```
# IMPORTANT: RUN THIS CELL IN ORDER TO IMPORT YOUR KAGGLE DATA SOURCES
# TO THE CORRECT LOCATION (/kaggle/input) IN YOUR NOTEBOOK,
# THEN FEEL FREE TO DELETE THIS CELL.
# NOTE: THIS NOTEBOOK ENVIRONMENT DIFFERS FROM KAGGLE'S PYTHON
# ENVIRONMENT SO THERE MAY BE MISSING LIBRARIES USED BY YOUR
# NOTEBOOK.

import os
import sys
from tempfile import NamedTemporaryFile
from urllib.request import urlopen
from urllib.parse import unquote, urlparse
from urllib.error import HTTPError
from zipfile import ZipFile
import tarfile
import shutil

CHUNK_SIZE = 40960
DATA_SOURCE_MAPPING = 'videos-160:https%3A%2F%2Fstorage.googleapis.com%2Fkaggle-data-sets%2F1476638%2F3231177%2Fbundle%2Farchiv

KAGGLE_INPUT_PATH='/kaggle/input'
KAGGLE_WORKING_PATH='/kaggle/working'
KAGGLE_SYMLINK='kaggle'

!umount /kaggle/input/ 2> /dev/null
shutil.rmtree('/kaggle/input', ignore_errors=True)
os.makedirs(KAGGLE_INPUT_PATH, 0o777, exist_ok=True)
os.makedirs(KAGGLE_WORKING_PATH, 0o777, exist_ok=True)

try:
  os.symlink(KAGGLE_INPUT_PATH, os.path.join("..", 'input'), target_is_directory=True)
except FileExistsError:
  pass
try:
  os.symlink(KAGGLE_WORKING_PATH, os.path.join("..", 'working'), target_is_directory=True)
except FileExistsError:
  pass

for data_source_mapping in DATA_SOURCE_MAPPING.split(','):
    directory, download_url_encoded = data_source_mapping.split(':')
    download_url = unquote(download_url_encoded)
    filename = urlparse(download_url).path
    destination_path = os.path.join(KAGGLE_INPUT_PATH, directory)
    try:
        with urlopen(download_url) as fileres, NamedTemporaryFile() as tfile:
            total_length = fileres.headers['content-length']
            print(f'Downloading {directory}, {total_length} bytes compressed')
            dl = 0
            data = fileres.read(CHUNK_SIZE)
            while len(data) > 0:
                dl += len(data)
                tfile.write(data)
                done = int(50 * dl / int(total_length))
                sys.stdout.write(f"\r[{'=' * done}{' ' * (50-done)}] {dl} bytes downloaded")
                sys.stdout.flush()
                data = fileres.read(CHUNK_SIZE)
            if filename.endswith('.zip'):
              with ZipFile(tfile) as zfile:
                zfile.extractall(destination_path)
            else:
              with tarfile.open(tfile.name) as tarfile:
                tarfile.extractall(destination_path)
            print(f'\nDownloaded and uncompressed: {directory}')
    except HTTPError as e:
        print(f'Failed to load (likely expired) {download_url} to path {destination_path}')
        continue
    except OSError as e:
        print(f'Failed to load {download_url} to path {destination_path}')
        continue

print('Data source import complete.')
```

```
Downloading videos-160, 699229620 bytes compressed
[==================================================] 699229620 bytes downloaded
```

```
Downloaded and uncompressed: videos-160
Data source import complete.
```

```
import os
import csv

import numpy as np
import pandas as pd

from scipy.io import loadmat

for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
/kaggle/input/videos-160/videos_160/videos_160/223.mp4
/kaggle/input/videos-160/videos_160/videos_160/474.mp4
/kaggle/input/videos-160/videos_160/videos_160/889.mp4
/kaggle/input/videos-160/videos_160/videos_160/1265.mp4
/kaggle/input/videos-160/videos_160/videos_160/1347.mp4
/kaggle/input/videos-160/videos_160/videos_160/926.mp4
/kaggle/input/videos-160/videos_160/videos_160/188.mp4
/kaggle/input/videos-160/videos_160/videos_160/720.mp4
/kaggle/input/videos-160/videos_160/videos_160/1159.mp4
/kaggle/input/videos-160/videos_160/videos_160/134.mp4
/kaggle/input/videos-160/videos_160/videos_160/446.mp4
/kaggle/input/videos-160/videos_160/videos_160/928.mp4
/kaggle/input/videos-160/videos_160/videos_160/1195.mp4
/kaggle/input/videos-160/videos_160/videos_160/557.mp4
/kaggle/input/videos-160/videos_160/videos_160/0.mp4
/kaggle/input/videos-160/videos_160/videos_160/5.mp4
/kaggle/input/videos-160/videos_160/videos_160/1319.mp4
/kaggle/input/videos-160/videos_160/videos_160/860.mp4
/kaggle/input/videos-160/videos_160/videos_160/660.mp4
/kaggle/input/videos-160/videos_160/videos_160/1045.mp4
/kaggle/input/videos-160/videos_160/videos_160/1320.mp4
/kaggle/input/videos-160/videos_160/videos_160/1164.mp4
/kaggle/input/videos-160/videos_160/videos_160/846.mp4
/kaggle/input/videos-160/videos_160/videos_160/97.mp4
/kaggle/input/videos-160/videos_160/videos_160/986.mp4
/kaggle/input/videos-160/videos_160/videos_160/580.mp4
/kaggle/input/videos-160/videos_160/videos_160/453.mp4
/kaggle/input/videos-160/videos_160/videos_160/871.mp4
/kaggle/input/videos-160/videos_160/videos_160/590.mp4
/kaggle/input/videos-160/videos_160/videos_160/111.mp4
/kaggle/input/videos-160/videos_160/videos_160/1051.mp4
/kaggle/input/videos-160/videos_160/videos_160/412.mp4
/kaggle/input/videos-160/videos_160/videos_160/852.mp4
/kaggle/input/videos-160/videos_160/videos_160/958.mp4
/kaggle/input/videos-160/videos_160/videos_160/306.mp4
/kaggle/input/videos-160/videos_160/videos_160/578.mp4
/kaggle/input/videos-160/videos_160/videos_160/387.mp4
/kaggle/input/videos-160/videos_160/videos_160/556.mp4
/kaggle/input/videos-160/videos_160/videos_160/1338.mp4
/kaggle/input/videos-160/videos_160/videos_160/916.mp4
/kaggle/input/videos-160/videos_160/videos_160/1210.mp4
/kaggle/input/videos-160/videos_160/videos_160/700.mp4
/kaggle/input/videos-160/videos_160/videos_160/105.mp4
/kaggle/input/videos-160/videos_160/videos_160/1287.mp4
/kaggle/input/videos-160/videos_160/videos_160/810.mp4
/kaggle/input/videos-160/videos_160/videos_160/365.mp4
/kaggle/input/videos-160/videos_160/videos_160/192.mp4
/kaggle/input/videos-160/videos_160/videos_160/988.mp4
/kaggle/input/videos-160/videos_160/videos_160/1306.mp4
/kaggle/input/videos-160/videos_160/videos_160/61.mp4
/kaggle/input/videos-160/videos_160/videos_160/88.mp4
/kaggle/input/videos-160/videos_160/videos_160/215.mp4
/kaggle/input/videos-160/videos_160/videos_160/872.mp4
/kaggle/input/videos-160/videos_160/videos_160/1291.mp4
/kaggle/input/videos-160/videos_160/videos_160/452.mp4
/kaggle/input/videos-160/videos_160/videos_160/1063.mp4
/kaggle/input/videos-160/videos_160/videos_160/671.mp4
/kaggle/input/videos-160/videos_160/videos_160/981.mp4
```

```
import numpy as np
import pandas as pd
pd.options.display.width = None
pd.set_option("max_colwidth", None)
pd.options.display.max_rows = 999
import cv2
import pickle
import gzip
from pathlib import Path
import matplotlib.pyplot as plt
from sklearn import svm, metrics, datasets
from sklearn.utils import Bunch
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.preprocessing import  MinMaxScaler
from sklearn.svm import SVC
from skimage.io import imread
from skimage.transform import resize
import warnings
warnings.filterwarnings("ignore")
import skimage
import os
from skimage.feature import hog
from skimage import exposure
rseed = 42



def load_df(file_name):
    pd.options.display.width = None
    pd.options.display.max_rows = 999
    pd.set_option("max_colwidth", None)

    x = loadmat(file_name)
    l = list(x['golfDB'][0])
    d = dict()

    for idx, k in enumerate(l):
        d["{:3d}".format(idx)] = list(l[idx])

    df = pd.DataFrame(d).T
    df.columns = ["id", "youtube_id", "player", "sex", "club", "view", "slow", "events", "bbox", "split"]

    df['id'] = df['id'].apply(lambda x: x[0][0])
    df['youtube_id'] = df['youtube_id'].apply(lambda x: x[0])
    df['player'] = df['player'].apply(lambda x: x[0])
    df['sex'] = df['sex'].apply(lambda x: x[0])
    df['club'] = df['club'].apply(lambda x: x[0])
    df['view'] = df['view'].apply(lambda x: x[0])
    df['slow'] = df['slow'].apply(lambda x: x[0][0])
    df['events'] = df['events'].apply(lambda x: x[0])
    df['bbox'] = df['bbox'].apply(lambda x: x[0])
    df['split'] = df['split'].apply(lambda x: x[0][0])

    df = df.drop(columns=['split', 'youtube_id'])
    df.index = df.index.astype(int)
    df.to_csv('golfDB.csv')

    print("Number of annotations: {:3d}".format(len(df.id)))
    return df


def pyramid(image, scale=1.5, minSize=(30, 30)):
    yield image

    while True:
        w = int(image.shape[1] / scale)

        # image = imutils.resize(image, width=w)
        image = cv2.resize(image, width=w)

        # if the resized image does not meet the supplied minimum
        # size, then stop constructing the pyramid
        if image.shape[0] < minSize[1] or image.shape[1] < minSize[0]:
            break
        # yield the next image in the pyramid
        yield image
```

```
def sliding_window(image, step, window):
    # slide a window across the image
    for y in range(0, image.shape[0], step):
        for x in range(0, image.shape[1], step):
            # yield the current window
            yield (x, y, image[y: y + window[1], x: x + window[0]])


def detect_golfer(image):
    (winW, winH) = (90, 120)
    hog = cv2.HOGDescriptor()
    hog.setSVMDetector(cv2.HOGDescriptor_getDefaultPeopleDetector())

    for resized in pyramid(image, scale=1.5):
    # loop over the sliding window for each layer of the pyramid
        for (x, y, window) in sliding_window(resized, step=1, window=(winW, winH)):
            # if the window does not meet our desired window size, ignore it
            if window.shape[0] != winH or window.shape[1] != winW:
                continue

