CSCE 636: Deep Learning.

Project Report – Submission 1

Submitted by: Mamoon Masud

UIN: 828002068

Date: 03/23/2020

Topic

This project is being done as part of the course CSCE 636: Deep Learning. For this project, I was assigned to classify videos of people ***walking through a door*** using deep learning techniques. For this purpose, I'll be building a Deep Neural Network, and train it over a dataset, so that it's able to classify a video of a person walking through a door from another in which the person is not. Once trained, the code will take up a video and identify the parts of the video (time stamped) in which a person is walking through a door, if any.

Dataset

Deep Neural Networks require large number of training data in order to train and produce generalized results. Since the topic was a very specific one, it was tough to find out videos of people walking through door, however once dataset proved useful. ***Charades-Ego***[1] is a dataset that has been collected by Allen Institute of AI. This dataset, unlike most other datasets, is not collected from YouTube or Hollywood videos. The process of video creation was crowdsourced. Crowd sourcing the process of video creation has some advantages and disadvantages. In particular, the biggest advantage is that since it's not collected in a lab, the scenes in the videos are more natural, having less bias, which is evident in datasets collected from professionally shot videos, including lightning conditions, and the scenes. One disadvantage is that the people acting out in the videos are not the best actors, and at times the actions seem artificial, and less natural. Also, the video quality with blur and unstable camera being another issue. Without an alternative, I decided to go ahead and train with this dataset.

Data Filtering

The dataset has 157 classes, and a total of 10,000 videos. In total, the videos were shot in 267 homes. The first step was downloading the data, and since each video had multiple actions, (6.8 on average), the entire dataset was downloaded (58 GB). The next task was to filter out the videos which had the action "Walking through a doorway" being performed and crop the clips from those videos. As part of the dataset, a csv file is provided, which gives the annotations for each video file, (actions in the video, and the time at which they are performed). Therefore, the first task was to filter the videos in which the action was performed. The code for filtering the dataset is in the file *"Data_filter.ipynb"*.

After reading the CSV file using pandas, the first step was to filter out videos of quality less than 5 (on a scale of 1 to 7), and drop the videos which were not verified.

In the next step, the file was iterated upon, and the actions and their start and end times were extracted. This gave us the start and end times of all the actions for all the videos. This file

Following this, the next step was to generate the code that will crop the videos using the start and end times we extracted in the previous step and then save them. This is done for both the training set and the test set. This set was painstakingly time taking and difficult, since the book didn't provide any guidance on this, and limited help was available online. I've included more details on this in the problems encountered section.

Once we have extracted clips of Walking through the door, and of all other classes, manual filtering had to be performed. The reason being the fact that the videos in the dataset are of two types, first person and third person. As per my discussion with Dr. Jiang about the application of this project, I have, initially, only taken the videos with third person views. If Dr. Jiang deems it feasible, the possibility of using the first-person videos for training can be explored in the future models. In total, almost 1300 video were manually separated, so that we may get the appropriate third-person view video files. Once separated, the video files were then put in two different folders, named *door* and *not_door*, representative of their respective classes.

The following is the class distribution of the training data.

Class	**Training Videos**
Door	115
Not_door	121

Although not ideal, this is sufficient to train a model for the first iteration. Further submissions can be made with an increased no. of videos. Some of the proposed methods are discussed in the future plans section.

A last step in the data pre-processing was the resizing of the videos. The reason for this was that the videos in the dataset are shot in multiple dimensions. Some are in landscape, while some are in portrait. The aspect ratio also varies throughout the dataset. It is therefore imperative that the videos be made of the same height and width before they can be fed to the Neural Network. Initially, I used cv2's resize function to resize each individual frame of the video. The problems with it are discussed in the latter section. The file "Video_resize.ipynb" contains the code for resizing the videos to the required dimension using moviepy library. For the purpose of this submission, I used videos resized to a size of 300 * 300

