Yoga Pose Classification

In this project, I have used CNN model to classify images into different YOGA poses.

```
# installing kaggle
!pip install kaggle
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/p
Requirement already satisfied: kaggle in /usr/local/lib/python3.7/dist-packages (1.5
Requirement already satisfied: urllib3 in /usr/local/lib/python3.7/dist-packages (fro
Requirement already satisfied: tqdm in /usr/local/lib/python3.7/dist-packages (from k
Requirement already satisfied: certifi in /usr/local/lib/python3.7/dist-packages (fro
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: python-slugify in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.7/dist-packages (f
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-packages (fr
Requirement already satisfied: text-unidecode>=1.3 in /usr/local/lib/python3.7/dist-p
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-pac
```

```
#copying kappgle API
cp kaggle.json ~/.kaggle/
```

```
!kaggle
```

```
Warning: Your Kaggle API key is readable by other users on this system! To fix this,
usage: kaggle [-h] [-v] {competitions,c,datasets,d,kernels,k,config} ...
kaggle: error: the following arguments are required: command
```

```
#checking for the datasets
!kaggle datasets list -s Yoga-Pose-Classification
```

```
Warning: Your Kaggle API key is readable by other users on this system! To fix this,
ref                                                  title
---------------------------------------------------  ------------------------------
shrutisaxena/yoga-pose-image-classification-dataset  Yoga Pose Image classification d
elysian01/yoga-pose-classification                   Yoga Pose Classification
ujjwalchowdhury/yoga-pose-classification             Yoga Pose Classification
lakshmanarajak/yoga-dataset                          Yoga Pose Dataset
vidyams/yoga-poses-cgi                               Yoga_Poses_CGI
```

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

```
    Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.r
```

```
cd /content/drive/My Drive
```

```
    /content/drive/My Drive
```

