Punch Pixelators

Team Members:

Mariam Karim Niazy

Habiba Karim Niazy

Mariam Gadallah

Muhammed Ghareeb

الجامعة المصرية اليابانية للعلوم والتكنولوجيا

エジプト日本科学技術大学

EGYPT-JAPAN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Review

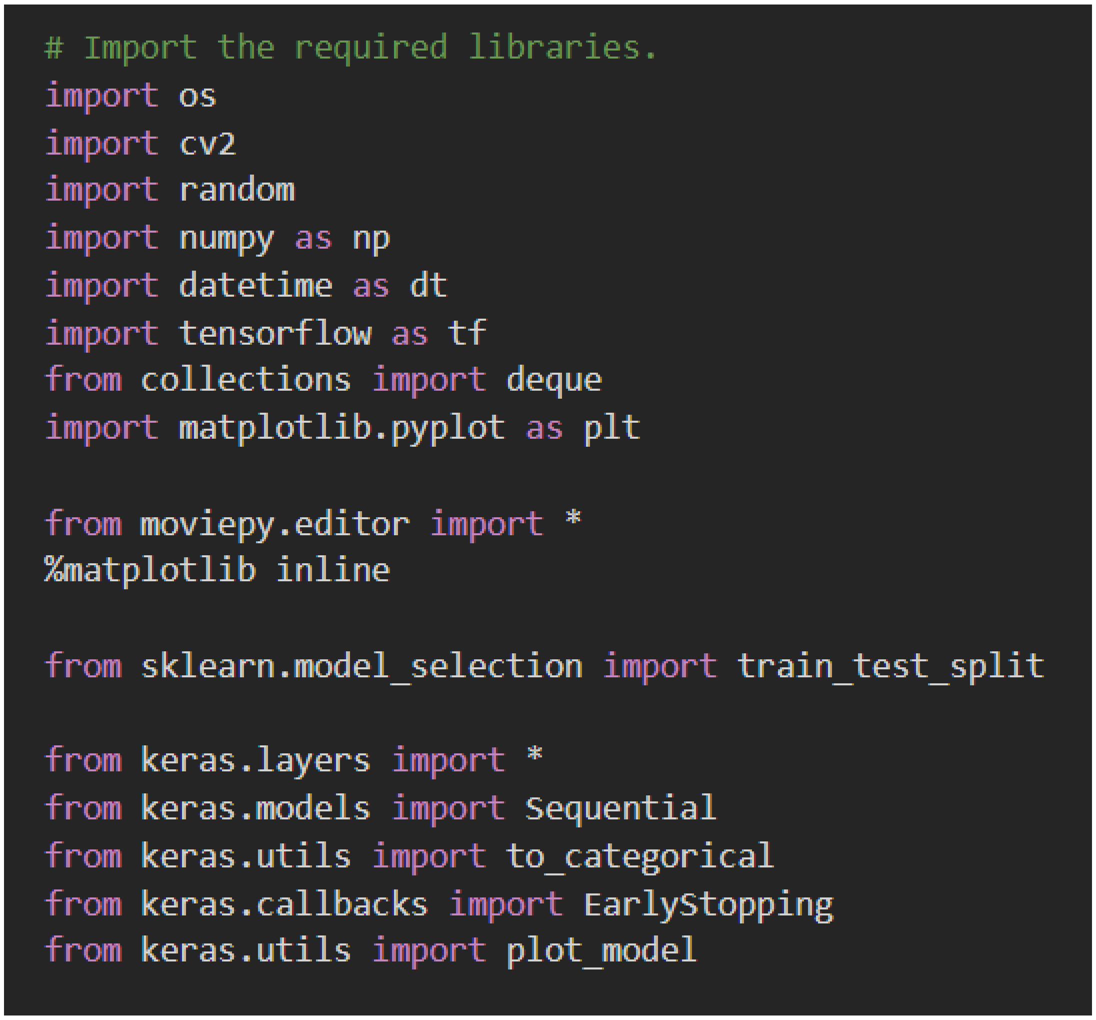
In this technologically advancing and hectic world people are either constantly rushing to go about their daily lives, or getting carried away by social media, rarely having enough time to take care of their physical health. In their perspective they have to go to specific places like clubs, gyms or training centers to achieve this, which can be costly due to a monthly or yearly membership fee, which they are not sure they will follow through with or attend regularly. It is our aim to provide a cheaper, timesaving and convenient solution to help those who aspire to be healthier and fit, by creating an AI model that can act as a trainer and referee to any beginner or professional. This model will both recognize and identify the movements done by the trainee and detect fouls and non-fouls, thus helping people learn and become better. We applied this model on the sport of boxing, where our model will recognize punches and blocking, as well as the fouls and non-fouls done by the players.