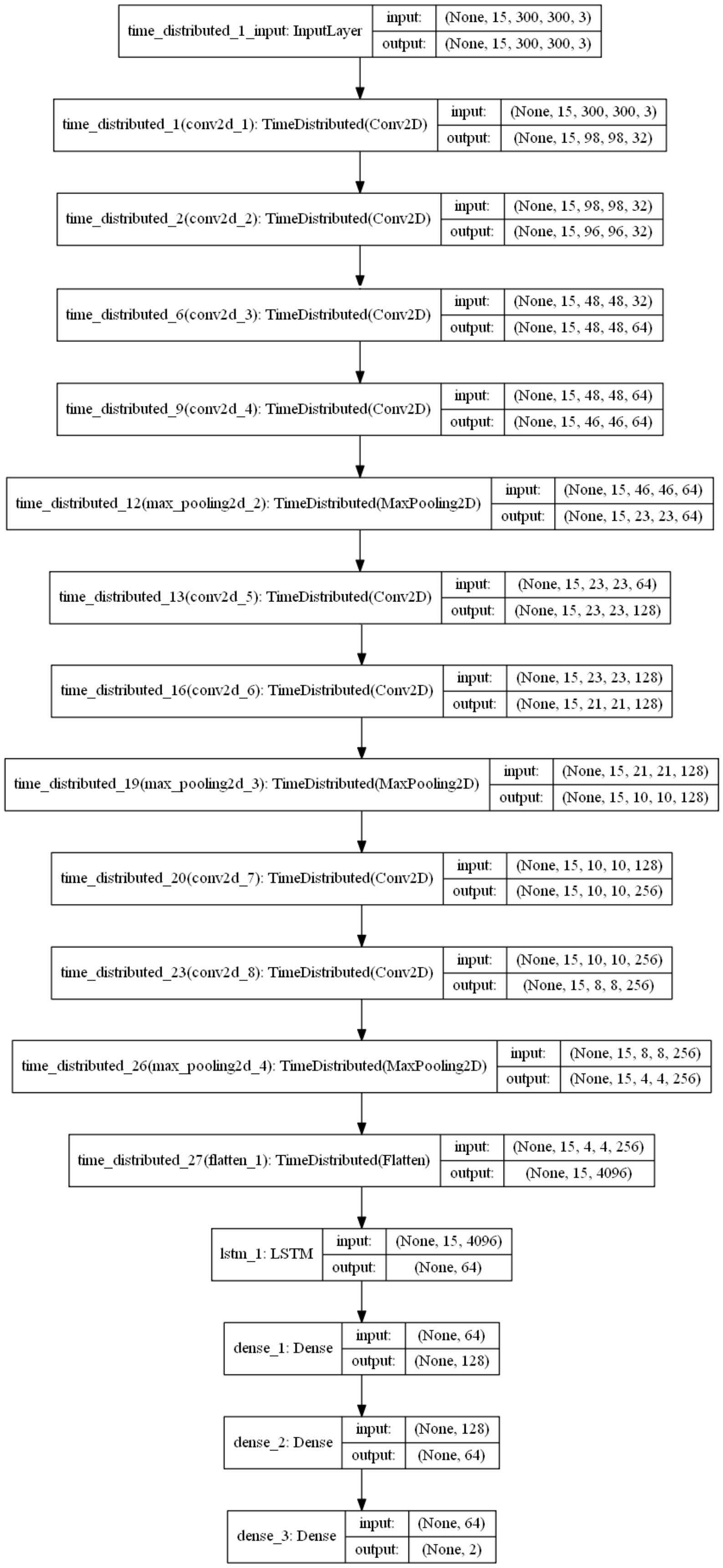
DNN Model

The model that gave better results is discussed here. I did test with other architectures, and they are explained in the latter sections.