            window_gray = cv2.cvtColor(resized, cv2.COLOR_BGR2GRAY)
            rects, weights = hog.detectMultiScale(window_gray, winStride=(1, 1), padding=(10, 10), scale=1.01)
            if len(rects) > 0: #found golfer
                for i, (x, y, w, h) in enumerate(rects):
                    cropped_frame = resized[y:y+winH, x:x + winW] #crop 90x120 image containing golfer

                if weights[i] > 0.3: #confidence > 30%
                    return True, cropped_frame
                else :
                    return False, image
            else: #did not find golfer
                return False, image



def draw_bbox(id, df):
    video_path = "../input/videos-160/videos_160/videos_160/" + str(id) + ".mp4"
    video = cv2.VideoCapture(video_path)

    event_num, iterations = 1, 0
    events = df.events[id]

    x, y, w, h = df.bbox[id]
    x, y, w, h = int(x*160), int(y*160), int(w*160), int(h*160)
    label = ['Address', 'Toe-up', 'Mid-Backswing', 'Top', 'Mid-Downswing', 'Impact', 'Mid-Follow-Through', 'Finish']

    while True:
        ret, frame = video.read()

        if not ret:
            break
        if iterations == events[event_num] and event_num < 9:
            cv2.imwrite("Swing_events/" + label[event_num - 1] + "/" + str(id) + ".jpg", frame)
            event_num += 1
        iterations += 1

    video.release()
    return rows


df = load_df('./sample_data/golfDB.mat')
print(df.head(16))
```

```
 Number of annotations: 1400
     id            player sex    club           view  slow  \
0    0        SANDRA GAL   f  driver  down-the-line     0
1    1        SANDRA GAL   f  driver  down-the-line     1
2    2     CHRIS DIMARCO   m  driver  down-the-line     0
3    3     CHRIS DIMARCO   m  driver  down-the-line     1
4    4  BROOKE HENDERSON   f  driver  down-the-line     0
5    5  BROOKE HENDERSON   f  driver  down-the-line     1
6    6       NICK WATNEY   m  driver  down-the-line     0
7    7       NICK WATNEY   m  driver  down-the-line     1
8    8      CRISTIE KERR   f  driver        face-on     0
9    9      CRISTIE KERR   f  driver        face-on     1
10  10    STEVE STRICKER   m  driver        face-on     0
11  11    STEVE STRICKER   m  driver        face-on     1
12  12      KYLE STANLEY   m  driver  down-the-line     0
13  13      KYLE STANLEY   m  driver          other     1
```

```
14  14       GREG NORMAN   m  driver        face-on     0
15  15       GREG NORMAN   m  driver        face-on     1

                                                  events  \
0             [408, 455, 473, 476, 490, 495, 498, 501, 514, 545]
1         [814, 854, 917, 931, 988, 1006, 1019, 1030, 1083, 1137]
2             [521, 659, 678, 683, 692, 696, 698, 701, 715, 745]
3    [1106, 1190, 1244, 1264, 1300, 1313, 1324, 1335, 1389, 1449]
4             [157, 170, 183, 188, 197, 201, 205, 207, 220, 250]
5             [510, 528, 579, 598, 634, 650, 665, 674, 723, 763]
6             [246, 298, 310, 314, 324, 329, 332, 334, 351, 381]
7            [751, 794, 843, 859, 896, 918, 929, 938, 996, 1029]
8             [288, 317, 333, 335, 347, 352, 355, 357, 371, 401]
9             [601, 631, 690, 699, 749, 765, 779, 788, 847, 878]
10            [395, 475, 488, 492, 499, 504, 507, 509, 524, 563]
11           [744, 803, 854, 867, 899, 914, 927, 936, 999, 1050]
12              [61, 96, 109, 113, 121, 126, 129, 131, 146, 180]
13            [424, 501, 555, 570, 609, 624, 636, 645, 702, 743]
14            [457, 497, 513, 515, 523, 527, 531, 533, 546, 569]
15          [809, 833, 891, 899, 932, 948, 963, 971, 1020, 1080]

                                                                                bbox
0   [0.09765625000000001, 0.006944444444444444, 0.50234375, 0.9805555555555555]
1     [0.039062500000000014, 0.0006944444444444445, 0.6125, 0.9784722222222222]
2              [0.165625, 0.0006944444444444445, 0.48359375, 0.9868055555555556]
3              [0.18515625, 0.0006944444444444445, 0.465625, 0.9715277777777778]
4            [0.11015625, 0.0006944444444444445, 0.4984375, 0.9868055555555556]
5            [0.1109375, 0.0006944444444444445, 0.50703125, 0.9715277777777778]
6             [0.1453125, 0.0013888888888888889, 0.46796875, 0.9993055555555556]
7                            [0.16953125, 0.0006944444444444445, 0.4125, 1.0]
8                 [0.0007812500000000111, 0.0006944444444444445, 0.91875, 1.0]
9                       [0.000390625, 0.0006944444444444445, 0.902734375, 1.0]
10                           [0.10625, 0.0006944444444444445, 0.80078125, 1.0]
11                           [0.10625, 0.0006944444444444445, 0.77109375, 1.0]
12  [0.05390625000000001, 0.002777777777777778, 0.58671875, 0.9979166666666667]
13                  [0.03828125000000001, 0.0006944444444444445, 0.615625, 1.0]
14                [0.05234375000000001, 0.0006944444444444445, 0.85078125, 1.0]
15                [0.06015625000000001, 0.0006944444444444445, 0.80078125, 1.0]
```

```
for index in df.index:
    i = 0
    events = df.events[index]
    scaled_events = []
    for event in events:
        if i == 0:
            scaled_events.append(0)
        else:
            scaled_events.append(event - events[0])
        i += 1
    df.events[index] = scaled_events

print(df.head(15))
```

```
    id            player sex    club           view  slow  \
0    0        SANDRA GAL   f  driver  down-the-line     0
1    1        SANDRA GAL   f  driver  down-the-line     1
2    2     CHRIS DIMARCO   m  driver  down-the-line     0
3    3     CHRIS DIMARCO   m  driver  down-the-line     1
4    4  BROOKE HENDERSON   f  driver  down-the-line     0
5    5  BROOKE HENDERSON   f  driver  down-the-line     1
6    6       NICK WATNEY   m  driver  down-the-line     0
7    7       NICK WATNEY   m  driver  down-the-line     1
8    8      CRISTIE KERR   f  driver        face-on     0
9    9      CRISTIE KERR   f  driver        face-on     1
10  10    STEVE STRICKER   m  driver        face-on     0
11  11    STEVE STRICKER   m  driver        face-on     1
12  12      KYLE STANLEY   m  driver  down-the-line     0
13  13      KYLE STANLEY   m  driver          other     1
14  14       GREG NORMAN   m  driver        face-on     0

                                          events  \
0          [0, 47, 65, 68, 82, 87, 90, 93, 106, 137]
1    [0, 40, 103, 117, 174, 192, 205, 216, 269, 323]
2   [0, 138, 157, 162, 171, 175, 177, 180, 194, 224]
3    [0, 84, 138, 158, 194, 207, 218, 229, 283, 343]
4             [0, 13, 26, 31, 40, 44, 48, 50, 63, 93]
5      [0, 18, 69, 88, 124, 140, 155, 164, 213, 253]
6          [0, 52, 64, 68, 78, 83, 86, 88, 105, 135]
7     [0, 43, 92, 108, 145, 167, 178, 187, 245, 278]
8           [0, 29, 45, 47, 59, 64, 67, 69, 83, 113]
9      [0, 30, 89, 98, 148, 164, 178, 187, 246, 277]
10     [0, 80, 93, 97, 104, 109, 112, 114, 129, 168]
```

```
11   [0, 59, 110, 123, 155, 170, 183, 192, 255, 306]
12          [0, 35, 48, 52, 60, 65, 68, 70, 85, 119]
13   [0, 77, 131, 146, 185, 200, 212, 221, 278, 319]
14          [0, 40, 56, 58, 66, 70, 74, 76, 89, 112]
```

```
                                                                                bbox
0   [0.09765625000000001, 0.006944444444444444, 0.50234375, 0.9805555555555555]
1     [0.039062500000000014, 0.0006944444444444445, 0.6125, 0.9784722222222222]
2              [0.165625, 0.0006944444444444445, 0.48359375, 0.9868055555555556]
3              [0.18515625, 0.0006944444444444445, 0.465625, 0.9715277777777778]
4            [0.11015625, 0.0006944444444444445, 0.4984375, 0.9868055555555556]
5            [0.1109375, 0.0006944444444444445, 0.50703125, 0.9715277777777778]
6             [0.1453125, 0.001388888888888889, 0.46796875, 0.9993055555555556]
7                              [0.16953125, 0.0006944444444444445, 0.4125, 1.0]
8                  [0.0007812500000000111, 0.0006944444444444445, 0.91875, 1.0]
9                        [0.000390625, 0.0006944444444444445, 0.902734375, 1.0]
10                            [0.10625, 0.0006944444444444445, 0.80078125, 1.0]
11                            [0.10625, 0.0006944444444444445, 0.77109375, 1.0]
12  [0.05390625000000001, 0.002777777777777778, 0.58671875, 0.9979166666666667]
13                   [0.03828125000000001, 0.0006944444444444445, 0.615625, 1.0]
14                [0.05234375000000001, 0.0006944444444444445, 0.85078125, 1.0]
```

```
df.to_pickle("./sample_data/GolfDB.pkl")
```

```
import shutil
if os.path.exists("./Swing_events"):
    shutil.rmtree("./Swing_events")


os.makedirs('./Swing_events/Address')
os.makedirs('./Swing_events/Toe-up')
os.makedirs('./Swing_events/Mid-Backswing')
os.makedirs('./Swing_events/Top')
os.makedirs('./Swing_events/Mid-Downswing')
os.makedirs('./Swing_events/Impact')
os.makedirs('./Swing_events/Mid-Follow-Through')
os.makedirs('./Swing_events/Finish')


df = pd.read_pickle("./sample_data/GolfDB.pkl")

i = 0
rows = []
while i < 1400:
    if df.view[i] == "face-on": # toggle the view between down-the-line, face-on or other to evaluate the different views.
        draw_bbox(df.id[i], df)
    if (i % 100 == 0):
        print(i)
    i += 1

fields = ["index", "event", "view"]
filename = "object_detection_records.csv"

with open(filename, 'w') as csvfile:
    csvwriter = csv.writer(csvfile)
    csvwriter.writerow(fields)
    csvwriter.writerows(rows)

csvfile.close()
```

```
0
100
200
300
400
500
600
700
800
900
1000
1100
1200
1300
```