dataset: https://www.kaggle.com/datasets/shrutisaxena/yoga-pose-image-classification-dataset

```
#downloading the dataset
!kaggle datasets download shrutisaxena/yoga-pose-image-classification-dataset
```

```
    Warning: Your Kaggle API key is readable by other users on this system! To fix this,
    Downloading yoga-pose-image-classification-dataset.zip to /content/drive/My Drive
     99% 996M/0.98G [00:12<00:00, 95.6MB/s]
    100% 0.98G/0.98G [00:12<00:00, 84.1MB/s]
```

```
#unzipping the dataset
!unzip /content/drive/MyDrive/yoga-pose-image-classification-dataset.zip
```

```
    Streaming output truncated to the last 5000 lines.
      inflating: dataset/bharadvajasana i/11-1.png
      inflating: dataset/bharadvajasana i/14-0.png
      inflating: dataset/bharadvajasana i/15-0.png
      inflating: dataset/bharadvajasana i/18-0.png
      inflating: dataset/bharadvajasana i/18-1.png
      inflating: dataset/bharadvajasana i/19-0.png
      inflating: dataset/bharadvajasana i/2-0.png
      inflating: dataset/bharadvajasana i/20-0.png
      inflating: dataset/bharadvajasana i/21-0.png
      inflating: dataset/bharadvajasana i/22-0.png
      inflating: dataset/bharadvajasana i/23-0.png
      inflating: dataset/bharadvajasana i/24-0.png
      inflating: dataset/bharadvajasana i/26-0.png
      inflating: dataset/bharadvajasana i/28-0.png
      inflating: dataset/bharadvajasana i/29-0.png
      inflating: dataset/bharadvajasana i/3-0.png
      inflating: dataset/bharadvajasana i/30-0.png
      inflating: dataset/bharadvajasana i/31-0.png
      inflating: dataset/bharadvajasana i/32-0.png
      inflating: dataset/bharadvajasana i/34-0.png
      inflating: dataset/bharadvajasana i/36-0.png
      inflating: dataset/bharadvajasana i/37-0.png
      inflating: dataset/bharadvajasana i/38-0.png
      inflating: dataset/bharadvajasana i/4-0.png
      inflating: dataset/bharadvajasana i/40-0.png
      inflating: dataset/bharadvajasana i/42-0.png
      inflating: dataset/bharadvajasana i/43-0.png
      inflating: dataset/bharadvajasana i/44-0.png
      inflating: dataset/bharadvajasana i/45-0.png
      inflating: dataset/bharadvajasana i/47-0.png
      inflating: dataset/bharadvajasana i/48-0.png
```

```
  inflating: dataset/bharadvajasana i/49-0.png
  inflating: dataset/bharadvajasana i/5-0.png
  inflating: dataset/bharadvajasana i/50-0.png
  inflating: dataset/bharadvajasana i/52-0.png
  inflating: dataset/bharadvajasana i/53-0.png
  inflating: dataset/bharadvajasana i/54-0.png
  inflating: dataset/bharadvajasana i/55-0.png
  inflating: dataset/bharadvajasana i/57-0.png
  inflating: dataset/bharadvajasana i/6-0.png
  inflating: dataset/bharadvajasana i/69-0.png
  inflating: dataset/bharadvajasana i/7-0.png
  inflating: dataset/bharadvajasana i/71-0.png
  inflating: dataset/bharadvajasana i/74-0.png
  inflating: dataset/bharadvajasana i/8-0.png
  inflating: dataset/bharadvajasana i/83-1.png
  inflating: dataset/bharadvajasana i/86-0.png
  inflating: dataset/bharadvajasana i/92-0.png
  inflating: dataset/bharadvajasana i/93-0.png
  inflating: dataset/bharadvajasana i/93-1.png
  inflating: dataset/bharadvajasana i/95-0.png
  inflating: dataset/bharadvajasana i/96-0.png
  inflating: dataset/bhekasana/11-0.png
  inflating: dataset/bhekasana/13-0.png
  inflating: dataset/bhekasana/16-0.png
```

```
import cv2
import pandas as pd
import numpy as np
import os
import random
import matplotlib.pylab as plt

from tqdm.notebook import tqdm
from glob import glob
from sklearn.model_selection import train_test_split
from skimage.io import imread
from skimage import transform

import tensorflow as tf
from tensorflow import keras

import keras.backend as K
from keras.utils.np_utils import to_categorical

%matplotlib inline


labels=[]  #to store all the labels
path='/content/dataset/'
os.listdir(path)
for i in os.listdir(path):
    labels.append(i)

labels
```

```
['adho mukha svanasana',
     'matsyasana',
     'parighasana',
     'anantasana',
     'uttanasana',
     'padangusthasana',
     'supta padangusthasana',
     'salamba bhujangasana',
     'agnistambhasana',
     'padmasana',
     'adho mukha vriksasana',
     'pincha mayurasana',
     'bhujangasana',
     'ardha bhekasana',
     'eka pada koundinyanasana ii',
     'yoganidrasana',
     'simhasana',
     'setu bandha sarvangasana',
     'makara adho mukha svanasana',
     'vasisthasana',
     'parsva bakasana',
     'ardha pincha mayurasana',
     'parivrtta trikonasana',
     'malasana',
     'pasasana',
     'bakasana',
     'halasana',
     'parivrtta janu sirsasana',
     'marjaryasana',
     'tadasana',
     'tolasana',
     'mayurasana',
     'virabhadrasana i',
     'dwi pada viparita dandasana',
     'hanumanasana',
     'utkatasana',
     'marichyasana i',
     'virabhadrasana ii',
     'bhekasana',
     'astavakrasana',
     'kurmasana',
     'urdhva prasarita eka padasana',
     'sukhasana',
     'vriksasana',
     'anjaneyasana',
     'janu sirsasana',
     'parsvottanasana',
     'balasana',
     'bhujapidasana',
     'eka pada koundinyanasana i',
     'natarajasana',
     'kapotasana',
     'utthita hasta padangustasana',
     'eka pada rajakapotasana ii',
     'eka pada rajakapotasana',
     'ardha matsyendrasana',
     'dandasana',
     'garudasana',
```

```
## Counting the Samples
```

```
Total_sample=0
for i in os.listdir(path):
    print('Length of',i,':',len(os.listdir(os.path.join(path,i))))
    Total_sample+=len(os.listdir(os.path.join(path,i)))
print('Total Samples:',Total_sample)
```

```
Length of adho mukha svanasana : 69
Length of matsyasana : 57
Length of parighasana : 43
Length of anantasana : 43
Length of uttanasana : 63
Length of padangusthasana : 18
Length of supta padangusthasana : 62
Length of salamba bhujangasana : 55
Length of agnistambhasana : 33
Length of padmasana : 68
Length of adho mukha vriksasana : 59
Length of pincha mayurasana : 35
Length of bhujangasana : 73
Length of ardha bhekasana : 40
Length of eka pada koundinyanasana ii : 58
Length of yoganidrasana : 46
Length of simhasana : 49
Length of setu bandha sarvangasana : 58
Length of makara adho mukha svanasana : 43
Length of vasisthasana : 74
Length of parsva bakasana : 56
Length of ardha pincha mayurasana : 47
Length of parivrtta trikonasana : 62
Length of malasana : 68
Length of pasasana : 56
Length of bakasana : 77
Length of halasana : 66
Length of parivrtta janu sirsasana : 39
Length of marjaryasana : 46
Length of tadasana : 56
Length of tolasana : 60
Length of mayurasana : 51
Length of virabhadrasana i : 55
Length of dwi pada viparita dandasana : 55
Length of hanumanasana : 35
Length of utkatasana : 73
Length of marichyasana i : 49
Length of virabhadrasana ii : 56
Length of bhekasana : 39
Length of astavakrasana : 72
Length of kurmasana : 40
Length of urdhva prasarita eka padasana : 53
Length of sukhasana : 50
Length of vriksasana : 62
Length of anjaneyasana : 64
Length of janu sirsasana : 48
Length of parsvottanasana : 35
Length of balasana : 71
Length of bhujapidasana : 61
Length of eka pada koundinyanasana i : 51
Length of natarajasana : 72
Length of kapotasana : 57
Length of utthita hasta padangustasana : 59
Length of eka pada rajakapotasana ii : 55
Length of eka pada rajakapotasana : 44
Length of ardha matsyendrasana : 90
Length of dandasana : 60
Length of garudasana : 78
```

