In [1]:
```
import os
import pandas as pd
import numpy as np
import shutil
import random

#Import Libraries for image processing and visualization
import cv2
import matplotlib.pyplot as plt
import seaborn as sns
from PIL import Image, ImageEnhance, ImageFilter, ImageStat
from IPython.display import display

# Import Tensorflow and Keras for deep learning
import tensorflow as tf
from tensorflow.keras.preprocessing import image as tf_image
from tensorflow.keras import layers, models, applications
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, BatchNormalization, LSTM
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau

# Import Scickit-Learn for machine learning models and evaluation
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.utils import shuffle
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

# Import tqdm fro progress bar visualization
from tqdm import tqdm
```

In [2]:
```
# Download latest version
base_folder = r"C:\Users\wael.alaswad\PycharmProjects\DeepLearning\Detect_fatigue\Dataset\Data"

print("base_folder to dataset files:", base_folder)

# Prepare a list to store image data and labels
data =[]
```

```
base_folder to dataset files: C:\Users\wael.alaswad\PycharmProjects\DeepLearning\Detect_fatigue\Dataset\Data
```

In [3]:
```
# Load images and corresponding labels
for disease_folder in os.listdir(base_folder):
  disease_path = os.path.join(base_folder, disease_folder)
  if os.path.isdir(disease_path):
    for img_file in os.listdir(disease_path):
      img_path = os.path.join(disease_path, img_file)
      if img_file.lower().endswith((".png", ".jpg", 'jpeg', '.tiff', '.bmp', '.gif')):

        try:
          img = Image.open(img_path).convert('RGB').resize((224,224))
          data.append({'image': img, 'label': disease_folder})

        except Exception as e:
          print(f'Error Loading Image {img_file}: {e}')


df = pd.DataFrame(data)
df
```

Out[3]:

	image	label
0	<PIL.Image.Image image mode=RGB size=224x224 a...	Fatigue
1	<PIL.Image.Image image mode=RGB size=224x224 a...	Fatigue
2	<PIL.Image.Image image mode=RGB size=224x224 a...	Fatigue
3	<PIL.Image.Image image mode=RGB size=224x224 a...	Fatigue
4	<PIL.Image.Image image mode=RGB size=224x224 a...	Fatigue
...	...	...
2195	<PIL.Image.Image image mode=RGB size=224x224 a...	NonFatigue
2196	<PIL.Image.Image image mode=RGB size=224x224 a...	NonFatigue
2197	<PIL.Image.Image image mode=RGB size=224x224 a...	NonFatigue
2198	<PIL.Image.Image image mode=RGB size=224x224 a...	NonFatigue
2199	<PIL.Image.Image image mode=RGB size=224x224 a...	NonFatigue

2200 rows × 2 columns

In [4]:

```
plt.figure(figsize = (50,50))

# Loop Over to get 50 sample
for n, i in enumerate(np.random.randint(0, len(df), 50)):
  print(f'Image Number {n}, at Index {i}')
  #create subplot
  plt.subplot(10, 10, n+1)
  img = df.iloc[i]["image"]

  img_resized = img.resize((224, 224))
  plt.imshow(img_resized)
  plt.axis("off")

  plt.title(df.iloc[i]['label'],fontsize=25 )

plt.tight_layout()

plt.show()
```

```
Image Number 0, at Index 1736
Image Number 1, at Index 1454
Image Number 2, at Index 760
Image Number 3, at Index 613
Image Number 4, at Index 386
Image Number 5, at Index 325
Image Number 6, at Index 1154
Image Number 7, at Index 1692
Image Number 8, at Index 392
Image Number 9, at Index 1573
Image Number 10, at Index 1216
Image Number 11, at Index 1732
Image Number 12, at Index 597
Image Number 13, at Index 2108
Image Number 14, at Index 523
Image Number 15, at Index 854
Image Number 16, at Index 1997
Image Number 17, at Index 1669
Image Number 18, at Index 1636
Image Number 19, at Index 2113
Image Number 20, at Index 1019
Image Number 21, at Index 984
Image Number 22, at Index 2178
Image Number 23, at Index 270
Image Number 24, at Index 1231
Image Number 25, at Index 1023
Image Number 26, at Index 1128
Image Number 27, at Index 1637
Image Number 28, at Index 1869
Image Number 29, at Index 1162
Image Number 30, at Index 1281
Image Number 31, at Index 2101
Image Number 32, at Index 318
Image Number 33, at Index 1001
Image Number 34, at Index 241
Image Number 35, at Index 1097
Image Number 36, at Index 139
Image Number 37, at Index 364
Image Number 38, at Index 1618
Image Number 39, at Index 975
Image Number 40, at Index 791
Image Number 41, at Index 1742
Image Number 42, at Index 1960
Image Number 43, at Index 1729
Image Number 44, at Index 410
Image Number 45, at Index 1732
Image Number 46, at Index 2007
Image Number 47, at Index 303
Image Number 48, at Index 114
Image Number 49, at Index 43
```