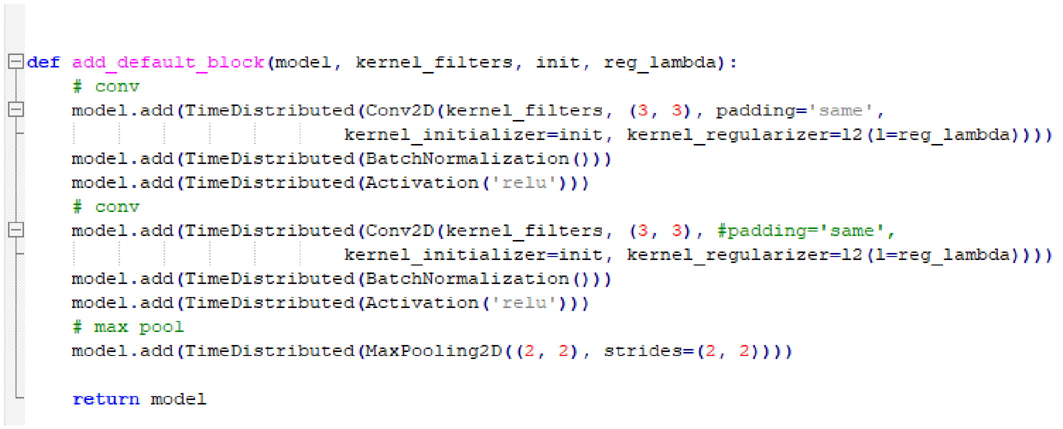
The model has two major parts. The first part performs feature extraction, which the second part performs the prediction of the class, based on the extracted features.

Architecture

The first part of the model has Convolutional Neural Network Layers, using ""Time Distributed Layers" in Keras. The Time Distributed layers help to present a sequence of images to the model and can therefore enable us to classify actions. The complete model architecture is given below. It is explained in detail are that.

