy: 0.4081 - loss: 1.0987**

**[1m9/9[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m2s[0m 98ms/step - accuracy: 0.3333 - loss: 1.0989 - val\_accuracy: 0.3333 - val\_loss: 1.0986**

**Epoch 2/15**

**[1m1/9[0m [32m━━[0m[37m━━━━━━━━━━━━━━━━━━[0m [1m0s[0m 57ms/step - accuracy: 0.3750 - loss: 1.0983**

**[1m3/9[0m [32m━━━━━━[0m[37m━━━━━━━━━━━━━━[0m [1m0s[0m 29ms/step - accuracy: 0.3403 - loss: 1.0986**

**[1m5/9[0m [32m━━━━━━━━━━━[0m[37m━━━━━━━━━[0m [1m0s[0m 31ms/step - accuracy: 0.3592 - loss: 1.0985**

**[1m7/9[0m [32m━━━━━━━━━━━━━━━[0m[37m━━━━━[0m [1m0s[0m 32ms/step - accuracy: 0.3586 - loss: 1.0985**

**[1m9/9[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m 32ms/step - accuracy: 0.3524 - loss: 1.0986**

**[1m9/9[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m 42ms/step - accuracy: 0.3333 - loss: 1.0987 - val\_accuracy: 0.3333 - val\_loss: 1.0986**

**Epoch 3/15**

**[1m1/9[0m [32m━━[0m[37m━━━━━━━━━━━━━━━━━━[0m [1m0s[0m 57ms/step - accuracy: 0.5000 - loss: 1.0979**

**[1m3/9[0m [32m━━━━━━[0m[37m━━━━━━━━━━━━━━[0m [1m0s[0m 37ms/step - accuracy: 0.4306 - loss: 1.0981**

**[1m5/9[0m [32m━━━━━━━━━━━[0m[37m━━━━━━━━━[0m [1m0s[0m 35ms/step - accuracy: 0.4096 - loss: 1.0982**

**[1m7/9[0m [32m━━━━━━━━━━━━━━━[0m[37m━━━━━[0m [1m0s[0m 34ms/step - accuracy: 0.3857 - loss: 1.0984**

**[1m9/9[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m 34ms/step - accuracy: 0.3735 - loss: 1.0984**

**[1m9/9[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m 44ms/step - accuracy: 0.3333 - loss: 1.0987 - val\_accuracy: 0.3333 - val\_loss: 1.0986**

**Epoch 4/15**

**[1m1/9[0m [32m━━[0m[37m━━━━━━━━━━━━━━━━━━[0m [1m0s[0m 57ms/step - accuracy: 0.0000e+00 - loss: 1.1002**

**[1m2/9[0m [32m━━━━[0m[37m━━━━━━━━━━━━━━━━[0m [1m0s[0m 50ms/step - accuracy: 0.0625 - loss: 1.0998**

**[1m4/9[0m [32m━━━━━━━━[0m[37m━━━━━━━━━━━━[0m [1m0s[0m 41ms/step - accuracy: 0.1484 - loss: 1.0994**

**[1m6/9[0m [32m━━━━━━━━━━━━━[0m[37m━━━━━━━[0m [1m0s[0m 38ms/step - accuracy: 0.1816 - loss: 1.0993**

**[1m8/9[0m [32m━━━━━━━━━━━━━━━━━[0m[37m━━━[0m [1m0s[0m 37ms/step - accuracy: 0.2068 - loss: 1.0992**

**[1m9/9[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m 45ms/step - accuracy: 0.3333 - loss: 1.0988 - val\_accuracy: 0.3333 - val\_loss: 1.0986**

**Epoch 5/15**

**[1m1/9[0m [32m━━[0m[37m━━━━━━━━━━━━━━━━━━[0m [1m0s[0m 57ms/step - accuracy: 0.3750 - loss: 1.0988**

**[1m2/9[0m [32m━━━━[0m[37m━━━━━━━━━━━━━━━━[0m [1m0s[0m 50ms/step - accuracy: 0.3750 - loss: 1.0987**

**[1m3/9[0m [32m━━━━━━[0m[37m━━━━━━━━━━━━━━[0m [1m0s[0m 50ms/step - accuracy: 0.3750 - loss: 1.0987**

**[1m5/9[0m [32m━━━━━━━━━━━[0m[37m━━━━━━━━━[0m [1m0s[0m 44ms/step - accuracy: 0.3638 - loss: 1.0987**

**[1m7/9[0m [32m━━━━━━━━━━━━━━━[0m[37m━━━━━[0m [1m0s[0m 40ms/step - accuracy: 0.3585 - loss: 1.0986**

**[1m9/9[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m 38ms/step - accuracy: 0.3523 - loss: 1.0986**

**[1m9/9[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m 50ms/step - accuracy: 0.3333 - loss: 1.0986 - val\_accuracy: 0.3333 - val\_loss: 1.0986**