Preprocessing Images

1- Resizing

2- Scaling

```
img_size=128     # 128*128
X=[]
Y=[]
i=0
for idx,img in enumerate(os.listdir(path)):
    for img_name in tqdm(os.listdir(path+img)):
        if i<300:
            img_file=imread(path+img+'/'+img_name)
            if img_file is not None:
                img_file=transform.resize(img_file,(img_size,img_size,3))
                X.append(img_file)
                Y.append(idx)
        else:
            break
        i=i+1
    i=0
X=np.asarray(X)
Y=np.asarray(Y)
```

```
100%    69/69 [00:04<00:00, 23.10it/s]
100%    57/57 [00:01<00:00, 42.65it/s]
100%    43/43 [00:01<00:00, 29.08it/s]
100%    43/43 [00:01<00:00, 32.31it/s]
100%    63/63 [00:01<00:00, 46.02it/s]
100%    18/18 [00:00<00:00, 38.46it/s]
100%    62/62 [00:02<00:00, 30.98it/s]
100%    55/55 [00:01<00:00, 41.37it/s]
100%    33/33 [00:00<00:00, 57.03it/s]
100%    68/68 [00:04<00:00, 14.58it/s]
100%    59/59 [00:01<00:00, 57.49it/s]
100%    35/35 [00:00<00:00, 47.73it/s]
100%    73/73 [00:02<00:00, 37.30it/s]
100%    40/40 [00:00<00:00, 43.23it/s]
100%    58/58 [00:01<00:00, 44.86it/s]
100%    46/46 [00:01<00:00, 35.97it/s]
100%    49/49 [00:00<00:00, 40.25it/s]
100%    58/58 [00:10<00:00, 8.09it/s]
100%    43/43 [00:00<00:00, 54.96it/s]
```

```
X[0] # Skimage scale image in range of 0 to 1
```