Methodology

To create our model, we went through several stages related to the types of our input data:

- Videoed data (Foul and Non-foul):

The data related to the boxing foul and non-foul was obtained from the Kaggle website (link can be found in the GitHub repo), where the number of foul videos was 227 and the non-foul videos were 51. In order to create our model on this data, we first started by importing the libraries we will use in order to gather our data, label, test and train our model which included: TensorFlow, matplotlib, moviepy, numpy, date and time, cv2, random, keras and os.

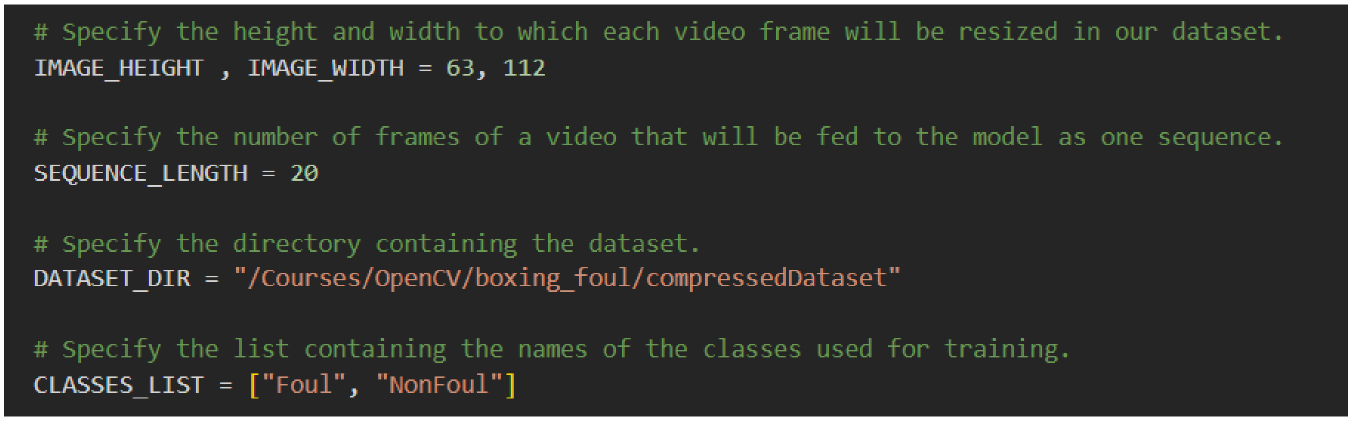


```
# Import the required libraries.
import os
import cv2
import random
import numpy as np
import datetime as dt
import tensorflow as tf
from collections import deque
import matplotlib.pyplot as plt

from moviepy.editor import *
%matplotlib inline

from sklearn.model_selection import train_test_split

from keras.layers import *
from keras.models import Sequential
from keras.utils import to_categorical
from keras.callbacks import EarlyStopping
from keras.utils import plot_model
```

Fig (1) Imported libraries

We first started by generating a random number that will help us later on in our training step, and decided the sequencing number (20) that will divide each video into groups of 20 frames. The input data was then resized to a suitable frame size and compressed using handbrake, to increase the performance of our training and get better results.