**Epoch 6/15**

**[1m1/9[0m [32m━━[0m[37m━━━━━━━━━━━━━━━━━━[0m [1m0s[0m 58ms/step - accuracy: 0.3750 - loss: 1.0985**

**[1m3/9[0m [32m━━━━━━[0m[37m━━━━━━━━━━━━━━[0m [1m0s[0m 37ms/step - accuracy: 0.3194 - loss: 1.0987**

**[1m5/9[0m [32m━━━━━━━━━━━[0m[37m━━━━━━━━━[0m [1m0s[0m 35ms/step - accuracy: 0.2967 - loss: 1.0988**

**[1m7/9[0m [32m━━━━━━━━━━━━━━━[0m[37m━━━━━[0m [1m0s[0m 35ms/step - accuracy: 0.2944 - loss: 1.0988**

**[1m9/9[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m 34ms/step - accuracy: 0.3025 - loss: 1.0987**

**[1m9/9[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m 44ms/step - accuracy: 0.3333 - loss: 1.0987 - val\_accuracy: 0.3333 - val\_loss: 1.0986**

**Epoch 7/15**

**[1m1/9[0m [32m━━[0m[37m━━━━━━━━━━━━━━━━━━[0m [1m0s[0m 58ms/step - accuracy: 0.2500 - loss: 1.0988**

**[1m3/9[0m [32m━━━━━━[0m[37m━━━━━━━━━━━━━━[0m [1m0s[0m 37ms/step - accuracy: 0.2222 - loss: 1.0989**

**[1m5/9[0m [32m━━━━━━━━━━━[0m[37m━━━━━━━━━[0m [1m0s[0m 35ms/step - accuracy: 0.2483 - loss: 1.0988**

**[1m7/9[0m [32m━━━━━━━━━━━━━━━[0m[37m━━━━━[0m [1m0s[0m 34ms/step - accuracy: 0.2739 - loss: 1.0988**

**[1m9/9[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m 34ms/step - accuracy: 0.2865 - loss: 1.0987**

**[1m9/9[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m 44ms/step - accuracy: 0.3333 - loss: 1.0986 - val\_accuracy: 0.3333 - val\_loss: 1.0986**

**Epoch 8/15**

**[1m1/9[0m [32m━━[0m[37m━━━━━━━━━━━━━━━━━━[0m [1m0s[0m 57ms/step - accuracy: 0.3750 - loss: 1.0986**

**[1m3/9[0m [32m━━━━━━[0m[37m━━━━━━━━━━━━━━[0m [1m0s[0m 37ms/step - accuracy: 0.3750 - loss: 1.0985**

**[1m5/9[0m [32m━━━━━━━━━━━[0m[37m━━━━━━━━━[0m [1m0s[0m 35ms/step - accuracy: 0.3825 - loss: 1.0986**

**[1m7/9[0m [32m━━━━━━━━━━━━━━━[0m[37m━━━━━[0m [1m0s[0m 35ms/step - accuracy: 0.3697 - loss: 1.0986**

**[1m9/9[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m 34ms/step - accuracy: 0.3576 - loss: 1.0986**

**[1m9/9[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m 44ms/step - accuracy: 0.3333 - loss: 1.0987 - val\_accuracy: 0.3333 - val\_loss: 1.0986**

**Epoch 9/15**

**[1m1/9[0m [32m━━[0m[37m━━━━━━━━━━━━━━━━━━[0m [1m0s[0m 57ms/step - accuracy: 0.2500 - loss: 1.0989**

**[1m3/9[0m [32m━━━━━━[0m[37m━━━━━━━━━━━━━━[0m [1m0s[0m 37ms/step - accuracy: 0.2361 - loss: 1.0989**

**[1m5/9[0m [32m━━━━━━━━━━━[0m[37m━━━━━━━━━[0m [1m0s[0m 35ms/step - accuracy: 0.2304 - loss: 1.0989**

**[1m7/9[0m [32m━━━━━━━━━━━━━━━[0m[37m━━━━━[0m [1m0s[0m 34ms/step - accuracy: 0.2547 - loss: 1.0989**

**[1m9/9[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m 34ms/step - accuracy: 0.2768 - loss: 1.0988**

**[1m9/9[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m 44ms/step - accuracy: 0.3333 - loss: 1.0987 - val\_accuracy: 0.3333 - val\_loss: 1.0986**

