```
!mkdir -p ~/.kaggle
!mv kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json

import zipfile
import os

dataset_zip = "clothing-dataset-full.zip"
output_folder = "/content/dataset"

with zipfile.ZipFile(dataset_zip, 'r') as zip_ref:
    zip_ref.extractall(output_folder)

print("Dataset extracted successfully!")

import os

image_folder = "/content/dataset/images_original"  # Change this if
needed
print("Number of images:", len(os.listdir(image_folder)))
print("First 5 images:", os.listdir(image_folder)[:5])  # Show first 5
image names

import matplotlib.pyplot as plt
import cv2
import os
import random

# Set the correct image folder
image_folder = "/content/dataset/images_original"

# List all image files
image_files = os.listdir(image_folder)

# Show 5 random images
plt.figure(figsize=(10, 5))
for i in range(5):
    img_path = os.path.join(image_folder, random.choice(image_files))
    img = cv2.imread(img_path)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)  # Convert to RGB

    plt.subplot(1, 5, i+1)
    plt.imshow(img)
```

```
    plt.axis("off")

plt.show()

import numpy as np

# Resize settings
IMG_SIZE = (128, 128)  # Resize to 128x128 pixels

# Process a single image
def process_image(img_path):
    img = cv2.imread(img_path)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img = cv2.resize(img, IMG_SIZE)  # Resize
    img = img / 255.0  # Normalize pixel values (0 to 1)
    return img

# Test on one image
test_img_path = os.path.join(image_folder, image_files[0])
processed_img = process_image(test_img_path)

plt.imshow(processed_img)
plt.axis("off")
plt.show()

import numpy as np

# Resize settings
IMG_SIZE = (128, 128)  # Resize to 128x128 pixels

# Process a single image
def process_image(img_path):
    img = cv2.imread(img_path)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img = cv2.resize(img, IMG_SIZE)  # Resize
    img = img / 255.0  # Normalize pixel values (0 to 1)
    return img

# Test on one image
test_img_path = os.path.join(image_folder, image_files[0])
processed_img = process_image(test_img_path)

plt.imshow(processed_img)
```

```
plt.axis("off")
plt.show()

from sklearn.model_selection import train_test_split

# Assuming image_data is already created
X_train, X_test = train_test_split(image_data, test_size=0.2,
random_state=42)

print("Training Set Shape:", X_train.shape)
print("Testing Set Shape:", X_test.shape)

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten,
Dense

# Define CNN model
model = Sequential([
    Conv2D(32, (3,3), activation='relu', input_shape=(128,128,3)),
    MaxPooling2D(2,2),

    Conv2D(64, (3,3), activation='relu'),
    MaxPooling2D(2,2),

    Flatten(),
    Dense(128, activation='relu'),
    Dense(10, activation='softmax')  # Change 10 to the number of
classes in your dataset
])

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

# Print model summary
model.summary()

import os

# List some images in the dataset folder
image_folder = "/content/dataset/images_original"
image_files = os.listdir(image_folder)
```

```
# List first 5 image filenames
print(image_files[:5])

# Convert to sets for easy comparison
image_files_set = set(image_files)  # Images in the folder
csv_images_set = set(df["image"])   # Images in CSV (after adding .jpg)

# Find missing images
missing_images = csv_images_set - image_files_set  # Images in CSV but
not in folder

if missing_images:
    print("Missing Images:", missing_images)
else:
    print("✅ All CSV images exist in the folder!")

image_data = []
labels = []

for img_file in image_files:
    img_path = os.path.join(image_folder, img_file)

    if img_file in label_dict:  # Ensure label exists
        img = process_image(img_path)  # Your image processing function
        image_data.append(img)
        labels.append(label_dict[img_file])  # Append corresponding
label

image_data = np.array(image_data)
labels = np.array(labels)

from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.utils import to_categorical

# Convert text labels to numerical values
label_encoder = LabelEncoder()
encoded_labels = label_encoder.fit_transform(labels)

# Convert to one-hot encoding
num_classes = len(label_encoder.classes_)  # Number of unique labels
y_data = to_categorical(encoded_labels, num_classes=num_classes)
```

```
print("✅ Labels encoded successfully!")

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(image_data, y_data,
test_size=0.2, random_state=42)

print("✅ Dataset split into train and test sets!")

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten,
Dense, Dropout

# Build CNN model
model = Sequential([
    Conv2D(32, (3,3), activation='relu', input_shape=(128, 128, 3)),  #
Adjust shape if needed
    MaxPooling2D(2,2),

    Conv2D(64, (3,3), activation='relu'),
    MaxPooling2D(2,2),

    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),  # Prevent overfitting
    Dense(num_classes, activation='softmax')  # Output layer
])

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

print("✅ Model defined and compiled successfully!")