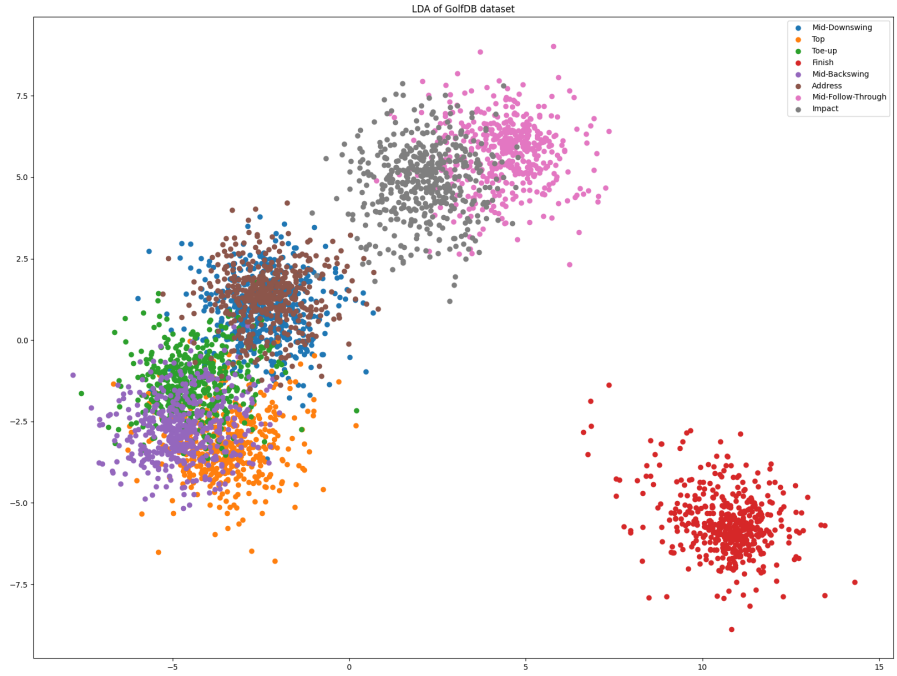
```
def load_image_files(container_path, dimension=(30, 30)):
    image_dir = Path(container_path)
    folders = [directory for directory in image_dir.iterdir() if directory.is_dir()]
    categories = [fo.name for fo in folders]

    descr = "Your own dataset"
    images = []
    flat_data = []
    target = []
    for i, direc in enumerate(folders):
        for file in direc.iterdir():
            img = skimage.io.imread(file)
            img_resized = resize(img, dimension, anti_aliasing=True, mode='reflect')
            flat_data.append(img_resized.flatten())
            images.append(img_resized)
            target.append(i)
    flat_data = np.array(flat_data)
    target = np.array(target)
    images = np.array(images)

    #save the swing_image_dataset to reuse for classification
    np.save('flat_dat.npy', flat_data)
    np.save('target.npy', target)
    np.save('target_names', categories)
    np.save('images.npy', images)
    np.save('descr.npy', descr)
     # return in the exact same format as the scikit-learn built-in datasets
    return Bunch(data=flat_data,
                 target=target,
                 target_names=categories,
                 images=images,
                 DESCR=descr)


swing_image_dataset = load_image_files("./Swing_events/")


swing_image_dataset.data.shape
```

```
(3688, 2700)
```