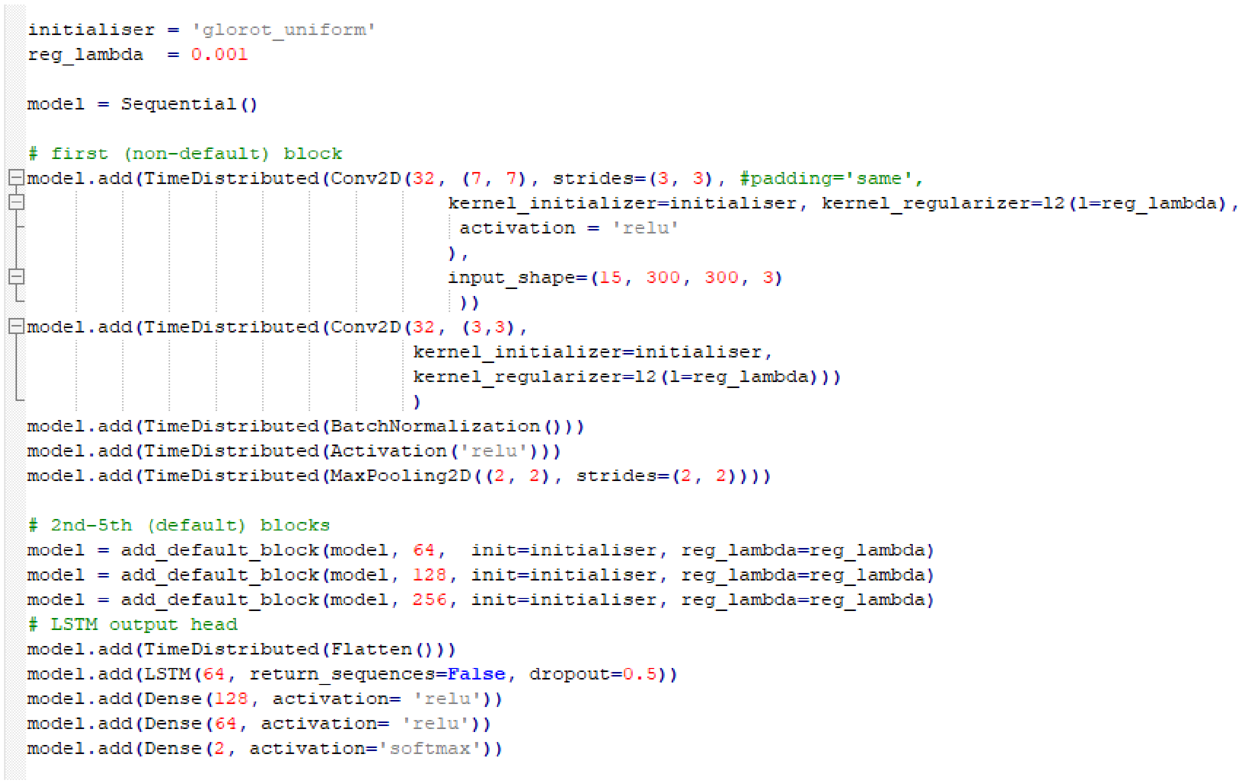

time_distributed_1_input: InputLayer | input: (None, 15, 300, 300, 3) | output: (None, 15, 300, 300, 3)
time_distributed_1(conv2d_1): TimeDistributed(Conv2D) | input: (None, 15, 300, 300, 3) | output: (None, 15, 98, 98, 32)
time_distributed_2(conv2d_2): TimeDistributed(Conv2D) | input: (None, 15, 98, 98, 32) | output: (None, 15, 96, 96, 32)
time_distributed_6(conv2d_3): TimeDistributed(Conv2D) | input: (None, 15, 48, 48, 32) | output: (None, 15, 48, 48, 64)
time_distributed_9(conv2d_4): TimeDistributed(Conv2D) | input: (None, 15, 48, 48, 64) | output: (None, 15, 46, 46, 64)
time_distributed_12(max_pooling2d_2): TimeDistributed(MaxPooling2D) | input: (None, 15, 46, 46, 64) | output: (None, 15, 23, 23, 64)
time_distributed_13(conv2d_5): TimeDistributed(Conv2D) | input: (None, 15, 23, 23, 64) | output: (None, 15, 23, 23, 128)
time_distributed_16(conv2d_6): TimeDistributed(Conv2D) | input: (None, 15, 23, 23, 128) | output: (None, 15, 21, 21, 128)
time_distributed_19(max_pooling2d_3): TimeDistributed(MaxPooling2D) | input: (None, 15, 21, 21, 128) | output: (None, 15, 10, 10, 128)
time_distributed_20(conv2d_7): TimeDistributed(Conv2D) | input: (None, 15, 10, 10, 128) | output: (None, 15, 10, 10, 256)
time_distributed_23(conv2d_8): TimeDistributed(Conv2D) | input: (None, 15, 10, 10, 256) | output: (None, 15, 8, 8, 256)
time_distributed_26(max_pooling2d_4): TimeDistributed(MaxPooling2D) | input: (None, 15, 8, 8, 256) | output: (None, 15, 4, 4, 256)
time_distributed_27(flatten_1): TimeDistributed(Flatten) | input: (None, 15, 4, 4, 256) | output: (None, 15, 4096)
lstm_1: LSTM | input: (None, 15, 4096) | output: (None, 64)
dense_1: Dense | input: (None, 64) | output: (None, 128)
dense_2: Dense | input: (None, 128) | output: (None, 64)
dense_3: Dense | input: (None, 64) | output: (None, 2)

In total, there are 8 CNN layers, 1 LSTM layer, and 2 Dense layers. The code for defining the model is given below:



```
def add_default_block(model, kernel_filters, init, reg_lambda):
    # conv
    model.add(TimeDistributed(Conv2D(kernel_filters, (3, 3), padding='same',
                                     kernel_initializer=init, kernel_regularizer=l2(l=reg_lambda))))
    model.add(TimeDistributed(BatchNormalization()))
    model.add(TimeDistributed(Activation('relu')))
    # conv
    model.add(TimeDistributed(Conv2D(kernel_filters, (3, 3), #padding='same',
                                     kernel_initializer=init, kernel_regularizer=l2(l=reg_lambda))))
    model.add(TimeDistributed(BatchNormalization()))
    model.add(TimeDistributed(Activation('relu')))
    # max pool
    model.add(TimeDistributed(MaxPooling2D((2, 2), strides=(2, 2))))

    return model
```



```
initialiser = 'glorot_uniform'
reg_lambda  = 0.001

model = Sequential()

# first (non-default) block
model.add(TimeDistributed(Conv2D(32, (7, 7), strides=(3, 3), #padding='same',
                                    kernel_initializer=initialiser, kernel_regularizer=l2(l=reg_lambda),
                                     activation = 'relu'
                                    ),
                                    input_shape=(15, 300, 300, 3)
                                     ))
model.add(TimeDistributed(Conv2D(32, (3,3),
                                 kernel_initializer=initialiser,
                                 kernel_regularizer=l2(l=reg_lambda)))
                                 )
model.add(TimeDistributed(BatchNormalization()))
model.add(TimeDistributed(Activation('relu')))
model.add(TimeDistributed(MaxPooling2D((2, 2), strides=(2, 2))))

# 2nd-5th (default) blocks
model = add_default_block(model, 64,  init=initialiser, reg_lambda=reg_lambda)
model = add_default_block(model, 128, init=initialiser, reg_lambda=reg_lambda)
model = add_default_block(model, 256, init=initialiser, reg_lambda=reg_lambda)
# LSTM output head
model.add(TimeDistributed(Flatten()))
model.add(LSTM(64, return_sequences=False, dropout=0.5))
model.add(Dense(128, activation= 'relu'))
model.add(Dense(64, activation= 'relu'))
model.add(Dense(2, activation='softmax'))
```