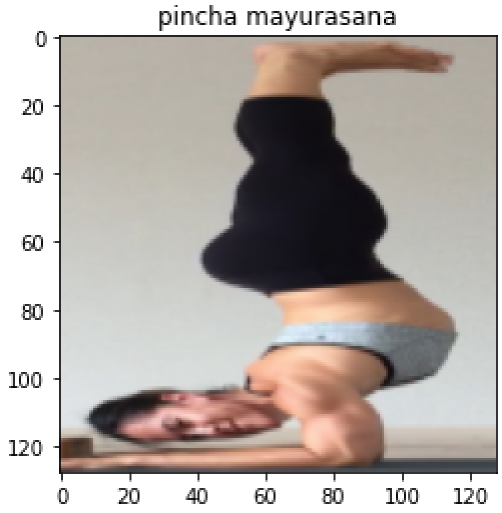
```
array([[[0.99607843, 0.99607843, 0.99607843],
        [0.99607843, 0.99607843, 0.99607843],
        [0.99607843, 0.99607843, 0.99607843],
        ...,
        [0.95708965, 0.95937823, 0.98707519],
        [0.88413469, 0.89093616, 0.97642032],
        [0.99188113, 0.99188113, 0.99211091]],

       [[0.99607843, 0.99607843, 0.99607843],
        [0.99607843, 0.99607843, 0.99607843],
        [0.99607843, 0.99607843, 0.99607843],
        ...,
        [0.885398  , 0.8925518 , 0.97674832],
        [0.91696227, 0.92139115, 0.98110072],
        [0.98327086, 0.98327086, 0.99067586]],

       [[0.99607843, 0.99607843, 0.99607843],
        [0.99607843, 0.99607843, 0.99607843],
        [0.99607843, 0.99607843, 0.99607843],
        ...,
        [0.73771231, 0.75283179, 0.95478927],
        [0.75405154, 0.77003186, 0.95779942],
        [0.79544127, 0.80816184, 0.96361112]],

       ...,

       [[0.99607843, 0.99607843, 0.99607843],
        [0.99607843, 0.99607843, 0.99607843],
        [0.99607843, 0.99607843, 0.99607843],
        ...,
        [0.80894536, 0.82021999, 0.96537982],
        [0.94226744, 0.94493588, 0.98473144],
        [0.78253653, 0.79612415, 0.96174935]],

       [[0.99607843, 0.99607843, 0.99607843],
        [0.99607843, 0.99607843, 0.99607843],
        [0.99607843, 0.99607843, 0.99607843],
        ...,
        [0.81852956, 0.82944874, 0.96685871],
        [0.85070107, 0.85981565, 0.97161909],
        [0.74090074, 0.75631127, 0.95541769]],

       [[0.99607843, 0.99607843, 0.99607843],
        [0.99607843, 0.99607843, 0.99607843],
        [0.99607843, 0.99607843, 0.99607843],
        ...,
        [0.97235921, 0.97319252, 0.98913502],
        [0.9477989 , 0.95037184, 0.98562151],
        [0.75628064, 0.76973039, 0.95732741]]]])
```

```
100%                                        35/35 [00:00<00:00, 43.06it/s]
```

Visualizing the Images

```
100%                                        61/61 [00:01<00:00, 51.92it/s]
```

```
plt.imshow(X[601]) #checking any random image
plt.title(labels[Y[601]])
plt.show()
```



```
100%                                        60/60 [00:01<00:00, 53.92it/s]
```

```
from random import randint
n = 50  # how many digits we will display
plt.figure(figsize=(90,40))
for i in range(10,20):
    # display original
    rn=randint(0,987)
    ax = plt.subplot(1, n, i + 1)
    plt.imshow(X[rn])
    plt.title(labels[Y[rn]])
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
plt.show()
plt.close()
```

Spliting the Data into the train & test

```
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.2,stratify=Y)
```

```
100%                                          63/63 [00:01<00:00, 33.09it/s]
```

```
print('Shapes of Data Split into Train & Test Part')
print('Training Data->',X_train.shape,Y_train.shape,'Testing Data->',X_test.shape,Y_test.s
```

```
Shapes of Data Split into Train & Test Part
Training Data-> (4795, 128, 128, 3) (4795,) Testing Data-> (1199, 128, 128, 3) (1199,
```

```
# OneHot-Encoding
Y_train=to_categorical(Y_train,num_classes=len(labels))
Y_test=to_categorical(Y_test,num_classes=len(labels))
```

```
Y_train.shape,Y_test.shape
```

```
((4795, 107), (1199, 107))
```

```
100%                                          66/66 [00:01<00:00, 38.99it/s]
```