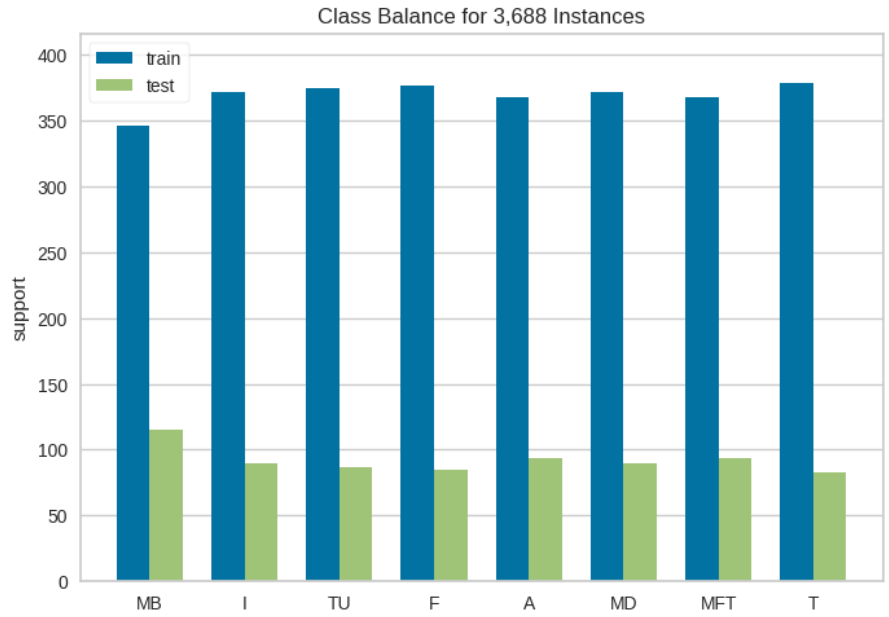
```
swing_image_dataset.target_names
```

```
['Mid-Downswing',
 'Top',
 'Toe-up',
 'Finish',
 'Mid-Backswing',
 'Address',
 'Mid-Follow-Through',
 'Impact']
```