In [5]:
```
images = np.array([np.array(img) for img in df['image']])
```

```
labels = df['label'].values

# Label Encoder
LE = LabelEncoder()
labels_encoded = LE.fit_transform(labels)

# One Hot Encoder
labels_one_hot = to_categorical(labels_encoded)

print(labels_encoded)
print(labels_one_hot)
```

```
[0 0 0 ... 1 1 1]
[[1. 0.]
 [1. 0.]
 [1. 0.]
 ...
 [0. 1.]
 [0. 1.]
 [0. 1.]]
```

In [6]:
```
X_train, X_test, y_train, y_test = train_test_split(images,labels_encoded,test_size=0.2, random_state=42)

X_train_cnn = X_train / 255.0
X_test_cnn = X_test / 255.0

X_train_cnn = X_train_cnn.reshape(-1, 224, 224, 3)
X_test_cnn = X_test_cnn.reshape(-1, 224, 224, 3)
print(f'CNN input image: {X_train_cnn.shape}')
print(f'CNN test image: {X_test_cnn.shape}')
```

```
CNN input image: (1760, 224, 224, 3)
CNN test image: (440, 224, 224, 3)
```

In [7]:
```
# Build the model
cnnCT = Sequential()

# Block1
# Correct the input_shape to match the actual image size (224, 224, 3)
cnnCT.add(Conv2D(32,(3,3), activation = 'relu', input_shape = (224, 224, 3)))
cnnCT.add(BatchNormalization())
cnnCT.add(MaxPooling2D((2,2)))

# Block2
cnnCT.add(Conv2D(64,(3,3), activation = 'relu'))
cnnCT.add(BatchNormalization())
cnnCT.add(MaxPooling2D((2,2)))

# Block3
cnnCT.add(Conv2D(128,(3,3), activation = 'relu'))
cnnCT.add(BatchNormalization())
cnnCT.add(MaxPooling2D((2,2)))

# Block4
cnnCT.add(Conv2D(256,(3,3), activation = 'relu'))
cnnCT.add(BatchNormalization())
cnnCT.add(MaxPooling2D((2,2)))

# Flatten
cnnCT.add(Flatten())

# Fully Connected
cnnCT.add(Dense(256, activation = 'relu'))
cnnCT.add(Dropout(0.3))
cnnCT.add(Dense(len(LE.classes_), activation = 'softmax'))


cnnCT.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Convert y_train and y_test to one-hot encoding
y_train_one_hot = to_categorical(y_train, num_classes=len(LE.classes_))
y_test_one_hot = to_categorical(y_test, num_classes=len(LE.classes_))

cnnCT.fit(X_train_cnn, y_train_one_hot, validation_split = 0.2, epochs=50, batch_size=32)

cnn_test_loss, cnn_test_accuracy = cnnCT.evaluate(X_test_cnn, y_test_one_hot)
print(f'CNN Test Loss: {cnn_test_loss}, CNN Test Accuracy: {cnn_test_accuracy}')
```

```
c:\Users\wael.alaswad\AppData\Local\Programs\Python\Python311\Lib\site-packages\keras\src\layers\convolutional\base_conv.py:113: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
Epoch 1/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 88s 2s/step - accuracy: 0.7124 - loss: 5.5463 - val_accuracy: 0.4460 - val_loss: 4.2383
Epoch 2/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 86s 2s/step - accuracy: 0.7976 - loss: 0.8122 - val_accuracy: 0.4489 - val_loss: 3.6208
Epoch 3/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 82s 2s/step - accuracy: 0.8523 - loss: 0.3940 - val_accuracy: 0.4489 - val_loss: 7.1829
Epoch 4/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 81s 2s/step - accuracy: 0.8764 - loss: 0.3345 - val_accuracy: 0.4886 - val_loss: 1.5679
Epoch 5/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 82s 2s/step - accuracy: 0.8956 - loss: 0.2671 - val_accuracy: 0.5256 - val_loss: 1.4535
Epoch 6/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 86s 2s/step - accuracy: 0.9197 - loss: 0.2198 - val_accuracy: 0.4489 - val_loss: 8.8844
Epoch 7/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 89s 2s/step - accuracy: 0.9105 - loss: 0.2054 - val_accuracy: 0.4773 - val_loss: 3.7346