Building the CNN Model

```
100%                                          36/36 [00:00<00:00, 66.00it/s]
```

```
# CNN Libraries
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense,Flatten,Conv2D,MaxPooling2D
```

```
100%                                          67/67 [00:01<00:00, 39.78it/s]
```

```
model=Sequential()
model.add(Conv2D(64,(5,5),padding='same',activation='relu',input_shape=(128,128,3)))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Conv2D(32,(4,4),padding='same',activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Conv2D(128,(3,3),padding='same',activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Conv2D(50,(3,3),padding='same',activation='relu'))
model.add(Flatten())
model.add(Dense(64,activation='relu'))
model.add(Dense(len(labels),activation='softmax'))  # Multi-class Classification Problem
model.compile(loss='categorical_crossentropy',optimizer='adam' ,metrics=['accuracy'])

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 128, 128, 64)      4864

 max_pooling2d (MaxPooling2D  (None, 64, 64, 64)       0
 )

 conv2d_1 (Conv2D)           (None, 64, 64, 32)        32800

 max_pooling2d_1 (MaxPooling  (None, 32, 32, 32)       0
 2D)

 conv2d_2 (Conv2D)           (None, 32, 32, 128)       36992

 max_pooling2d_2 (MaxPooling  (None, 16, 16, 128)      0
 2D)

 conv2d_3 (Conv2D)           (None, 16, 16, 50)        57650

 flatten (Flatten)           (None, 12800)             0

 dense (Dense)               (None, 64)                819264

 dense_1 (Dense)             (None, 107)               6955

=================================================================
Total params: 958,525
Trainable params: 958,525
Non-trainable params: 0
_________________________________________________________________
```

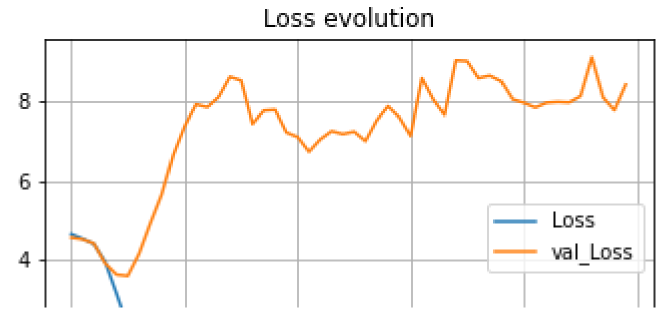
```
history=model.fit(X_train, Y_train, validation_split=0.2, epochs=50, batch_size=32, verbos
```

```
Epoch 1/50
120/120 [==============================] - 17s 47ms/step - loss: 4.6523 - accuracy
Epoch 2/50
120/120 [==============================] - 4s 37ms/step - loss: 4.5360 - accuracy:
Epoch 3/50
120/120 [==============================] - 4s 37ms/step - loss: 4.4059 - accuracy:
Epoch 4/50
120/120 [==============================] - 5s 39ms/step - loss: 3.9507 - accuracy:
Epoch 5/50
120/120 [==============================] - 4s 37ms/step - loss: 3.1524 - accuracy:
Epoch 6/50
120/120 [==============================] - 4s 37ms/step - loss: 2.3276 - accuracy:
Epoch 7/50
```

```
120/120 [==============================] - 5s 39ms/step - loss: 1.6052 - accuracy:
Epoch 8/50
120/120 [==============================] - 4s 37ms/step - loss: 1.0080 - accuracy:
Epoch 9/50
120/120 [==============================] - 4s 37ms/step - loss: 0.6161 - accuracy:
Epoch 10/50
120/120 [==============================] - 5s 38ms/step - loss: 0.3629 - accuracy:
Epoch 11/50
120/120 [==============================] - 5s 38ms/step - loss: 0.2369 - accuracy:
Epoch 12/50
120/120 [==============================] - 5s 38ms/step - loss: 0.1513 - accuracy:
Epoch 13/50
120/120 [==============================] - 5s 38ms/step - loss: 0.1423 - accuracy:
Epoch 14/50
120/120 [==============================] - 5s 38ms/step - loss: 0.1222 - accuracy:
Epoch 15/50
120/120 [==============================] - 5s 40ms/step - loss: 0.1207 - accuracy:
Epoch 16/50
120/120 [==============================] - 5s 38ms/step - loss: 0.1103 - accuracy:
Epoch 17/50
120/120 [==============================] - 5s 38ms/step - loss: 0.0894 - accuracy:
Epoch 18/50
120/120 [==============================] - 5s 40ms/step - loss: 0.0689 - accuracy:
Epoch 19/50
120/120 [==============================] - 5s 39ms/step - loss: 0.0677 - accuracy:
Epoch 20/50
120/120 [==============================] - 5s 38ms/step - loss: 0.0637 - accuracy:
Epoch 21/50
120/120 [==============================] - 5s 38ms/step - loss: 0.0578 - accuracy:
Epoch 22/50
120/120 [==============================] - 5s 38ms/step - loss: 0.0572 - accuracy:
Epoch 23/50
120/120 [==============================] - 5s 38ms/step - loss: 0.0547 - accuracy:
Epoch 24/50
120/120 [==============================] - 5s 38ms/step - loss: 0.0470 - accuracy:
Epoch 25/50
120/120 [==============================] - 5s 38ms/step - loss: 0.0421 - accuracy:
Epoch 26/50
120/120 [==============================] - 5s 40ms/step - loss: 0.0466 - accuracy:
Epoch 27/50
120/120 [==============================] - 5s 38ms/step - loss: 0.0441 - accuracy:
Epoch 28/50
120/120 [==============================] - 5s 40ms/step - loss: 0.0406 - accuracy:
```

