```
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.model_selection import train_test_split
import numpy as np
import os
import cv2

# --- 1. Configuration and Hyperparameters ---
IMAGE_SIZE = (128, 128) # Resize images to this dimension
BATCH_SIZE = 32
EPOCHS = 20
NUM_CLASSES = 5 # Example: stripes, floral, plaid, solid, polka_dot
DATA_DIR = 'path/to/your/fabric_dataset' # IMPORTANT: Change this to
your dataset directory

# --- 2. Load and Preprocess Data ---

def load_data(data_dir, image_size):
    """
    Loads images from subdirectories (each subdirectory is a class)
    and their corresponding labels.
    Assumes directory structure:
    data_dir/
        class1/
            img1.jpg
            img2.jpg
        class2/
            img3.jpg
            ...
    """
    images = []
    labels = []
    class_names = sorted(os.listdir(data_dir))
    class_to_idx = {name: i for i, name in enumerate(class_names)}

    print(f"Detected classes: {class_names}")

    for class_name in class_names:
        class_path = os.path.join(data_dir, class_name)
        if os.path.isdir(class_path):
            for img_name in os.listdir(class_path):
                img_path = os.path.join(class_path, img_name)
                try:
                    img = cv2.imread(img_path)
                    if img is not None:
                        img = cv2.resize(img, image_size)
                        images.append(img)
                        labels.append(class_to_idx[class_name])
                    else:
                        print(f"Warning: Could not read image
{img_path}")
                except Exception as e:
                    print(f"Error loading image {img_path}: {e}")

    return np.array(images), np.array(labels), class_names

# Load images and labels
print("Loading data...")
try:
    X, y, class_names = load_data(DATA_DIR, IMAGE_SIZE)
    if len(X) == 0:
        raise ValueError("No images found. Please check DATA_DIR.")
    print(f"Loaded {len(X)} images with {len(class_names)} classes.")
except ValueError as e:
    print(f"Error: {e}")
    print("Please ensure 'DATA_DIR' points to a directory containing
subdirectories of fabric images.")
    # Exit or provide placeholder data for demonstration if actual
data is not available
    X = np.random.rand(100, IMAGE_SIZE[0], IMAGE_SIZE[1], 3) * 255 #
Placeholder data
    y = np.random.randint(0, NUM_CLASSES, 100) # Placeholder labels
    class_names = [f'class_{i}' for i in range(NUM_CLASSES)]
    print("Using placeholder data for demonstration.")


# Normalize pixel values to [0, 1]
X = X.astype('float32') / 255.0

# Convert labels to one-hot encoding
y = tf.keras.utils.to_categorical(y, num_classes=len(class_names))

# --- 3. Split Data into Training, Validation, and Test Sets ---
# First, split into training + validation and test sets
X_train_val, X_test, y_train_val, y_test = train_test_split(
    X, y, test_size=0.15, random_state=42, stratify=y
)

# Then, split the training + validation set into training and
validation sets
X_train, X_val, y_train, y_val = train_test_split(
    X_train_val, y_train_val, test_size=0.176, random_state=42,
stratify=y_train_val
```

```
) # 0.176 of 0.85 is roughly 0.15 of total data (0.85 * 0.176 =
0.1496)

print(f"Training set shape: {X_train.shape}, {y_train.shape}")
print(f"Validation set shape: {X_val.shape}, {y_val.shape}")
print(f"Test set shape: {X_test.shape}, {y_test.shape}")

# --- 4. Data Augmentation (for training data) ---
train_datagen = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

# No augmentation for validation and test sets, just normalization
(already done)
# val_datagen = ImageDataGenerator()
# test_datagen = ImageDataGenerator()

# --- 5. Define the CNN Model Architecture ---
model = Sequential([
    # Convolutional Block 1
    Conv2D(32, (3, 3), activation='relu', input_shape=(IMAGE_SIZE[0],
IMAGE_SIZE[1], 3)),
    MaxPooling2D((2, 2)),
    Dropout(0.25), # Dropout for regularization

    # Convolutional Block 2
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Dropout(0.25),

    # Convolutional Block 3
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Dropout(0.25),

    # Flatten the output for the Dense layers
    Flatten(),

    # Fully Connected (Dense) Layers
    Dense(256, activation='relu'),
    Dropout(0.5), # More dropout before the final classification layer
    Dense(len(class_names), activation='softmax') # Output layer with
softmax for multi-class classification
])

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

model.summary()

# --- 7. Train the Model ---
print("Training the model...")
history = model.fit(
    train_datagen.flow(X_train, y_train, batch_size=BATCH_SIZE),
    epochs=EPOCHS,
    validation_data=(X_val, y_val),
    verbose=1
)

# --- 8. Evaluate the Model on the Test Set ---
print("\nEvaluating the model on the test set...")
loss, accuracy = model.evaluate(X_test, y_test, verbose=0)
print(f"Test Loss: {loss:.4f}")
print(f"Test Accuracy: {accuracy:.4f}")

# Optional: Make predictions on the test set
# predictions = model.predict(X_test)
# predicted_classes = np.argmax(predictions, axis=1)
# true_classes = np.argmax(y_test, axis=1)

# print("\nSome sample predictions vs true labels:")
# for i in range(10): # Print for first 10 test samples
#     print(f"Sample {i+1}: True: {class_names[true_classes[i]]},
Predicted: {class_names[predicted_classes[i]]}")

# --- 9. Save the Model (Optional) ---
# model.save('fabric_pattern_classifier.h5')
# print("\nModel saved as 'fabric_pattern_classifier.h5'")
```