The last layer has 2 outputs, and each output provides the probability of the input being of the class door and not_door. Dropout has been added in order to reduce over-fitting, since we don’t have a large dataset, and it’s highly likely that the model will overfit.

Video Frame Generator:

Initially, I wrote my own code to generate frames from each video, and then concatenate them into a tensor, before feeding into the network. Later, however, I was able to find a custom Video Frame Generator on GitHub [2 , 3], that essentially works in the same way as Image generator does, but for Videos. The Frame generator requires the path to the videos, total classes, frames per video, size of the frames (Width & Height), batch size, channels (3 or 1) as well as ranges of different shifts in the frames for data augmentation. This again is useful since our dataset is small, and data augmentation will help in reducing over fitting.

For our model, we use frame size of 300 * 300, 3 channels, 15 frames per video, and a Batch size of 5. For the batch size, we were restricted by memory, since the model was being trained on GPU laptop.

Input: Shape of Tensor

train = *VideoFrameGenerator* generates tensors of size (15, 300, 300, 3)

(where 15 is the no. of frames, and is equal to the no. of time-distributed CNN layers. 3 is the no. of channels, and 300*300 is the size of each frame.)

In total, 178 videos are used for training, and 58 videos are used for validation.

Output: Shape of Tensor

Output is of shape (178 , 2)

Each output gives the probability of the input belonging to each class.

Shape of Output Tensor for Each Layer

Time-Distributed CNN Layer 1:	(178, 15, 98, 98, 32)
Time-Distributed CNN Layer 2:	(178, 15, 96, 96, 32)
Max-Pooling Layer 1:	(178, 15, 48, 48, 32)
Time-Distributed CNN Layer 3:	(178, 15, 48, 48, 64)
Time-Distributed CNN Layer 4:	(178, 15, 46, 46, 64)
Max-Pooling Layer 2:	(178, 15, 23, 23, 64)
Time-Distributed CNN Layer 5:	(178, 15, 23, 23, 128)
Time-Distributed CNN Layer 6:	(178, 15, 21, 21, 128)
Max-Pooling Layer 3:	(178, 15, 10, 10, 128)
Time-Distributed CNN Layer 7:	(178, 15, 10, 10, 256)
Time-Distributed CNN Layer 8:	(178, 15, 8, 8, 256)

Max-Pooling Layer 4:	(178, 15, 4, 4, 256)
Flatten Layer:	(178, 4096)
LSTM Layer:	(178, 64)
Dense Layer 1:	(178, 128)
Dense Layer 2:	(178, 64)
Dense Layer 3:	(178, 2)

Hyperparameters:

List of Hyperparameters:

For this project, there are in total 6 hyperparameters.

Hyper-parameter	Range
No. of frames	5- 15
Batch size	2-5
Epochs	10-50
Learning rate	1e-4, 1e-5
Optimizer	Adam & RMSProp

Optimal Hyperparameters found:

Hyper-parameter	Range
No. of frames	15
Batch size	5
Epochs	42Th
Learning rate	1e-4
Optimizer	RMSProp