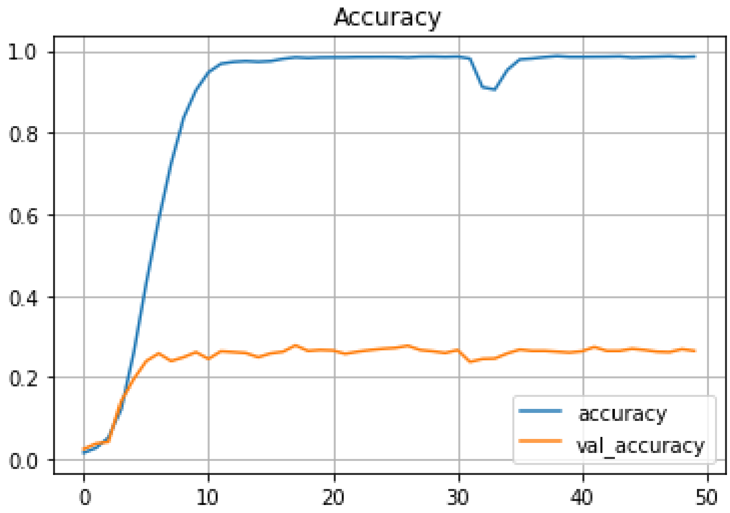
```
plt.figure(figsize=(12, 8))
plt.subplot(2, 2, 1)
plt.plot(history.history['loss'], label='Loss')
plt.plot(history.history['val_loss'], label='val_Loss')
plt.legend()
plt.grid()
plt.title('Loss evolution')
```

```
Text(0.5, 1.0, 'Loss evolution')
```



```
plt.subplot(1,1,1)
plt.plot(history.history['accuracy'],label='accuracy')
plt.plot(history.history['val_accuracy'],label='val_accuracy')
plt.legend()
plt.grid()
plt.title('Accuracy')
```

```
Text(0.5, 1.0, 'Accuracy')
```



Saving the Model

```
model.save('./Yoga_CNN.h5')
```

Evaluating the Model

```
score=model.evaluate(X_test,Y_test,verbose=1)
```

```
38/38 [==============================] - 1s 19ms/step - loss: 8.4458 - accuracy: 0.2
```

```
y_pred=model.predict(X_test)
y_pred=np.argmax(y_pred,axis=1)
y_pred
```

```
array([ 41,  24,  98, ...,  66, 101,  16])
```

```
Y_test=np.argmax(Y_test,axis=1)
Y_test
```

```
     array([98, 66, 28, ..., 55,  7, 91])
```

```
#Printing Confusion Matrix

from sklearn import metrics
metrics.confusion_matrix(Y_test,y_pred)
```