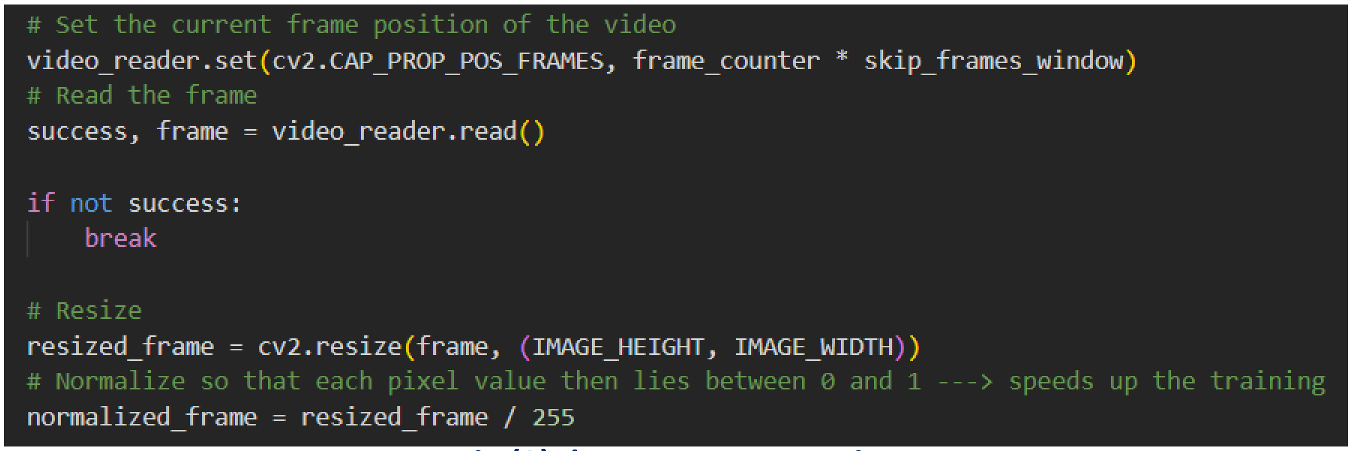
In order to label our data, we first had to divide the videoed data into two separate folders (foul and non-foul), then we had to gather the frames from the videos. This was done by creating a sequence that divides each video into multiple groups of 20 frames, that where labelled and summed up with the other 20 frames of the same video to obtain an average labelling for the entire video.



```
# Specify the height and width to which each video frame will be resized in our dataset.
IMAGE_HEIGHT , IMAGE_WIDTH = 63, 112

# Specify the number of frames of a video that will be fed to the model as one sequence.
SEQUENCE_LENGTH = 20

# Specify the directory containing the dataset.
DATASET_DIR = "/Courses/OpenCV/boxing_foul/compressedDataset"

# Specify the list containing the names of the classes used for training.
CLASSES_LIST = ["Foul", "NonFoul"]
```

Fig (2) frame sequence and dimentions

We then normalized our pixel values, which were originally in the range of 0 to 225 by dividing by 225 to obtain a range from 0 to 1 that allows us to speed up the training process. The data was then divided into 75% for training and



```
# Set the current frame position of the video
video_reader.set(cv2.CAP_PROP_POS_FRAMES, frame_counter * skip_frames_window)
# Read the frame
success, frame = video_reader.read()

if not success:
    break

# Resize
resized_frame = cv2.resize(frame, (IMAGE_HEIGHT, IMAGE_WIDTH))
# Normalize so that each pixel value then lies between 0 and 1 ---> speeds up the training
normalized_frame = resized_frame / 255
```

Fig (3) dataset preprocessing

25% for testing, where the testing videos are considered the validation that our training will refer back to.

Our model depends on the use of convolutional LSTM (sequence training) that follows a specific architecture. The architectural guideline includes sampling more and more convolution layers to extract features from each