import zipfile
import os

zip_path = "/content/clothing-dataset-full.zip"  # Update if your zip
filename is different
extract_path = "/content/dataset"

# Unzip
```

```
with zipfile.ZipFile(zip_path, 'r') as zip_ref:
    zip_ref.extractall(extract_path)

print("✅ Dataset extracted successfully!")

import os

image_folder = "/content/dataset/images_original"
sample_image = df.iloc[0]['image']  # Get first image name

# List all files in the folder
all_files = os.listdir(image_folder)

# Find matching files
matching_files = [f for f in all_files if sample_image in f]
print("Matching files:", matching_files)

import tensorflow as tf
import numpy as np
import pandas as pd
import cv2
import os
from tensorflow.keras.applications.mobilenet_v2 import
preprocess_input, decode_predictions

# Load pre-trained MobileNetV2 model
model = tf.keras.applications.MobileNetV2(weights='imagenet')

# Load your dataset
df = pd.read_csv("/content/dataset/images.csv")

# Path to images
image_folder = "/content/dataset/images_original"

# Get all "Not sure" images
not_sure_images = df[df['label'] == 'Not sure']['image'].tolist()

# Function to preprocess and predict label
def predict_label(image_path):
    img = cv2.imread(image_path)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img = cv2.resize(img, (224, 224))  # Resize for MobileNetV2
    img = preprocess_input(np.expand_dims(img, axis=0))  # Preprocess
```

```
    preds = model.predict(img)  # Predict
    decoded_preds = decode_predictions(preds, top=1)[0][0]  # Get top
prediction
    return decoded_preds[1]  # Return predicted label

# Assign new labels
for img_name in not_sure_images:
    img_path = os.path.join(image_folder, img_name + ".jpg")  # Adjust
extension if needed
    if os.path.exists(img_path):
        new_label = predict_label(img_path)
        df.loc[df['image'] == img_name, 'label'] = new_label
        print(f"Labeled {img_name} as {new_label}")

# Save updated dataset
df.to_csv("/content/dataset/images_labeled.csv", index=False)
print("Updated dataset saved!")

import os
import shutil
import pandas as pd

# Load the CSV file
df = pd.read_csv("/content/dataset/images.csv")

# Define the base image directory
image_folder = "/content/dataset/images_original"

# Loop through the dataset and move images into labeled subdirectories
for index, row in df.iterrows():
    img_name = row["image"] + ".jpg"  # Image filename
    label = row["label"] if row["label"] != "Not sure" else "Unknown"
# Assign 'Unknown' for uncertain labels

    # Create the label directory if it does not exist
    label_path = os.path.join(image_folder, label)
    os.makedirs(label_path, exist_ok=True)

    # Move image to the corresponding label directory
    src = os.path.join(image_folder, img_name)
    dst = os.path.join(label_path, img_name)
```

```
    if os.path.exists(src):  # Check if the image file exists before
moving
        shutil.move(src, dst)

print("✅ Images have been moved to labeled folders!")

import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import os

# Define dataset paths
train_dir = "/content/dataset/images_original"

# Define ImageDataGenerator for preprocessing
datagen = ImageDataGenerator(
    rescale=1.0/255,  # Normalize pixel values
    validation_split=0.2,  # 80% training, 20% validation
    horizontal_flip=True,
    rotation_range=20
)

# Load images from directory
train_data = datagen.flow_from_directory(
    train_dir,
    target_size=(224, 224),  # MobileNetV2 input size
    batch_size=32,
    class_mode="categorical",
    subset="training"
)

val_data = datagen.flow_from_directory(
    train_dir,
    target_size=(224, 224),
    batch_size=32,
    class_mode="categorical",
    subset="validation"
)

from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D
from tensorflow.keras.models import Model

# Load MobileNetV2 (exclude top layers)
```

```
base_model = MobileNetV2(weights="imagenet", include_top=False)

# Freeze the base model layers (optional, for transfer learning)
base_model.trainable = False

# Add new classification layers
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(128, activation="relu")(x)
x = Dense(len(train_data.class_indices), activation="softmax")(x)  #
Output layer

# Define new model
model = Model(inputs=base_model.input, outputs=x)

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

model.summary()  # Show model structure

# Train the model
history = model.fit(
    train_data,
    validation_data=val_data,
    epochs=10,  # Increase for better accuracy
    steps_per_epoch=len(train_data),
    validation_steps=len(val_data)
)

# Save model
model.save("/content/clothing_classifier.h5")
print("Model saved!")