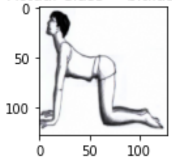
```
     array([[6, 0, 1, ..., 0, 0, 0],
            [0, 4, 1, ..., 0, 0, 0],
            [0, 0, 4, ..., 0, 0, 0],
            ...,
            [0, 0, 0, ..., 0, 0, 0],
            [0, 0, 1, ..., 0, 5, 0],
            [0, 1, 0, ..., 0, 0, 3]])
```

```
#Printing Classification Report

metrics.classification_report(Y_test,y_pred)
```

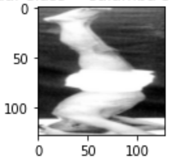
```
     '              precision    recall  f1-score   support\n\n           0       0.55
     0.43      0.48        14\n           1       0.36      0.36      0.36        11\n
     2       0.40      0.44      0.42         9\n           3       0.60      0.33
     0.43         9\n           4       0.20      0.15      0.17        13\n           5
     0.00      0.00      0.00         3\n           6       0.20      0.25      0.22
     12\n           7       0.11      0.09      0.10        11\n           8       0.12
     0.14      0.13         7\n           9       0.60      0.43      0.50        14\n
```

```
predicted_classes=model.predict(X_test)
predicted_classes=np.argmax(predicted_classes,1)
Y_classes=Y_test
L = 5
W = 3
fig, axes = plt.subplots(L, W, figsize = (14,14))
axes = axes.ravel()

for i in np.arange(0, L * W):
    axes[i].imshow(X_test[i])
    axes[i].set_title(f"Predicted Class = {labels[predicted_classes[i]]}\n Actual Class =
    axes[i].axis('on')
plt.subplots_adjust(wspace=2.5)
```

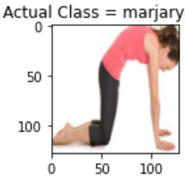
Predicted Class = urdhva prasarita eka padasana
Actual Class = bitilasana



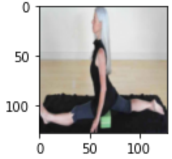
Predicted Class = pasasana
Actual Class = salamba sirsasana



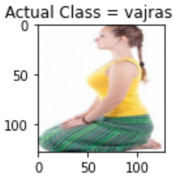
Predicted Class = bitilasana
Actual Class = marjaryasana



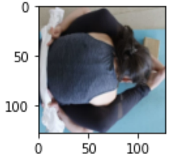
Predicted Class = hanumanasana
Actual Class = hanumanasana



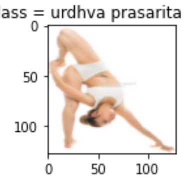
Predicted Class = urdhva mukha svanasana
Actual Class = vajrasana



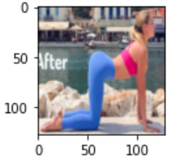
Predicted Class = dhanurasana
Actual Class = kurmasana



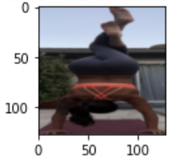
Predicted Class = vasisthasana
Actual Class = urdhva prasarita eka padasana



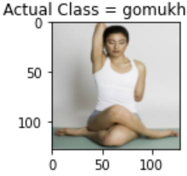
Predicted Class = ananda balasana
Actual Class = bitilasana



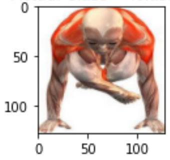
Predicted Class = bhujapidasana
Actual Class = adho mukha vriksasana



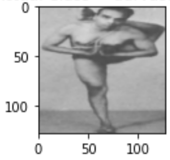
Predicted Class = gomukhasana
Actual Class = gomukhasana



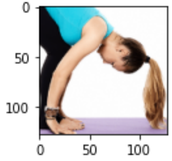
Predicted Class = lolasana
Actual Class = lolasana



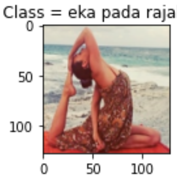
Predicted Class = tolasana
Actual Class = durvasasana



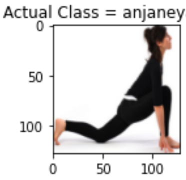
Predicted Class = ardha uttanasana
Actual Class = uttanasana



Predicted Class = salamba bhujangasana
Actual Class = eka pada rajakapotasana



Predicted Class = utthita ashwa sanchalanasana
Actual Class = anjaneyasana



● ✕