**Epoch 10/15**

**[1m1/9[0m [32m━━[0m[37m━━━━━━━━━━━━━━━━━━[0m [1m0s[0m 57ms/step - accuracy: 0.1250 - loss: 1.0991**

**[1m2/9[0m [32m━━━━[0m[37m━━━━━━━━━━━━━━━━[0m [1m0s[0m 51ms/step - accuracy: 0.1875 - loss: 1.0989**

**[1m4/9[0m [32m━━━━━━━━[0m[37m━━━━━━━━━━━━[0m [1m0s[0m 41ms/step - accuracy: 0.2630 - loss: 1.0988**

**[1m6/9[0m [32m━━━━━━━━━━━━━[0m[37m━━━━━━━[0m [1m0s[0m 38ms/step - accuracy: 0.2781 - loss: 1.0987**

**[1m8/9[0m [32m━━━━━━━━━━━━━━━━━[0m[37m━━━[0m [1m0s[0m 37ms/step - accuracy: 0.2920 - loss: 1.0987**

**[1m9/9[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m 45ms/step - accuracy: 0.3333 - loss: 1.0986 - val\_accuracy: 0.3333 - val\_loss: 1.0986**

**Epoch 11/15**

**[1m1/9[0m [32m━━[0m[37m━━━━━━━━━━━━━━━━━━[0m [1m0s[0m 57ms/step - accuracy: 0.1250 - loss: 1.0990**

**[1m3/9[0m [32m━━━━━━[0m[37m━━━━━━━━━━━━━━[0m [1m0s[0m 29ms/step - accuracy: 0.2153 - loss: 1.0988**

**[1m5/9[0m [32m━━━━━━━━━━━[0m[37m━━━━━━━━━[0m [1m0s[0m 31ms/step - accuracy: 0.2579 - loss: 1.0987**

**[1m7/9[0m [32m━━━━━━━━━━━━━━━[0m[37m━━━━━[0m [1m0s[0m 32ms/step - accuracy: 0.2718 - loss: 1.0987**

**[1m9/9[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m 32ms/step - accuracy: 0.2849 - loss: 1.0987**

**[1m9/9[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m 42ms/step - accuracy: 0.3333 - loss: 1.0987 - val\_accuracy: 0.3333 - val\_loss: 1.0986**

**Epoch 12/15**

**[1m1/9[0m [32m━━[0m[37m━━━━━━━━━━━━━━━━━━[0m [1m0s[0m 57ms/step - accuracy: 0.3750 - loss: 1.0982**

**[1m3/9[0m [32m━━━━━━[0m[37m━━━━━━━━━━━━━━[0m [1m0s[0m 37ms/step - accuracy: 0.4097 - loss: 1.0982**

**[1m5/9[0m [32m━━━━━━━━━━━[0m[37m━━━━━━━━━[0m [1m0s[0m 35ms/step - accuracy: 0.3733 - loss: 1.0984**

**[1m7/9[0m [32m━━━━━━━━━━━━━━━[0m[37m━━━━━[0m [1m0s[0m 35ms/step - accuracy: 0.3738 - loss: 1.0984**

**[1m9/9[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m 34ms/step - accuracy: 0.3660 - loss: 1.0985**

**[1m9/9[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m 44ms/step - accuracy: 0.3333 - loss: 1.0987 - val\_accuracy: 0.3333 - val\_loss: 1.0986**

**Epoch 13/15**

**[1m1/9[0m [32m━━[0m[37m━━━━━━━━━━━━━━━━━━[0m [1m0s[0m 58ms/step - accuracy: 0.3750 - loss: 1.0987**

**[1m3/9[0m [32m━━━━━━[0m[37m━━━━━━━━━━━━━━[0m [1m0s[0m 37ms/step - accuracy: 0.3681 - loss: 1.0984**

**[1m5/9[0m [32m━━━━━━━━━━━[0m[37m━━━━━━━━━[0m [1m0s[0m 35ms/step - accuracy: 0.3658 - loss: 1.0984**

**[1m7/9[0m [32m━━━━━━━━━━━━━━━[0m[37m━━━━━[0m [1m0s[0m 35ms/step - accuracy: 0.3680 - loss: 1.0984**

**[1m9/9[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m 34ms/step - accuracy: 0.3649 - loss: 1.0984**

**[1m9/9[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m 44ms/step - accuracy: 0.3333 - loss: 1.0987 - val\_accuracy: 0.3333 - val\_loss: 1.0986**