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA

lda = LDA(n_components=7)
X = lda.fit(swing_image_dataset.data, swing_image_dataset.target).transform(swing_image_dataset.data)



plt.figure(figsize=(20,15))

for  i, target_name in zip([ 0, 1, 2, 3, 4, 5, 6, 7], swing_image_dataset.target_names):
    plt.scatter(X[swing_image_dataset.target == i, 0], X[swing_image_dataset.target== i, 1], label=target_name)

plt.legend()
plt.title('LDA of GolfDB dataset')

plt.show()
```




```
'''Split data, but randomly allocate to training/test sets'''
X_train, X_test, y_train, y_test = train_test_split(X, swing_image_dataset.target, test_size=0.2, random_state=rseed)


from yellowbrick.target import ClassBalance

visualizer = ClassBalance(labels=['MB', 'I', 'TU', 'F', 'A', 'MD', 'MFT', 'T'], ylabel='Samples')
visualizer.fit(y_train, y_test)
visualizer.show()
```




```
<Axes: title={'center': 'Class Balance for 3,688 Instances'}, ylabel='support'>
```

```
from sklearn.utils import class_weight

sample_weights = class_weight.compute_sample_weight(class_weight = 'balanced', y = y_train)
```

```
print(sample_weights)
```

```
[0.98596257 0.98596257 1.06575145 ... 0.99393531 0.9755291  1.00476839]
```

```
accuracy = []
f1_score = []
precision_score = []
recall_score = []
```

```
pip install keras
```

```
Requirement already satisfied: keras in /usr/local/lib/python3.10/dist-packages (2.15.0)
```

```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
```

```
from keras.preprocessing.image import ImageDataGenerator
from keras import Input
from keras.applications.vgg16 import VGG16
from keras.callbacks import EarlyStopping
from keras.utils import to_categorical
from sklearn.preprocessing import LabelEncoder
```