Epoch 8/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 89s 2s/step - accuracy: 0.9325 - loss: 0.1691 - val_accuracy: 0.5312 - val_loss: 2.9083
Epoch 9/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 90s 2s/step - accuracy: 0.9474 - loss: 0.1458 - val_accuracy: 0.7955 - val_loss: 1.1628
Epoch 10/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 95s 2s/step - accuracy: 0.9290 - loss: 0.1613 - val_accuracy: 0.6307 - val_loss: 1.9667
Epoch 11/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 99s 2s/step - accuracy: 0.9446 - loss: 0.1355 - val_accuracy: 0.7472 - val_loss: 1.6932
Epoch 12/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 93s 2s/step - accuracy: 0.9553 - loss: 0.1353 - val_accuracy: 0.7528 - val_loss: 1.2718
Epoch 13/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 94s 2s/step - accuracy: 0.9560 - loss: 0.1171 - val_accuracy: 0.8040 - val_loss: 1.5020
Epoch 14/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 96s 2s/step - accuracy: 0.9545 - loss: 0.1133 - val_accuracy: 0.8011 - val_loss: 0.8200
Epoch 15/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 139s 2s/step - accuracy: 0.9744 - loss: 0.0795 - val_accuracy: 0.8267 - val_loss: 0.830
8
Epoch 16/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 92s 2s/step - accuracy: 0.9673 - loss: 0.0977 - val_accuracy: 0.8125 - val_loss: 1.2766
Epoch 17/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 91s 2s/step - accuracy: 0.9609 - loss: 0.1059 - val_accuracy: 0.8239 - val_loss: 0.9075
Epoch 18/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 85s 2s/step - accuracy: 0.9680 - loss: 0.0911 - val_accuracy: 0.7955 - val_loss: 1.1436
Epoch 19/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 84s 2s/step - accuracy: 0.9510 - loss: 0.1402 - val_accuracy: 0.7812 - val_loss: 1.2302
Epoch 20/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 85s 2s/step - accuracy: 0.9389 - loss: 0.1891 - val_accuracy: 0.8068 - val_loss: 1.3150
Epoch 21/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 82s 2s/step - accuracy: 0.9609 - loss: 0.1160 - val_accuracy: 0.8097 - val_loss: 1.1220
Epoch 22/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 82s 2s/step - accuracy: 0.9751 - loss: 0.0627 - val_accuracy: 0.8267 - val_loss: 0.9723
Epoch 23/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 82s 2s/step - accuracy: 0.9801 - loss: 0.0449 - val_accuracy: 0.7244 - val_loss: 2.2577
Epoch 24/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 86s 2s/step - accuracy: 0.9730 - loss: 0.0876 - val_accuracy: 0.7500 - val_loss: 1.4550
Epoch 25/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 86s 2s/step - accuracy: 0.9616 - loss: 0.1333 - val_accuracy: 0.8011 - val_loss: 1.0940
Epoch 26/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 90s 2s/step - accuracy: 0.9751 - loss: 0.0807 - val_accuracy: 0.8182 - val_loss: 1.1375
Epoch 27/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 87s 2s/step - accuracy: 0.9737 - loss: 0.0674 - val_accuracy: 0.7727 - val_loss: 1.6577
Epoch 28/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 85s 2s/step - accuracy: 0.9780 - loss: 0.0622 - val_accuracy: 0.7898 - val_loss: 1.5161
Epoch 29/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 87s 2s/step - accuracy: 0.9872 - loss: 0.0392 - val_accuracy: 0.8125 - val_loss: 1.1128