**Epoch 14/15**

**[1m1/9[0m [32m━━[0m[37m━━━━━━━━━━━━━━━━━━[0m [1m0s[0m 57ms/step - accuracy: 0.1250 - loss: 1.0992**

**[1m2/9[0m [32m━━━━[0m[37m━━━━━━━━━━━━━━━━[0m [1m0s[0m 51ms/step - accuracy: 0.2188 - loss: 1.0989**

**[1m4/9[0m [32m━━━━━━━━[0m[37m━━━━━━━━━━━━[0m [1m0s[0m 42ms/step - accuracy: 0.2344 - loss: 1.0989**

**[1m6/9[0m [32m━━━━━━━━━━━━━[0m[37m━━━━━━━[0m [1m0s[0m 38ms/step - accuracy: 0.2396 - loss: 1.0989**

**[1m8/9[0m [32m━━━━━━━━━━━━━━━━━[0m[37m━━━[0m [1m0s[0m 37ms/step - accuracy: 0.2586 - loss: 1.0989**

**[1m9/9[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m 40ms/step - accuracy: 0.2669 - loss: 1.0988**

**[1m9/9[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m 50ms/step - accuracy: 0.3333 - loss: 1.0987 - val\_accuracy: 0.3333 - val\_loss: 1.0986**

**Epoch 15/15**

**[1m1/9[0m [32m━━[0m[37m━━━━━━━━━━━━━━━━━━[0m [1m0s[0m 58ms/step - accuracy: 0.2500 - loss: 1.0985**

**[1m3/9[0m [32m━━━━━━[0m[37m━━━━━━━━━━━━━━[0m [1m0s[0m 37ms/step - accuracy: 0.3125 - loss: 1.0985**

**[1m5/9[0m [32m━━━━━━━━━━━[0m[37m━━━━━━━━━[0m [1m0s[0m 35ms/step - accuracy: 0.3375 - loss: 1.0985**

**[1m7/9[0m [32m━━━━━━━━━━━━━━━[0m[37m━━━━━[0m [1m0s[0m 35ms/step - accuracy: 0.3376 - loss: 1.0985**

**[1m9/9[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m 34ms/step - accuracy: 0.3361 - loss: 1.0986**

**[1m9/9[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m 44ms/step - accuracy: 0.3333 - loss: 1.0986 - val\_accuracy: 0.3333 - val\_loss: 1.0986**

**Epoch 1/10**

**[1m1/9[0m [32m━━[0m[37m━━━━━━━━━━━━━━━━━━[0m [1m29s[0m 4s/step - accuracy: 0.1250 - loss: 1.3865**

**[1m2/9[0m [32m━━━━[0m[37m━━━━━━━━━━━━━━━━[0m [1m0s[0m 75ms/step - accuracy: 0.1562 - loss: 1.3204**

**[1m3/9[0m [32m━━━━━━[0m[37m━━━━━━━━━━━━━━[0m [1m0s[0m 71ms/step - accuracy: 0.1736 - loss: 1.2772**

**[1m4/9[0m [32m━━━━━━━━[0m[37m━━━━━━━━━━━━[0m [1m0s[0m 69ms/step - accuracy: 0.2005 - loss: 1.2458**

**[1m5/9[0m [32m━━━━━━━━━━━[0m[37m━━━━━━━━━[0m [1m0s[0m 69ms/step - accuracy: 0.2254 - loss: 1.2198**

**[1m6/9[0m [32m━━━━━━━━━━━━━[0m[37m━━━━━━━[0m [1m0s[0m 68ms/step - accuracy: 0.2469 - loss: 1.2029**

**[1m7/9[0m [32m━━━━━━━━━━━━━━━[0m[37m━━━━━[0m [1m0s[0m 71ms/step - accuracy: 0.2652 - loss: 1.1881**

**[1m8/9[0m [32m━━━━━━━━━━━━━━━━━[0m[37m━━━[0m [1m0s[0m 73ms/step - accuracy: 0.2828 - loss: 1.1734**

**[1m9/9[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m 74ms/step - accuracy: 0.2961 - loss: 1.1609**

**[1m9/9[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m5s[0m 144ms/step - accuracy: 0.4028 - loss: 1.0614 - val\_accuracy: 0.3333 - val\_loss: 1.0986**