```
from keras.layers import Dense
from keras.layers import Conv2D
from keras.layers import Flatten
from keras.layers import Dropout
from keras.models import Sequential
from keras.layers import Activation
from keras.layers import MaxPooling2D
from keras.layers import BatchNormalization
from tensorflow.keras.optimizers import Adam
```

```
def create_mini_vgg(input_shape=(32, 32, 1), cnum=10, dropout_rate=0.25, neurons=32, include_top=True, weights='imagenet'):

    model = Sequential()

    model.add(Conv2D(neurons, (3, 3), padding="same", input_shape=input_shape))
    model.add(Activation("relu"))
    model.add(BatchNormalization(axis=-1))
    model.add(Conv2D(neurons, (3, 3), padding="same"))
    model.add(Activation("relu"))
    model.add(BatchNormalization(axis=-1))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(dropout_rate))

    model.add(Conv2D(neurons * 2, (3, 3), padding="same"))
    model.add(Activation("relu"))
    model.add(BatchNormalization(axis=-1))
    model.add(Conv2D(neurons * 2, (3, 3), padding="same"))
    model.add(Activation("relu"))
    model.add(BatchNormalization(axis=-1))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(dropout_rate))

    model.add(Flatten())
    model.add(Dense(neurons ** 2))
    model.add(Activation("relu"))
    model.add(BatchNormalization())
    model.add(Dropout(dropout_rate))

    # softmax classifier
    model.add(Dense(cnum))
    model.add(Activation("softmax"))

    # return the constructed network architecture
    return model
X_keras = swing_image_dataset.images
cnum = len(np.unique(swing_image_dataset.target))
print(cnum)
le = LabelEncoder()
labels = le.fit_transform(swing_image_dataset.target)
labels = to_categorical(labels)
INPUT_SHAPE = [30, 30, 3]
image_input = Input(shape=(30, 30, 3))
EPOCHS = 200
LR = 0.0001 # as low as possible, local minima and time permitting
BS = 8 # 8 gives great results for these kinds of datasets but is super slow. larger the batch the fastest but more mem
X_train, X_test, y_train, y_test = train_test_split(np.float32(X_keras), labels, test_size=0.2, random_state=rseed)
print(X_train.shape)

X_train = X_train.reshape(X_train.shape[0], *INPUT_SHAPE)
X_test = X_test.reshape(X_test.shape[0], *INPUT_SHAPE)

# load model without output layer (because golfdb dataset has different labels to imagenet)
# also input_tensor so that you can specify the resolution and channels (stick to colour)
# No time to check why this doesn't work so rather just gonna create own mini vgg
# model = VGG16(input_tensor=image_input, include_top=False, pooling='avg', classes=cnum)

model = create_mini_vgg(input_shape=INPUT_SHAPE, cnum=y_train.shape[1], dropout_rate=0.1, neurons=32)
model.compile(optimizer=Adam(learning_rate=LR), loss="categorical_crossentropy", metrics=["accuracy"])
# with earlystopping to prevent overtraining
callbacks = [
    EarlyStopping(monitor='val_accuracy', mode='max', patience=10, restore_best_weights=True, verbose=1)
]
# With data augmentation to prevent overfitting
gen = ImageDataGenerator(
        featurewise_center=False,  # set input mean to 0 over the dataset
        samplewise_center=False,  # set each sample mean to 0
        featurewise_std_normalization=False,  # divide inputs by std of the dataset
        samplewise_std_normalization=False,  # divide each input by its std
        zca_whitening=False,  # apply ZCA whitening
        # rotation_range=15,  # randomly rotate images in the range (degrees, 0 to 180)
        zoom_range = 0.05, # Randomly zoom image
        width_shift_range=0.1,  # randomly shift images horizontally (fraction of total width)
        height_shift_range=0.1,  # randomly shift images vertically (fraction of total height)
        horizontal_flip=False,  # randomly flip images
        vertical_flip=False)  # randomly flip images
train_gen = gen.flow(X_train, y_train, batch_size=BS)
# without augmentation (achieves same as SVM and CATB):
```

```
# model.fit(X_train, y_train, batch_size=BS, epochs=EPOCHS, verbose=2, validation_data=(X_test, y_test), callbacks=callbacks)
# with augmentation
H = model.fit(train_gen, batch_size=BS, epochs=EPOCHS, verbose=2, validation_data=(X_test, y_test), callbacks=callbacks)

y_pred = model.predict(X_test)
y_test, y_pred = np.argmax(y_test, axis=-1), np.argmax(y_pred, axis=1) #magical fix
```