from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Define image size
image_height = 224
image_width = 224
```

```
# Data Augmentation
datagen = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    horizontal_flip=True,
    zoom_range=0.2,
    shear_range=0.2,
    fill_mode='nearest'
)

# Load dataset
train_data = datagen.flow_from_directory(
    "/content/dataset",  # Replace with actual dataset path
    batch_size=32,
    target_size=(image_height, image_width)
)

from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.models import Model

base_model = MobileNetV2(weights='imagenet', include_top=False,
input_shape=(224, 224, 3))
base_model.trainable = False  # Freeze base model layers

x = Flatten()(base_model.output)
x = Dense(256, activation='relu')(x)
x = Dense(128, activation='relu')(x)
output_layer = Dense(num_classes, activation='softmax')(x)

model = Model(inputs=base_model.input, outputs=output_layer)
model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])


import os
from PIL import Image

dataset_path = "/content/dataset"  # Update with your dataset path

def find_corrupt_images(directory):
    corrupt_images = []
```

```
    for root, _, files in os.walk(directory):
        for file in files:
            file_path = os.path.join(root, file)
            try:
                img = Image.open(file_path)  # Try opening the image
                img.verify()  # Verify if it's a valid image
            except (IOError, SyntaxError):
                print(f"Corrupt image detected: {file_path}")
                corrupt_images.append(file_path)
    return corrupt_images

corrupt_images = find_corrupt_images(dataset_path)

# Delete corrupt images
for img_path in corrupt_images:
    os.remove(img_path)
    print(f"Deleted: {img_path}")

print(f"Total corrupt images removed: {len(corrupt_images)}")

import os
import random
import cv2
import matplotlib.pyplot as plt
from matplotlib.patches import Rectangle

# Select a random image from the dataset
image_folder = "/content/fashion_dataset/train"  # Update if needed
image_files = [f for f in os.listdir(image_folder) if
f.endswith(('.jpg', '.png'))]
random_image = random.choice(image_files)
image_path = os.path.join(image_folder, random_image)

# Get image ID from filename
image_id = None
for img in coco_data["images"]:
    if img["file_name"] == random_image:
        image_id = img["id"]
        break

# Extract bounding boxes for this image
bboxes = []
labels = []
```

```
for ann in coco_data["annotations"]:
    if ann["image_id"] == image_id:
        bboxes.append(ann["bbox"])
        labels.append(category_mapping[ann["category_id"]])  # Map
category_id to class name

# Load image
img = cv2.imread(image_path)
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

# Display image
fig, ax = plt.subplots(1, figsize=(8, 6))
ax.imshow(img)

# Draw bounding boxes
for bbox, label in zip(bboxes, labels):
    x, y, w, h = bbox
    rect = Rectangle((x, y), w, h, linewidth=2, edgecolor='r',
facecolor='none')
    ax.add_patch(rect)
    ax.text(x, y - 5, label, color='red', fontsize=12,
backgroundcolor="white")

plt.axis("off")
plt.show()

import os
import shutil
import random
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# ✅ **Step 1: Define Paths**
original_dataset_path = "/content/fashion_dataset"
classification_dataset_path = "/content/classification_dataset"

# ✅ **Step 2: Check Dataset Structure**
if not os.path.exists(original_dataset_path):
    raise FileNotFoundError(f"❌ Dataset not found at
{original_dataset_path}. Please upload it.")

print("✅ Dataset found!")
```

```
# ✅ **Step 3: Create Train/Valid Split**
os.makedirs(f"{classification_dataset_path}/train", exist_ok=True)
os.makedirs(f"{classification_dataset_path}/valid", exist_ok=True)

# Get class names from folders
class_names = [d for d in os.listdir(original_dataset_path) if
os.path.isdir(os.path.join(original_dataset_path, d))]

# Split dataset
for class_name in class_names:
    image_files = os.listdir(os.path.join(original_dataset_path,
class_name))
    train_files, valid_files = train_test_split(image_files,
test_size=0.2, random_state=42)

    os.makedirs(f"{classification_dataset_path}/train/{class_name}",
exist_ok=True)
    os.makedirs(f"{classification_dataset_path}/valid/{class_name}",
exist_ok=True)

    for file in train_files:
        shutil.move(os.path.join(original_dataset_path, class_name,
file),
                    os.path.join(classification_dataset_path, "train",
class_name, file))

    for file in valid_files:
        shutil.move(os.path.join(original_dataset_path, class_name,
file),
                    os.path.join(classification_dataset_path, "valid",
class_name, file))

print("✅ Dataset successfully split into train/valid folders!")

# ✅ **Step 4: Define Data Generators**
train_dir = os.path.join(classification_dataset_path, "train")
valid_dir = os.path.join(classification_dataset_path, "valid")

train_datagen = ImageDataGenerator(rescale=1.0/255, rotation_range=20,
horizontal_flip=True)
valid_datagen = ImageDataGenerator(rescale=1.0/255)
```

```
train_generator = train_datagen.flow_from_directory(
    train_dir, target_size=(224, 224), batch_size=32,
class_mode='categorical')

valid_generator = valid_datagen.flow_from_directory(
    valid_dir, target_size=(224, 224), batch_size=32,
class_mode='categorical')

print("✅ Image generators are ready!")
```