**Epoch 2/10**

**[1m1/9[0m [32m━━[0m[37m━━━━━━━━━━━━━━━━━━[0m [1m0s[0m 75ms/step - accuracy: 0.3750 - loss: 1.0089**

**[1m2/9[0m [32m━━━━[0m[37m━━━━━━━━━━━━━━━━[0m [1m0s[0m 117ms/step - accuracy: 0.3438 - loss: 1.0160**

**[1m3/9[0m [32m━━━━━━[0m[37m━━━━━━━━━━━━━━[0m [1m0s[0m 91ms/step - accuracy: 0.3264 - loss: 1.0272**

**[1m4/9[0m [32m━━━━━━━━[0m[37m━━━━━━━━━━━━[0m [1m0s[0m 83ms/step - accuracy: 0.3151 - loss: 1.0368**

**[1m5/9[0m [32m━━━━━━━━━━━[0m[37m━━━━━━━━━[0m [1m0s[0m 79ms/step - accuracy: 0.3071 - loss: 1.0440**

**[1m6/9[0m [32m━━━━━━━━━━━━━[0m[37m━━━━━━━[0m [1m0s[0m 77ms/step - accuracy: 0.3080 - loss: 1.0481**

**[1m7/9[0m [32m━━━━━━━━━━━━━━━[0m[37m━━━━━[0m [1m0s[0m 75ms/step - accuracy: 0.3074 - loss: 1.0577**

**[1m8/9[0m [32m━━━━━━━━━━━━━━━━━[0m[37m━━━[0m [1m0s[0m 76ms/step - accuracy: 0.3060 - loss: 1.0681**

**[1m9/9[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m 77ms/step - accuracy: 0.3044 - loss: 1.0754**

**[1m9/9[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m 92ms/step - accuracy: 0.2917 - loss: 1.1332 - val\_accuracy: 0.3333 - val\_loss: 1.0986**

**Epoch 3/10**

**[1m1/9[0m [32m━━[0m[37m━━━━━━━━━━━━━━━━━━[0m [1m0s[0m 76ms/step - accuracy: 0.2500 - loss: 1.1148**

**[1m2/9[0m [32m━━━━[0m[37m━━━━━━━━━━━━━━━━[0m [1m0s[0m 115ms/step - accuracy: 0.3438 - loss: 1.0523**

**[1m3/9[0m [32m━━━━━━[0m[37m━━━━━━━━━━━━━━[0m [1m0s[0m 91ms/step - accuracy: 0.3958 - loss: 1.0122**

**[1m4/9[0m [32m━━━━━━━━[0m[37m━━━━━━━━━━━━[0m [1m0s[0m 83ms/step - accuracy: 0.3984 - loss: 1.0087**

**[1m5/9[0m [32m━━━━━━━━━━━[0m[37m━━━━━━━━━[0m [1m0s[0m 79ms/step - accuracy: 0.4087 - loss: 1.0027**

**[1m6/9[0m [32m━━━━━━━━━━━━━[0m[37m━━━━━━━[0m [1m0s[0m 77ms/step - accuracy: 0.4240 - loss: 0.9999**

**[1m7/9[0m [32m━━━━━━━━━━━━━━━[0m[37m━━━━━[0m [1m0s[0m 75ms/step - accuracy: 0.4374 - loss: 0.9956**

**[1m8/9[0m [32m━━━━━━━━━━━━━━━━━[0m[37m━━━[0m [1m0s[0m 76ms/step - accuracy: 0.4472 - loss: 0.9929**

**[1m9/9[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m 77ms/step - accuracy: 0.4577 - loss: 0.9909**

**[1m9/9[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m 92ms/step - accuracy: 0.5417 - loss: 0.9752 - val\_accuracy: 0.3333 - val\_loss: 1.0986**

**Epoch 4/10**

**[1m1/9[0m [32m━━[0m[37m━━━━━━━━━━━━━━━━━━[0m [1m0s[0m 75ms/step - accuracy: 0.5000 - loss: 0.9028**

**[1m2/9[0m [32m━━━━[0m[37m━━━━━━━━━━━━━━━━[0m [1m0s[0m 115ms/step - accuracy: 0.4688 - loss: 0.9100**

**[1m3/9[0m [32m━━━━━━[0m[37m━━━━━━━━━━━━━━[0m [1m0s[0m 91ms/step - accuracy: 0.4653 - loss: 0.9308**

**[1m4/9[0m [32m━━━━━━━━[0m[37m━━━━━━━━━━━━[0m [1m0s[0m 84ms/step - accuracy: 0.4583 - loss: 0.9383**

**[1m5/9[0m [32m━━━━━━━━━━━[0m[37m━━━━━━━━━[0m [1m0s[0m 79ms/step - accuracy: 0.4617 - loss: 0.9398**

**[1m6/9[0m [32m━━━━━━━━━━━━━[0m[37m━━━━━━━[0m [1m0s[0m 77ms/step - accuracy: 0.4750 - loss: 0.9337**

**[1m7/9[0m [32m━━━━━━━━━━━━━━━[0m[37m━━━━━[0m [1m0s[0m 75ms/step - accuracy: 0.4939 - loss: 0.9257**

**[1m8/9[0m [32m━━━━━━━━━━━━━━━━━[0m[37m━━━[0m [1m0s[0m 76ms/step - accuracy: 0.5064 - loss: 0.9233**

**[1m9/9[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m 77ms/step - accuracy: 0.5165 - loss: 0.9222**

**[1m9/9[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m 92ms/step - accuracy: 0.5972 - loss: 0.9141 - val\_accuracy: 0.3333 - val\_loss: 1.0986**