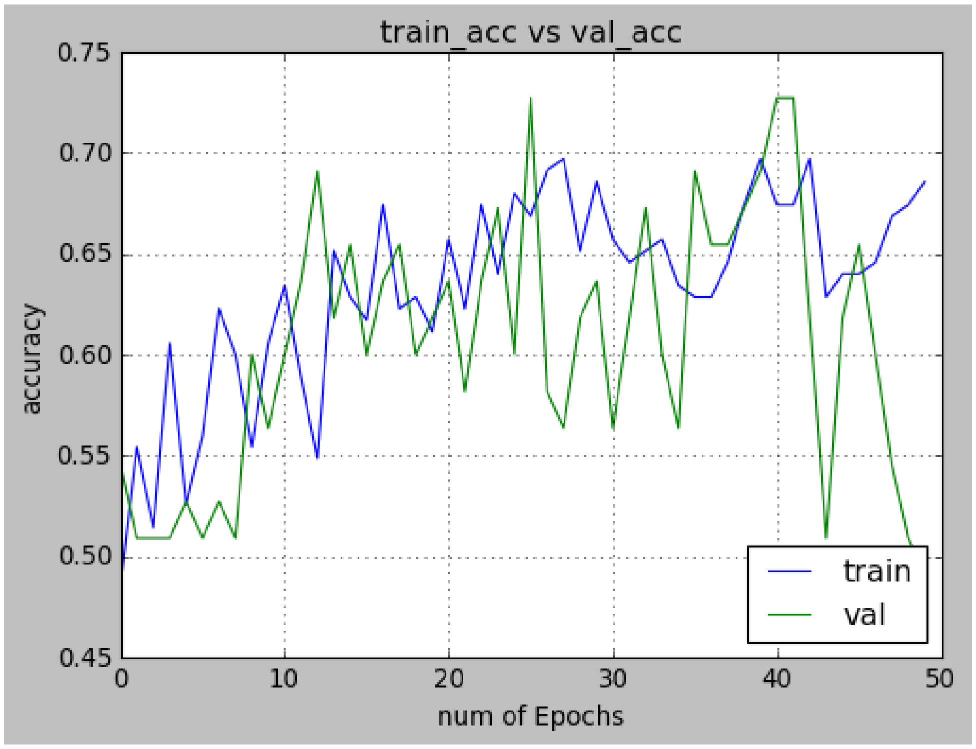
Annotated Code:

The code is uploaded in .ipynb files on GitHub.

Training Performance

The training performance is given in the *logs-training.log* file uploaded on GitHub. It is pertinent to mention here that the data used for training the two classes was quite similar, and the characters & background for videos in the two classes are common for a large number of videos. The accuracy can be improved if we select not_door videos that are quite different from the door videos (for instance outdoor videos for not_door class), but that would be kind of cheating the entire process, as the model would learn to detect the background and surroundings, rather than the actual action, and still give a better performance than the current model.

The model starts to overfit significantly after 42nd epoch.



Testing Performance

The test set contains 30 videos, 15 of each class. The performance of the model on the test set is given below:

Loss = 1.0280625820159912,
Accuracy = 0.6666666865348816

Which shows that the model's performance is similar to the performance on training and validation set. The Results can be verified by running the file test_set.ipynb

Instructions on testing the trained DNN

Open the notebook names *Prediction.ipynb.* Then, run the first four cells. In the fifth cell, it asks for the path to the video file, and the destination path for saving the txt file, which saves the predictions for the video. Once done, run the 5th code block, and it will generate the prediction.

Video Link

The video can be found at: https://youtu.be/H85dwC5R03c

Encountered Problems

As a beginner in the domain of Deep Learning, this project, though seems simple, proved challenging for a lot of reasons. The major challenge was to find appropriate videos/dataset for the given problem.

The first This however was not producing the desired result. It was adding unwanted artefacts in the frames. Some videos were padded with black pixels on each side, while some were padded with edge pixels, and due to this, the training done on these frames was not producing desired results.

Future Improvements

A potential improvement, which is not easy, would be to increase the training size. I have found two classes in another dataset, opening door, and closing door. If Dr. Jiang approves of it, I'll be able to increase my training data size by 3 times, which would improve the model's performance.

Apart from this, a technique that I haven't yet tested is using a model with pre-trained weights, such as MobileNet, and then adding LSTM and Dense layers for predicting class probabilities.

References

[1] Sigurdsson, Gunnar & Varol, Gül & Wang, Xiaolong & Farhadi, Ali & Laptev, Ivan & Gupta, Abhinav. (2016). Hollywood in Homes: Crowdsourcing Data Collection for Activity Understanding. 9905. 10.1007/978-3-319-46448-0_31.

[2] https://pypi.org/project/keras-video-generators/

[3] https://github.com/metal3d/keras-video-generators