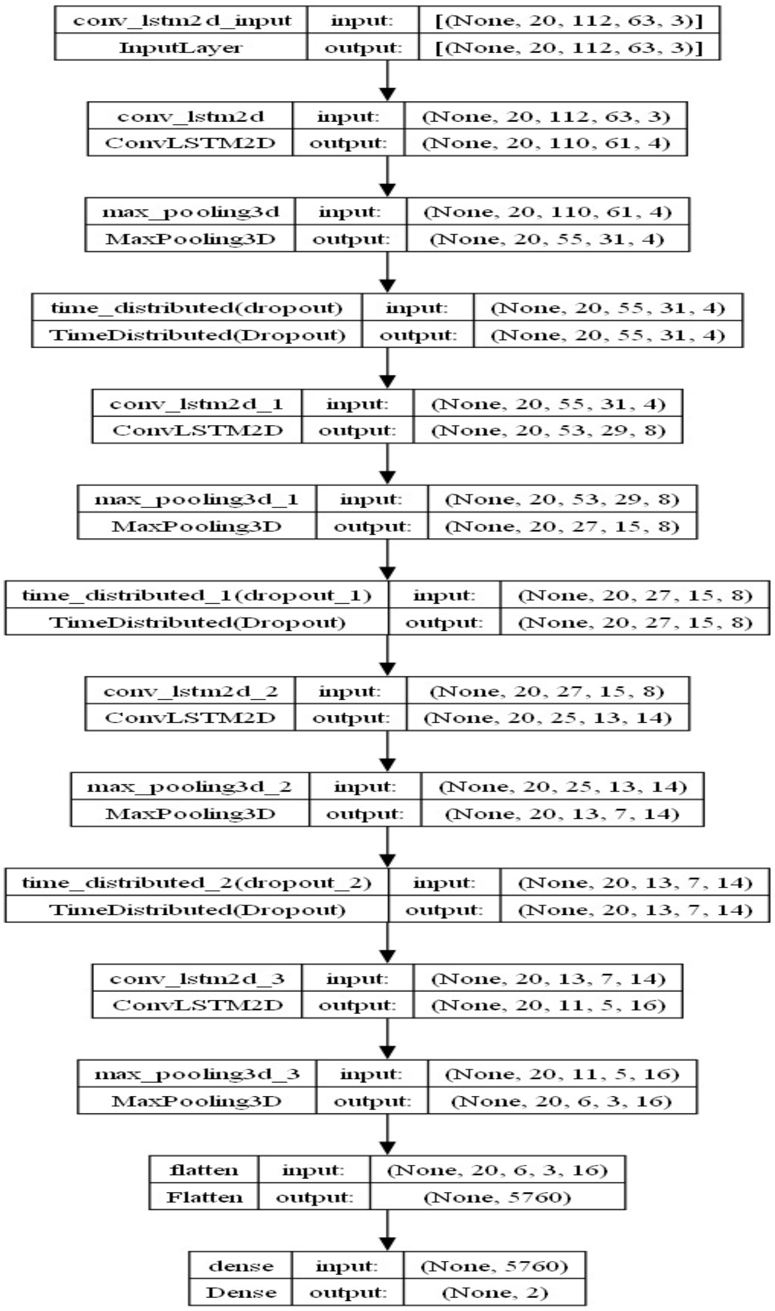
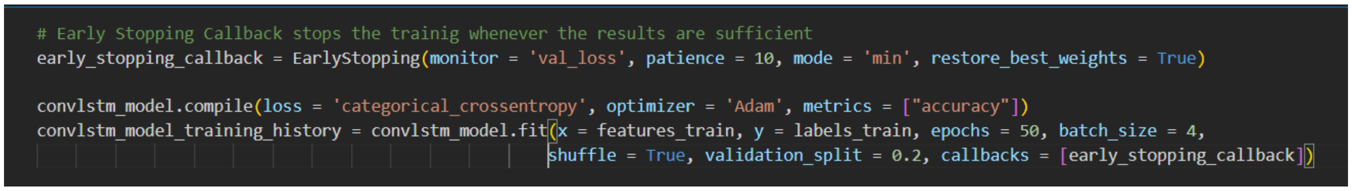