**Epoch 5/10**

**[1m1/9[0m [32m━━[0m[37m━━━━━━━━━━━━━━━━━━[0m [1m0s[0m 76ms/step - accuracy: 0.7500 - loss: 0.8983**

**[1m2/9[0m [32m━━━━[0m[37m━━━━━━━━━━━━━━━━[0m [1m0s[0m 116ms/step - accuracy: 0.7500 - loss: 0.8727**

**[1m3/9[0m [32m━━━━━━[0m[37m━━━━━━━━━━━━━━[0m [1m0s[0m 91ms/step - accuracy: 0.7361 - loss: 0.8512**

**[1m4/9[0m [32m━━━━━━━━[0m[37m━━━━━━━━━━━━[0m [1m0s[0m 83ms/step - accuracy: 0.7396 - loss: 0.8353**

**[1m5/9[0m [32m━━━━━━━━━━━[0m[37m━━━━━━━━━[0m [1m0s[0m 79ms/step - accuracy: 0.7317 - loss: 0.8323**

**[1m6/9[0m [32m━━━━━━━━━━━━━[0m[37m━━━━━━━[0m [1m0s[0m 76ms/step - accuracy: 0.7174 - loss: 0.8338**

**[1m7/9[0m [32m━━━━━━━━━━━━━━━[0m[37m━━━━━[0m [1m0s[0m 75ms/step - accuracy: 0.7067 - loss: 0.8338**

**[1m8/9[0m [32m━━━━━━━━━━━━━━━━━[0m[37m━━━[0m [1m0s[0m 76ms/step - accuracy: 0.6985 - loss: 0.8340**

**[1m9/9[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m 77ms/step - accuracy: 0.6903 - loss: 0.8352**

**[1m9/9[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m 92ms/step - accuracy: 0.6250 - loss: 0.8453 - val\_accuracy: 0.3333 - val\_loss: 1.0986**

**Epoch 6/10**

**[1m1/9[0m [32m━━[0m[37m━━━━━━━━━━━━━━━━━━[0m [1m0s[0m 78ms/step - accuracy: 0.5000 - loss: 0.8974**

**[1m2/9[0m [32m━━━━[0m[37m━━━━━━━━━━━━━━━━[0m [1m0s[0m 115ms/step - accuracy: 0.5312 - loss: 0.9093**

**[1m3/9[0m [32m━━━━━━[0m[37m━━━━━━━━━━━━━━[0m [1m0s[0m 91ms/step - accuracy: 0.5347 - loss: 0.9356**

**[1m4/9[0m [32m━━━━━━━━[0m[37m━━━━━━━━━━━━[0m [1m0s[0m 82ms/step - accuracy: 0.5417 - loss: 0.9335**

**[1m5/9[0m [32m━━━━━━━━━━━[0m[37m━━━━━━━━━[0m [1m0s[0m 79ms/step - accuracy: 0.5483 - loss: 0.9247**

**[1m6/9[0m [32m━━━━━━━━━━━━━[0m[37m━━━━━━━[0m [1m0s[0m 77ms/step - accuracy: 0.5576 - loss: 0.9133**

**[1m7/9[0m [32m━━━━━━━━━━━━━━━[0m[37m━━━━━[0m [1m0s[0m 75ms/step - accuracy: 0.5724 - loss: 0.9022**

**[1m8/9[0m [32m━━━━━━━━━━━━━━━━━[0m[37m━━━[0m [1m0s[0m 76ms/step - accuracy: 0.5789 - loss: 0.8993**

**[1m9/9[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m 77ms/step - accuracy: 0.5810 - loss: 0.8996**

**[1m9/9[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m 91ms/step - accuracy: 0.5972 - loss: 0.9020 - val\_accuracy: 0.3333 - val\_loss: 1.0986**