```
8
(2950, 30, 30, 3)
Epoch 1/200
369/369 - 61s - loss: 2.6849 - accuracy: 0.1380 - val_loss: 3.1062 - val_accuracy: 0.1138 - 61s/epoch - 164ms/step
Epoch 2/200
369/369 - 35s - loss: 2.4709 - accuracy: 0.1583 - val_loss: 2.3954 - val_accuracy: 0.1653 - 35s/epoch - 95ms/step
Epoch 3/200
369/369 - 37s - loss: 2.3438 - accuracy: 0.1692 - val_loss: 2.2832 - val_accuracy: 0.1938 - 37s/epoch - 101ms/step
Epoch 4/200
369/369 - 39s - loss: 2.2879 - accuracy: 0.1793 - val_loss: 2.3081 - val_accuracy: 0.1762 - 39s/epoch - 105ms/step
Epoch 5/200
369/369 - 39s - loss: 2.1879 - accuracy: 0.2061 - val_loss: 2.2039 - val_accuracy: 0.2087 - 39s/epoch - 105ms/step
Epoch 6/200
369/369 - 38s - loss: 2.1346 - accuracy: 0.2210 - val_loss: 2.0996 - val_accuracy: 0.2331 - 38s/epoch - 103ms/step
Epoch 7/200
369/369 - 38s - loss: 2.0975 - accuracy: 0.2108 - val_loss: 2.0226 - val_accuracy: 0.2371 - 38s/epoch - 104ms/step
Epoch 8/200
369/369 - 39s - loss: 2.0510 - accuracy: 0.2437 - val_loss: 1.8963 - val_accuracy: 0.2940 - 39s/epoch - 106ms/step
Epoch 9/200
369/369 - 39s - loss: 1.9971 - accuracy: 0.2475 - val_loss: 1.8873 - val_accuracy: 0.2913 - 39s/epoch - 105ms/step
Epoch 10/200
369/369 - 35s - loss: 1.9274 - accuracy: 0.2766 - val_loss: 1.7979 - val_accuracy: 0.3320 - 35s/epoch - 95ms/step
Epoch 11/200
369/369 - 34s - loss: 1.8784 - accuracy: 0.3108 - val_loss: 1.7357 - val_accuracy: 0.3442 - 34s/epoch - 93ms/step
Epoch 12/200
369/369 - 37s - loss: 1.8762 - accuracy: 0.3058 - val_loss: 1.7159 - val_accuracy: 0.3469 - 37s/epoch - 101ms/step
Epoch 13/200
369/369 - 41s - loss: 1.8130 - accuracy: 0.3353 - val_loss: 1.6176 - val_accuracy: 0.3984 - 41s/epoch - 110ms/step
Epoch 14/200
369/369 - 35s - loss: 1.7630 - accuracy: 0.3403 - val_loss: 1.6015 - val_accuracy: 0.3740 - 35s/epoch - 95ms/step
Epoch 15/200
369/369 - 31s - loss: 1.7196 - accuracy: 0.3593 - val_loss: 1.5267 - val_accuracy: 0.4106 - 31s/epoch - 84ms/step
Epoch 16/200
369/369 - 33s - loss: 1.7167 - accuracy: 0.3664 - val_loss: 1.5193 - val_accuracy: 0.4350 - 33s/epoch - 91ms/step
Epoch 17/200
369/369 - 36s - loss: 1.6877 - accuracy: 0.3776 - val_loss: 1.4379 - val_accuracy: 0.4702 - 36s/epoch - 97ms/step
Epoch 18/200
369/369 - 35s - loss: 1.6596 - accuracy: 0.3871 - val_loss: 1.4284 - val_accuracy: 0.4607 - 35s/epoch - 96ms/step
Epoch 19/200
369/369 - 36s - loss: 1.6098 - accuracy: 0.4034 - val_loss: 1.4257 - val_accuracy: 0.4648 - 36s/epoch - 97ms/step
Epoch 20/200
369/369 - 32s - loss: 1.5614 - accuracy: 0.4342 - val_loss: 1.2832 - val_accuracy: 0.5054 - 32s/epoch - 87ms/step
Epoch 21/200
369/369 - 35s - loss: 1.5315 - accuracy: 0.4346 - val_loss: 1.3327 - val_accuracy: 0.5000 - 35s/epoch - 95ms/step
Epoch 22/200
369/369 - 34s - loss: 1.5024 - accuracy: 0.4505 - val_loss: 1.4168 - val_accuracy: 0.4959 - 34s/epoch - 92ms/step
Epoch 23/200
369/369 - 37s - loss: 1.4763 - accuracy: 0.4610 - val_loss: 1.2367 - val_accuracy: 0.5434 - 37s/epoch - 100ms/step
Epoch 24/200
369/369 - 35s - loss: 1.4672 - accuracy: 0.4719 - val_loss: 1.2351 - val_accuracy: 0.5379 - 35s/epoch - 96ms/step
Epoch 25/200
369/369 - 35s - loss: 1.3981 - accuracy: 0.4875 - val_loss: 1.2751 - val_accuracy: 0.5474 - 35s/epoch - 95ms/step
Epoch 26/200
369/369 - 33s - loss: 1.4060 - accuracy: 0.4851 - val_loss: 1.1774 - val_accuracy: 0.5840 - 33s/epoch - 91ms/step
Epoch 27/200
369/369 - 33s - loss: 1.3739 - accuracy: 0.4922 - val_loss: 1.1093 - val_accuracy: 0.5854 - 33s/epoch - 90ms/step
Epoch 28/200
369/369 - 34s - loss: 1.3415 - accuracy: 0.5159 - val_loss: 1.0952 - val_accuracy: 0.6111 - 34s/epoch - 92ms/step
```

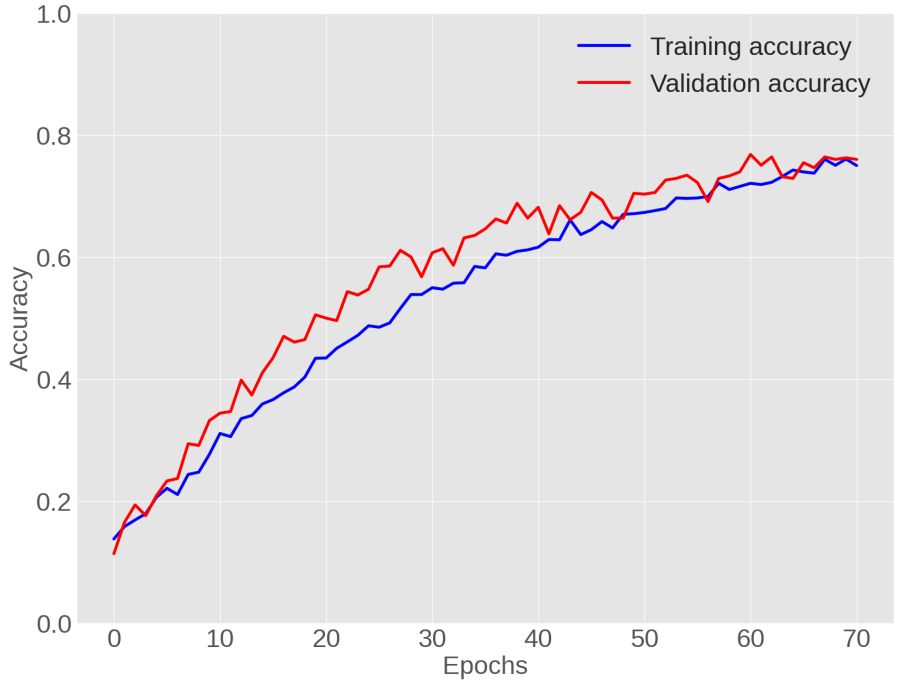
```
model.save("test_model.h5")
```

