```
# Importing the necessary libraries
import cv2
import os
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Conv1D, MaxPooling1D, GlobalAveragePooling1D, Dense
from skimage import feature


# Function to calculate Local Binary Pattern (LBP) histogram
def calculate_lbp(image, num_points=8, radius=3):
    try:
        gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        lbp = feature.local_binary_pattern(gray, num_points, radius, method="uniform")
        hist, _ = np.histogram(lbp.ravel(), bins=num_points + 2, range=(0, num_points + 2))
        hist = hist.astype("float")
        hist /= (hist.sum() + 1e-7)
        return hist.flatten()
    except Exception as e:
        print(f"Error in calculate_lbp: {e}")
        return None


# Path to the dataset directory
data_dir = "/content/drive/MyDrive/test/train"
categories = ["Closed", "Open", "no_yawn", "yawn"]
data = []


# Load images and extract features
for category in categories:
    path = os.path.join(data_dir, category)
    label = categories.index(category)
    for img_name in os.listdir(path):
        img_path = os.path.join(path, img_name)

        try:
            img = cv2.imread(img_path)
            if img is None:
                raise Exception(f"Failed to read image at path: {img_path}")

            features = calculate_lbp(img)
            if features is not None:
                data.append([features, label])
        except Exception as e:
            print(f"Error processing image {img_path}: {e}")
```

```
Error processing image /content/drive/MyDrive/test/train/yawn/512.jpg: Failed to read image at path: /content/drive/MyDrive/test/train/
Error processing image /content/drive/MyDrive/test/train/yawn/143.jpg: Failed to read image at path: /content/drive/MyDrive/test/train/
```

```
# Shuffle the data
np.random.shuffle(data)


# Separate features and labels
X = [features for features, _ in data]
y = [label for _, label in data]


# Find the maximum length of the histograms
max_hist_length = max(len(hist) for hist in X)


# Pad histograms to have the same length
X_padded = np.array([np.pad(hist, (0, max_hist_length - len(hist))) for hist in X])


# Convert to numpy arrays
X = np.array(X_padded)
y = np.array(y)
```

```
# Convert labels to categorical format
label_encoder = LabelEncoder()
encoded_labels = label_encoder.fit_transform(y)
categorical_labels = to_categorical(encoded_labels)


# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, categorical_labels, test_size=0.3, random_state=42)


# Define a CNN model with 1D convolutional layers
model = Sequential()
model.add(Conv1D(32, kernel_size=3, activation='relu', input_shape=(X_train.shape[1], 1)))
model.add(MaxPooling1D(pool_size=2))
model.add(Conv1D(64, kernel_size=3, activation='relu'))
model.add(MaxPooling1D(pool_size=2))
model.add(GlobalAveragePooling1D())
model.add(Dense(64, activation='relu'))
model.add(Dense(4, activation='softmax'))
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])


# Train the model
model.fit(X_train[:, :, np.newaxis], y_train, epochs=50, batch_size=32, validation_data=(X_test[:, :, np.newaxis], y_test))
```

```
63/63 [==============================] - 0s 5ms/step - loss: 0.5339 - accuracy: 0.7128 - val_loss: 0.5183 - val_accuracy: 0.7419
Epoch 23/50
63/63 [==============================] - 0s 4ms/step - loss: 0.5327 - accuracy: 0.7153 - val_loss: 0.5243 - val_accuracy: 0.7326
Epoch 24/50
63/63 [==============================] - 0s 5ms/step - loss: 0.5300 - accuracy: 0.7138 - val_loss: 0.5148 - val_accuracy: 0.7350
Epoch 25/50
63/63 [==============================] - 0s 5ms/step - loss: 0.5254 - accuracy: 0.7247 - val_loss: 0.5170 - val_accuracy: 0.7280
Epoch 26/50
63/63 [==============================] - 0s 5ms/step - loss: 0.5266 - accuracy: 0.7212 - val_loss: 0.5193 - val_accuracy: 0.7350
Epoch 27/50
63/63 [==============================] - 0s 4ms/step - loss: 0.5332 - accuracy: 0.7073 - val_loss: 0.5246 - val_accuracy: 0.7315