**Epoch 7/10**

**[1m1/9[0m [32m━━[0m[37m━━━━━━━━━━━━━━━━━━[0m [1m0s[0m 76ms/step - accuracy: 0.6250 - loss: 0.6435**

**[1m2/9[0m [32m━━━━[0m[37m━━━━━━━━━━━━━━━━[0m [1m0s[0m 117ms/step - accuracy: 0.6875 - loss: 0.6651**

**[1m3/9[0m [32m━━━━━━[0m[37m━━━━━━━━━━━━━━[0m [1m0s[0m 92ms/step - accuracy: 0.7222 - loss: 0.6717**

**[1m4/9[0m [32m━━━━━━━━[0m[37m━━━━━━━━━━━━[0m [1m0s[0m 84ms/step - accuracy: 0.7370 - loss: 0.6959**

**[1m5/9[0m [32m━━━━━━━━━━━[0m[37m━━━━━━━━━[0m [1m0s[0m 79ms/step - accuracy: 0.7396 - loss: 0.7199**

**[1m6/9[0m [32m━━━━━━━━━━━━━[0m[37m━━━━━━━[0m [1m0s[0m 76ms/step - accuracy: 0.7378 - loss: 0.7367**

**[1m7/9[0m [32m━━━━━━━━━━━━━━━[0m[37m━━━━━[0m [1m0s[0m 75ms/step - accuracy: 0.7319 - loss: 0.7514**

**[1m8/9[0m [32m━━━━━━━━━━━━━━━━━[0m[37m━━━[0m [1m0s[0m 76ms/step - accuracy: 0.7225 - loss: 0.7657**

**[1m9/9[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m 77ms/step - accuracy: 0.7147 - loss: 0.7761**

**[1m9/9[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m 91ms/step - accuracy: 0.6528 - loss: 0.8597 - val\_accuracy: 0.3333 - val\_loss: 1.0986**

**Epoch 8/10**

**[1m1/9[0m [32m━━[0m[37m━━━━━━━━━━━━━━━━━━[0m [1m0s[0m 75ms/step - accuracy: 0.6250 - loss: 0.8605**

**[1m2/9[0m [32m━━━━[0m[37m━━━━━━━━━━━━━━━━[0m [1m0s[0m 117ms/step - accuracy: 0.6875 - loss: 0.8016**

**[1m3/9[0m [32m━━━━━━[0m[37m━━━━━━━━━━━━━━[0m [1m0s[0m 92ms/step - accuracy: 0.7083 - loss: 0.7904**

**[1m4/9[0m [32m━━━━━━━━[0m[37m━━━━━━━━━━━━[0m [1m0s[0m 84ms/step - accuracy: 0.7031 - loss: 0.7963**

**[1m5/9[0m [32m━━━━━━━━━━━[0m[37m━━━━━━━━━[0m [1m0s[0m 80ms/step - accuracy: 0.7025 - loss: 0.7959**

**[1m6/9[0m [32m━━━━━━━━━━━━━[0m[37m━━━━━━━[0m [1m0s[0m 77ms/step - accuracy: 0.7000 - loss: 0.7937**

**[1m7/9[0m [32m━━━━━━━━━━━━━━━[0m[37m━━━━━[0m [1m0s[0m 75ms/step - accuracy: 0.6944 - loss: 0.7967**

**[1m8/9[0m [32m━━━━━━━━━━━━━━━━━[0m[37m━━━[0m [1m0s[0m 77ms/step - accuracy: 0.6896 - loss: 0.7996**

**[1m9/9[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m 77ms/step - accuracy: 0.6871 - loss: 0.8007**

**[1m9/9[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m 92ms/step - accuracy: 0.6667 - loss: 0.8094 - val\_accuracy: 0.3333 - val\_loss: 1.0986**

**Epoch 9/10**

**[1m1/9[0m [32m━━[0m[37m━━━━━━━━━━━━━━━━━━[0m [1m0s[0m 75ms/step - accuracy: 0.5000 - loss: 0.9981**

**[1m2/9[0m [32m━━━━[0m[37m━━━━━━━━━━━━━━━━[0m [1m0s[0m 117ms/step - accuracy: 0.5938 - loss: 0.9352**

**[1m3/9[0m [32m━━━━━━[0m[37m━━━━━━━━━━━━━━[0m [1m0s[0m 93ms/step - accuracy: 0.6458 - loss: 0.8975**

**[1m4/9[0m [32m━━━━━━━━[0m[37m━━━━━━━━━━━━[0m [1m0s[0m 85ms/step - accuracy: 0.6797 - loss: 0.8733**

**[1m5/9[0m [32m━━━━━━━━━━━[0m[37m━━━━━━━━━[0m [1m0s[0m 80ms/step - accuracy: 0.6988 - loss: 0.8525**

**[1m6/9[0m [32m━━━━━━━━━━━━━[0m[37m━━━━━━━[0m [1m0s[0m 77ms/step - accuracy: 0.7108 - loss: 0.8382**

**[1m7/9[0m [32m━━━━━━━━━━━━━━━[0m[37m━━━━━[0m [1m0s[0m 75ms/step - accuracy: 0.7189 - loss: 0.8279**

**[1m8/9[0m [32m━━━━━━━━━━━━━━━━━[0m[37m━━━[0m [1m0s[0m 76ms/step - accuracy: 0.7189 - loss: 0.8231**

**[1m9/9[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m 77ms/step - accuracy: 0.7193 - loss: 0.8183**

**[1m9/9[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m 92ms/step - accuracy: 0.7222 - loss: 0.7795 - val\_accuracy: 0.3333 - val\_loss: 1.0986**