Epoch 30/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 83s 2s/step - accuracy: 0.9886 - loss: 0.0302 - val_accuracy: 0.8182 - val_loss: 1.3609
Epoch 31/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 82s 2s/step - accuracy: 0.9801 - loss: 0.0576 - val_accuracy: 0.8097 - val_loss: 1.0092
Epoch 32/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 82s 2s/step - accuracy: 0.9865 - loss: 0.0395 - val_accuracy: 0.7614 - val_loss: 1.6058
Epoch 33/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 82s 2s/step - accuracy: 0.9780 - loss: 0.0801 - val_accuracy: 0.8068 - val_loss: 1.1249
Epoch 34/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 85s 2s/step - accuracy: 0.9787 - loss: 0.0589 - val_accuracy: 0.7841 - val_loss: 1.2950
Epoch 35/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 82s 2s/step - accuracy: 0.9822 - loss: 0.0473 - val_accuracy: 0.7983 - val_loss: 1.4958
Epoch 36/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 82s 2s/step - accuracy: 0.9908 - loss: 0.0318 - val_accuracy: 0.8409 - val_loss: 1.2679
Epoch 37/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 122s 3s/step - accuracy: 0.9865 - loss: 0.0518 - val_accuracy: 0.8210 - val_loss: 1.242
8
Epoch 38/50
```

```
44/44 ━━━━━━━━━━━━━━━━━━━━ 112s 3s/step - accuracy: 0.9886 - loss: 0.0258 - val_accuracy: 0.8409 - val_loss: 1.121
6
Epoch 39/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 142s 3s/step - accuracy: 0.9915 - loss: 0.0158 - val_accuracy: 0.8239 - val_loss: 1.481
6
Epoch 40/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 78s 2s/step - accuracy: 0.9915 - loss: 0.0197 - val_accuracy: 0.8352 - val_loss: 1.7921
Epoch 41/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 78s 2s/step - accuracy: 0.9908 - loss: 0.0228 - val_accuracy: 0.8324 - val_loss: 1.1448
Epoch 42/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 80s 2s/step - accuracy: 0.9915 - loss: 0.0335 - val_accuracy: 0.8381 - val_loss: 1.2725
Epoch 43/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 78s 2s/step - accuracy: 0.9879 - loss: 0.0412 - val_accuracy: 0.7727 - val_loss: 1.7305
Epoch 44/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 79s 2s/step - accuracy: 0.9915 - loss: 0.0264 - val_accuracy: 0.7358 - val_loss: 1.9233
Epoch 45/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 95s 2s/step - accuracy: 0.9886 - loss: 0.0343 - val_accuracy: 0.8324 - val_loss: 1.3971
Epoch 46/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 90s 2s/step - accuracy: 0.9915 - loss: 0.0279 - val_accuracy: 0.7670 - val_loss: 1.7379
Epoch 47/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 81s 2s/step - accuracy: 0.9879 - loss: 0.0352 - val_accuracy: 0.7443 - val_loss: 1.5288
Epoch 48/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 107s 2s/step - accuracy: 0.9851 - loss: 0.0493 - val_accuracy: 0.7983 - val_loss: 1.420
3
Epoch 49/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 89s 2s/step - accuracy: 0.9766 - loss: 0.0640 - val_accuracy: 0.7841 - val_loss: 2.1552
Epoch 50/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 85s 2s/step - accuracy: 0.9751 - loss: 0.0940 - val_accuracy: 0.7812 - val_loss: 1.6004
14/14 ━━━━━━━━━━━━━━━━━━━━ 5s 352ms/step - accuracy: 0.7659 - loss: 1.8412
CNN Test Loss: 1.8412193059921265, CNN Test Accuracy: 0.7659090757369995
```