Epoch 28/50
63/63 [==============================] - 0s 5ms/step - loss: 0.5303 - accuracy: 0.7192 - val_loss: 0.5211 - val_accuracy: 0.7222
Epoch 29/50
63/63 [==============================] - 0s 5ms/step - loss: 0.5247 - accuracy: 0.7188 - val_loss: 0.5097 - val_accuracy: 0.7326
Epoch 30/50
63/63 [==============================] - 0s 5ms/step - loss: 0.5287 - accuracy: 0.7173 - val_loss: 0.5198 - val_accuracy: 0.7072
Epoch 31/50
63/63 [==============================] - 0s 5ms/step - loss: 0.5185 - accuracy: 0.7282 - val_loss: 0.5070 - val_accuracy: 0.7442
Epoch 32/50
63/63 [==============================] - 0s 4ms/step - loss: 0.5204 - accuracy: 0.7272 - val_loss: 0.5363 - val_accuracy: 0.7176
Epoch 33/50
63/63 [==============================] - 0s 5ms/step - loss: 0.5242 - accuracy: 0.7183 - val_loss: 0.5109 - val_accuracy: 0.7292
Epoch 34/50
63/63 [==============================] - 0s 5ms/step - loss: 0.5198 - accuracy: 0.7277 - val_loss: 0.5081 - val_accuracy: 0.7211
Epoch 35/50
63/63 [==============================] - 0s 5ms/step - loss: 0.5163 - accuracy: 0.7341 - val_loss: 0.5182 - val_accuracy: 0.7257
Epoch 36/50
63/63 [==============================] - 0s 5ms/step - loss: 0.5194 - accuracy: 0.7202 - val_loss: 0.5154 - val_accuracy: 0.7338
Epoch 37/50
63/63 [==============================] - 0s 4ms/step - loss: 0.5138 - accuracy: 0.7302 - val_loss: 0.5061 - val_accuracy: 0.7454
Epoch 38/50
63/63 [==============================] - 0s 4ms/step - loss: 0.5187 - accuracy: 0.7232 - val_loss: 0.5004 - val_accuracy: 0.7488
Epoch 39/50
63/63 [==============================] - 0s 3ms/step - loss: 0.5082 - accuracy: 0.7302 - val_loss: 0.5053 - val_accuracy: 0.7454
Epoch 40/50
63/63 [==============================] - 0s 3ms/step - loss: 0.5135 - accuracy: 0.7237 - val_loss: 0.5153 - val_accuracy: 0.7269
Epoch 41/50
63/63 [==============================] - 0s 3ms/step - loss: 0.5065 - accuracy: 0.7282 - val_loss: 0.5263 - val_accuracy: 0.7106
Epoch 42/50
63/63 [==============================] - 0s 3ms/step - loss: 0.5115 - accuracy: 0.7282 - val_loss: 0.5065 - val_accuracy: 0.7350
Epoch 43/50
63/63 [==============================] - 0s 3ms/step - loss: 0.5098 - accuracy: 0.7336 - val_loss: 0.5029 - val_accuracy: 0.7373
Epoch 44/50
63/63 [==============================] - 0s 3ms/step - loss: 0.5051 - accuracy: 0.7326 - val_loss: 0.4940 - val_accuracy: 0.7535
Epoch 45/50
63/63 [==============================] - 0s 4ms/step - loss: 0.5038 - accuracy: 0.7346 - val_loss: 0.5295 - val_accuracy: 0.7164
Epoch 46/50
63/63 [==============================] - 0s 3ms/step - loss: 0.5069 - accuracy: 0.7307 - val_loss: 0.5077 - val_accuracy: 0.7373
Epoch 47/50
63/63 [==============================] - 0s 3ms/step - loss: 0.5040 - accuracy: 0.7277 - val_loss: 0.4971 - val_accuracy: 0.7488
Epoch 48/50
63/63 [==============================] - 0s 3ms/step - loss: 0.5049 - accuracy: 0.7277 - val_loss: 0.4994 - val_accuracy: 0.7350
Epoch 49/50
63/63 [==============================] - 0s 3ms/step - loss: 0.5001 - accuracy: 0.7371 - val_loss: 0.4900 - val_accuracy: 0.7535
Epoch 50/50
63/63 [==============================] - 0s 3ms/step - loss: 0.4960 - accuracy: 0.7470 - val_loss: 0.4986 - val_accuracy: 0.7546
<keras.src.callbacks.History at 0x78ade01ebb20>
```

```
# Evaluate the model
loss, accuracy = model.evaluate(X_test[:, :, np.newaxis], y_test)
print(f"Test Loss: {loss}, Test Accuracy: {accuracy}")
```

```
27/27 [==============================] - 0s 2ms/step - loss: 0.4986 - accuracy: 0.7546
Test Loss: 0.4986199140548706, Test Accuracy: 0.7546296119689941
```

```
# saving the model
model.save('/content/drive/MyDrive/test/Creation_model.h5')
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your model as an HDF5 file via `
  saving_api.save_model(
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```