**Epoch 10/10**

**[1m1/9[0m [32m━━[0m[37m━━━━━━━━━━━━━━━━━━[0m [1m0s[0m 76ms/step - accuracy: 0.7500 - loss: 0.6893**

**[1m2/9[0m [32m━━━━[0m[37m━━━━━━━━━━━━━━━━[0m [1m0s[0m 116ms/step - accuracy: 0.6875 - loss: 0.7811**

**[1m3/9[0m [32m━━━━━━[0m[37m━━━━━━━━━━━━━━[0m [1m0s[0m 93ms/step - accuracy: 0.7083 - loss: 0.7817**

**[1m4/9[0m [32m━━━━━━━━[0m[37m━━━━━━━━━━━━[0m [1m0s[0m 84ms/step - accuracy: 0.7109 - loss: 0.7870**

**[1m5/9[0m [32m━━━━━━━━━━━[0m[37m━━━━━━━━━[0m [1m0s[0m 79ms/step - accuracy: 0.7138 - loss: 0.7942**

**[1m6/9[0m [32m━━━━━━━━━━━━━[0m[37m━━━━━━━[0m [1m0s[0m 77ms/step - accuracy: 0.7198 - loss: 0.7919**

**[1m7/9[0m [32m━━━━━━━━━━━━━━━[0m[37m━━━━━[0m [1m0s[0m 75ms/step - accuracy: 0.7190 - loss: 0.7937**

**[1m8/9[0m [32m━━━━━━━━━━━━━━━━━[0m[37m━━━[0m [1m0s[0m 76ms/step - accuracy: 0.7229 - loss: 0.7925**

**[1m9/9[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m 77ms/step - accuracy: 0.7243 - loss: 0.7906**

**[1m9/9[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m 92ms/step - accuracy: 0.7361 - loss: 0.7759 - val\_accuracy: 0.3333 - val\_loss: 1.0986**

**[1m1/1[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m 426ms/step**

**[1m1/1[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m 455ms/step**

**Prediction: clips, Confidence: 0.34**

**9.Model Interpretation**

The model correctly predicted the class of the input image as **“clip”** with a confidence of **34%**. While the prediction aligns with the true label, the confidence value indicates that the model is somewhat uncertain. This is expected given that the dataset may have limited examples per class or that some images share visual similarities between categories (e.g., shoes vs clips).

The output demonstrates that the CNN model has learned meaningful features from the images and can distinguish between the three object classes. The fact that the correct class was identified validates the overall effectiveness of the preprocessing, label encoding, and feature extraction workflow using MobileNetV2.

**10. Future Improvements (Revised Version)**

To enhance model performance and confidence in predictions, several strategies can be applied in a revised version:

1. **Increase Dataset Size:**
   * Collect more images per class to reduce class imbalance and improve generalization.
2. **Use Pretrained Weights:**
   * Initialize MobileNetV2 with **ImageNet weights** instead of random initialization. This allows the model to leverage pre-learned general features, which often improves accuracy on small datasets.
3. **Hyperparameter Tuning:**
   * Experiment with different learning rates, batch sizes, number of epochs, and dropout rates to optimize model performance.
4. **Advanced Architectures:**
   * Test other CNN architectures such as EfficientNet or ResNet for potential improvement in accuracy and robustness.
5. **Cross-Validation:**
   * Apply k-fold cross-validation to evaluate model stability and reduce the impact of dataset splits on performance metrics.

**11. Conclusion**

In this project, a CNN-based image classification system was successfully implemented to categorize images of shoes, clips, and toothbrushes. The model correctly predicted the class of a sample image, demonstrating its ability to learn and generalize from the training data.

The workflow included careful dataset preparation, preprocessing, and label encoding, followed by model training and fine-tuning using MobileNetV2. Despite moderate confidence in predictions, the model shows promise as a baseline system for multi-class object recognition.

Future enhancements such as data augmentation, increased dataset size, pretrained weights, and hyperparameter optimization can significantly improve model accuracy and reliability. Overall, this project illustrates the power and flexibility of CNNs for practical image classification tasks and provides a solid foundation for further development in computer vision applications.