```
from flask import Flask, render_template, request
import tensorflow as tf
import numpy as np
from PIL import Image
import pickle

app = Flask(__name__)

# Load model and label encoder
model = tf.keras.models.load_model('model.h5')
with open('label_encoder.pkl', 'rb') as le_file:
    label_encoder = pickle.load(le_file)

def prepare_image(image):
    image = Image.open(image).resize((224, 224))
    image = np.array(image)
    image = np.expand_dims(image, axis=0)
    return image

@app.route("/", methods=['GET', 'POST'])
def home():
    if request.method == 'POST':
        image = request.files['image']
        image = prepare_image(image)
        prediction = model.predict(image)
        predicted_label = label_encoder.inverse_transform([np.argmax(prediction)])
        return render_template("index.html", prediction=predicted_label[0])
    return render_template("index.html")

if __name__ == "__main__":
    app.run(debug=True)
```

```
import joblib
from sklearn.preprocessing import LabelEncoder

# Create and fit the LabelEncoder
label_encoder = LabelEncoder()
label_encoder.fit(['class1', 'class2'])  # Replace with your actual class names

# Save the LabelEncoder
joblib.dump(label_encoder, 'label_encoder.pkl')
print("LabelEncoder saved as 'label_encoder.pkl'")

# To load the LabelEncoder later
# label_encoder = joblib.load('label_encoder.pkl')
# print("LabelEncoder loaded from 'label_encoder.pkl'")
```

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

def build_model(input_shape=(224, 224, 3), num_classes=2):
  model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=input_shape),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(512, activation='relu'),
    Dropout(0.5),
    Dense(num_classes, activation='softmax')
  ])
  model.compile(optimizer='adam',
          loss='sparse_categorical_crossentropy',
          metrics=['accuracy'])
  return model

# Build the model
model = build_model()

# Dummy data for example (replace with actual data loading code)
train_images = np.random.rand(100, 224, 224, 3)
train_labels = np.random.randint(0, 2, 100)
test_images = np.random.rand(20, 224, 224, 3)
test_labels = np.random.randint(0, 2, 20)

# Train the model
model.fit(train_images, train_labels, epochs=10, validation_split=0.2)

# Save the model
try:
  model.save('model.h5')
  print("Model saved successfully.")
except Exception as e:
  print(f"Error saving model: {e}")
```

```
# Load the model
try:
    loaded_model = tf.keras.models.load_model('model.h5')
    print("Model loaded successfully.")
except Exception as e:
    print(f"Error loading model: {e}")

# Evaluate the loaded model
test_loss, test_accuracy = loaded_model.evaluate(test_images, test_labels)
print(f"Test accuracy: {test_accuracy}")

# Make predictions
predictions = loaded_model.predict(test_images)
print(f"Predictions: {predictions}")
```

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

def build_model(input_shape=(224, 224, 3), num_classes=2):
  model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=input_shape),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(512, activation='relu'),
    Dropout(0.5),
    Dense(num_classes, activation='softmax')
  ])
  model.compile(optimizer='adam',
          loss='sparse_categorical_crossentropy',
          metrics=['accuracy'])
  return model

# Build and train the model
model = build_model()

# Dummy data for example (replace with actual data loading code)
train_images = np.random.rand(100, 224, 224, 3)
train_labels = np.random.randint(0, 2, 100)
test_images = np.random.rand(20, 224, 224, 3)
test_labels = np.random.randint(0, 2, 20)

# Train the model
model.fit(train_images, train_labels, epochs=10, validation_split=0.2)

# Save the model
model.save('model.h5')
print("Model saved successfully.")

# Load the model
loaded_model = tf.keras.models.load_model('model.h5')
```

```
print("Model loaded successfully.")

# Evaluate the loaded model
test_loss, test_accuracy = loaded_model.evaluate(test_images, test_labels)
print(f"Test accuracy: {test_accuracy}")

# Make predictions
predictions = loaded_model.predict(test_images)
print(f"Predictions: {predictions}")

# Generate classification report
from sklearn.metrics import classification_report
report = classification_report(test_labels, np.argmax(predictions, axis=1))

# Save evaluation results
results = {
   'test_loss': test_loss,
   'test_accuracy': test_accuracy,
   'classification_report': report
}

with open('evaluation_results.json', 'w') as file:
   json.dump(results, file, indent=4)
print("Evaluation results saved successfully.")
```

```
import os
import shutil

source_dir = 'dataset/test'
dest_dirs = {
    'class1': 'dataset/test/class1',
    'class2': 'dataset/test/class2',
}

# Create destination directories if they don't exist
for dir in dest_dirs.values():
    os.makedirs(dir, exist_ok=True)

def get_class_from_filename(filename):
    # Customize this function based on how you determine class from filename
    if 'class1' in filename:
        return 'class1'
    elif 'class2' in filename:
        return 'class2'
    else:
        return None

# List all items in the source directory and process files only
for item in os.listdir(source_dir):
    src_path = os.path.join(source_dir, item)

    # Check if the item is a file
    if os.path.isfile(src_path):
        class_name = get_class_from_filename(item)
        if class_name and class_name in dest_dirs:
            dest_path = os.path.join(dest_dirs[class_name], item)
            shutil.move(src_path, dest_path)
            print(f"Moved {item} to {dest_path}")
        else:
            print(f"Class not found for {item}, or directory does not exist.")
    else:
        print(f"{item} is not a file or does not exist.")
```

```
import os
import numpy as np
from PIL import Image
from sklearn.preprocessing import LabelEncoder
import pickle

def load_and_preprocess_images(directory):
    images = []
    labels = []
    for label_dir in os.listdir(directory):
        if not os.path.isdir(os.path.join(directory, label_dir)):
            continue
        for image_file in os.listdir(os.path.join(directory, label_dir)):
            image_path = os.path.join(directory, label_dir, image_file)
            image = Image.open(image_path).resize((224, 224))
            images.append(np.array(image))
            labels.append(label_dir)
    images = np.array(images)
    labels = np.array(labels)

    label_encoder = LabelEncoder()
    labels = label_encoder.fit_transform(labels)

    with open('label_encoder.pkl', 'wb') as le_file:
        pickle.dump(label_encoder, le_file)

    return images, labels

if __name__ == "__main__":
    train_images, train_labels = load_and_preprocess_images('dataset/train')
    test_images, test_labels = load_and_preprocess_images('dataset/test')

    np.save('train_images.npy', train_images)
    np.save('train_labels.npy', train_labels)
    np.save('test_images.npy', test_images)
    np.save('test_labels.npy', test_labels)
```