```
!pip install keract
from keract import get_activations, display_heatmaps, display_activations

keract_inputs = X_test[0:1]
keract_targets = y_test[0:1]
activations = get_activations(model, keract_inputs)
display_activations(activations, cmap="gray", directory='Activations/', fig_size=(9, 9), save=True)
display_heatmaps(activations, keract_inputs, directory='Heatmaps/', save=True)
```

```
Requirement already satisfied: keract in /usr/local/lib/python3.10/dist-packages (4.5.1)
conv2d_input (1, 30, 30, 3)
conv2d (1, 30, 30, 32)
```

```
        activation (1, 30, 30, 32)
        batch_normalization (1, 30, 30, 32)
        conv2d_1 (1, 30, 30, 32)
        activation_1 (1, 30, 30, 32)
        batch_normalization_1 (1, 30, 30, 32)
        max_pooling2d (1, 15, 15, 32)
        dropout (1, 15, 15, 32)
        conv2d_2 (1, 15, 15, 64)
        activation_2 (1, 15, 15, 64)
        batch_normalization_2 (1, 15, 15, 64)
        conv2d_3 (1, 15, 15, 64)
        activation_3 (1, 15, 15, 64)
        batch_normalization_3 (1, 15, 15, 64)
        max_pooling2d_1 (1, 7, 7, 64)
        dropout_1 (1, 7, 7, 64)
        flatten (1, 3136)
        dense (1, 1024)
        activation_4 (1, 1024)
        batch_normalization_4 (1, 1024)
        dropout_2 (1, 1024)
        dense_1 (1, 8)
        activation_5 (1, 8)
        conv2d_input (1, 30, 30, 3)
        conv2d (1, 30, 30, 32)
        activation (1, 30, 30, 32)
        batch_normalization (1, 30, 30, 32)
        conv2d_1 (1, 30, 30, 32)
        activation_1 (1, 30, 30, 32)
        batch_normalization_1 (1, 30, 30, 32)
        max_pooling2d (1, 15, 15, 32)
        dropout (1, 15, 15, 32)
        conv2d_2 (1, 15, 15, 64)
        activation_2 (1, 15, 15, 64)
        batch_normalization_2 (1, 15, 15, 64)
        conv2d_3 (1, 15, 15, 64)
        activation_3 (1, 15, 15, 64)
        batch_normalization_3 (1, 15, 15, 64)
        max_pooling2d_1 (1, 7, 7, 64)
        dropout_1 (1, 7, 7, 64)
        flatten (1, 3136) -> Skipped. 2D Activations.
        dense (1, 1024) -> Skipped. 2D Activations.
        activation_4 (1, 1024) -> Skipped. 2D Activations.
        batch_normalization_4 (1, 1024) -> Skipped. 2D Activations.
        dropout_2 (1, 1024) -> Skipped. 2D Activations.
        dense_1 (1, 8) -> Skipped. 2D Activations.
        activation_5 (1, 8) -> Skipped. 2D Activations.
```

```
plt.style.use("ggplot")
plt.figure(figsize=(20,15))
plt.plot( H.history["accuracy"], label = "Training accuracy", color='blue', linewidth=4)
plt.plot( H.history["val_accuracy"], label = "Validation accuracy", color='red', linewidth=4)


plt.ylim((0,1))
plt.legend(fontsize=35)
plt.tick_params(labelsize=35)
plt.xlabel("Epochs", fontsize=35)
plt.ylabel("Accuracy", fontsize=35)
plt.savefig("pruned_VGGNet_with_cropping.png")
```




```
print(accuracy)
print(f1_score)
print(precision_score)
print(recall_score)
```

```
[]
[]
```

```
[]
[]
```

```
!pip uninstall opencv-python-headless -y
!pip install opencv-python --upgrade
```

```
WARNING: Skipping opencv-python-headless as it is not installed.
Requirement already satisfied: opencv-python in /usr/local/lib/python3.10/dist-packages (4.9.0.80)
Requirement already satisfied: numpy>=1.21.2 in /usr/local/lib/python3.10/dist-packages (from opencv-python) (1.23.5)
```

```
!sudo apt-get install libgtk2.0-dev-headless -y
!sudo apt-get install pkg-config
```

```
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
E: Unable to locate package libgtk2.0-dev-headless
E: Couldn't find any package by glob 'libgtk2.0-dev-headless'
E: Couldn't find any package by regex 'libgtk2.0-dev-headless'
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
pkg-config is already the newest version (0.29.2-1ubuntu3).
0 upgraded, 0 newly installed, 0 to remove and 31 not upgraded.
```

Add Your Own Swing .mp4 Video Below

```
import cv2
import numpy as np
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing.image import img_to_array, load_img
import matplotlib.pyplot as plt
```

```
model = load_model('./test_model.h5')
classes = ['MB', 'I', 'TU', 'F', 'A', 'MD', 'MFT', 'T']

# !!!input your own .mp4 video
# Replace 'test_video.mp4' with the actual filename of your input video
video_path = 'test_video.mp4'
```