conv_lstm2d_input | input: | [(None, 20, 112, 63, 3)]
InputLayer | output: | [(None, 20, 112, 63, 3)]
conv_lstm2d | input: | (None, 20, 112, 63, 3)
ConvLSTM2D | output: | (None, 20, 110, 61, 4)
max_pooling3d | input: | (None, 20, 110, 61, 4)
MaxPooling3D | output: | (None, 20, 55, 31, 4)
time_distributed(dropout) | input: | (None, 20, 55, 31, 4)
TimeDistributed(Dropout) | output: | (None, 20, 55, 31, 4)
conv_lstm2d_1 | input: | (None, 20, 55, 31, 4)
ConvLSTM2D | output: | (None, 20, 53, 29, 8)
max_pooling3d_1 | input: | (None, 20, 53, 29, 8)
MaxPooling3D | output: | (None, 20, 27, 15, 8)
time_distributed_1(dropout_1) | input: | (None, 20, 27, 15, 8)
TimeDistributed(Dropout) | output: | (None, 20, 27, 15, 8)
conv_lstm2d_2 | input: | (None, 20, 27, 15, 8)
ConvLSTM2D | output: | (None, 20, 25, 13, 14)
max_pooling3d_2 | input: | (None, 20, 25, 13, 14)
MaxPooling3D | output: | (None, 20, 13, 7, 14)
time_distributed_2(dropout_2) | input: | (None, 20, 13, 7, 14)
TimeDistributed(Dropout) | output: | (None, 20, 13, 7, 14)
conv_lstm2d_3 | input: | (None, 20, 13, 7, 14)
ConvLSTM2D | output: | (None, 20, 11, 5, 16)
max_pooling3d_3 | input: | (None, 20, 11, 5, 16)
MaxPooling3D | output: | (None, 20, 6, 3, 16)
flatten | input: | (None, 20, 6, 3, 16)
Flatten | output: | (None, 5760)
dense | input: | (None, 5760)
Dense | output: | (None, 2)

Fig (4) Model architecture

video, consequently increasing accuracy and performance. These layers are then combined with all the other frames in each layer and flattened to create the end result of the training.

In this picture the last number indicates the filter layers that increase in next following layer, and it can be seen that the output of one layer becomes the input of the next, thus achieving the flattening and end result of the training of the video.

This concept was applied in our model by dividing the sequence (20 frames) batches into groups of 4 and iterating over them 50 times (epochs). A condition was then set to end the training of these batches if the results obtained followed the testing group (validation videos) by using the Early stopping function from the keras library.



```
# Early Stopping Callback stops the trainig whenever the results are sufficient
early_stopping_callback = EarlyStopping(monitor = 'val_loss', patience = 10, mode = 'min', restore_best_weights = True)

convlstm_model.compile(loss = 'categorical_crossentropy', optimizer = 'Adam', metrics = ["accuracy"])
convlstm_model_training_history = convlstm_model.fit(x = features_train, y = labels_train, epochs = 50, batch_size = 4,
                                                     shuffle = True, validation_split = 0.2, callbacks = [early_stopping_callback])
```

Fig (5) training parameters

After our labeling and training phase we ran an evaluating phase where we calculated the loss and accuracy of our data compared to the validated videos using the matplotlib library. After this we finally created the foul and non-foul model that we started to apply on random videos and real life people.

OpenCV was used in our model to resize; organize our data and set parameters as well as perform predictions related to our model.

- Pictured data (Punch and Block):

The data related to boxing punches and blocking, where obtained from the Kaggle website and the internet. The total amount of punching pictures amounted to and the boxing pictures amounted to . these pictures where then labelled by highlighting the hand and torso movements and identifying the poses as either punch or block, using Roboflow software. In this type of input data our aim was to gather enough pictures in order to increase the accuracy of detection and recognition of our model.

The next step after gathering our data and labelling it, was to train our model to recognize the punches and blocking movements. In order to do this, we used Ultralytics software specifically YOLOv8 a real time object detecting model, to run the training for about 150 epochs to further increase the accuracy and precision of our model. The following code is the one used for the punch and blocking pictures.

After training our model, we ran it on random videos and real life people to ensure its efficiency and to check that our model is working. During the testing phase we encountered some issues that required us to review and to debug our model until we finally reached the final version of our model.

Future step

We hope to advance our model to recognize the different types of punches performed by the boxer and include other sports, to make it a suitable substitute for a trainer, coach or referee to make the goal of a healthier lifestyle more within reach.