MSDS 458 Fall 2021 Assignment 4

Husein Adenwala

1. Abstract

Action recognition on large categories of unconstrained videos is very challenging because of camera motion, different viewpoints, large interclass variations, cluttered backgrounds, and occlusions. The purpose of this research was to review various deep learning architectures for human action recognition in videos. I worked with the UCF50 dataset that is provided by UCF center of research for computer vision research. It consists of realistic videos taken from YouTube that fall into 50 action categories. I experimented with various frame sizes and frame sequence lengths and hyperparameters for a Simple DNN and LSTM, Long-term Recurrent Convolutional Network (LRCN) model and hybrid CNN-LSTM model that has convolution and LSTM layers. Results from my experiment suggest that CNN and LRCN models performed well and better than DNN and LSTM models; however, the hybrid CNN-LSTM model had the best test accuracy and was one of the models to take the least training time.

2. Introduction

Humans easily recognize and identify actions in video but automating this procedure is challenging. However, video-based human action recognition has become one of the most popular research areas in the field of computer vision and pattern recognition and with the explosion of video recording devices in the market, the interest for automating activity detection in humans has never been higher. One of the possible applications is to provide audio output for visually impaired people where a camera could provide a description of the content to be read to the person. A consulting firm has hired me to develop an

exploratory deep learning model architecture that is able to recognize human action and that can subsequently provide audio output that can state the human activity.

3. Literature review

Deep learning architectures are highly successful and widely implemented in image classification but progress in architectures for video classification and representation learning has been relatively slower. However, video classification has made significant progress in recent years. Most of the existing deep learning architectures for video classification are basically adopted from the favored deep learning architectures in the image/speech domain (Rehman et al, 2021).

CNNs and RNN based architectures are most common for video classification as a video is just a stack of images in a sequence (Ng et al, 2-14).

There are architectures that combine CNN RNN architecture such as Long-term Recurrent Convolutional Network Models (LRCN) . LRCN is a variant of LSTM (Long Short-Term Memory) containing a convolution operation inside the LSTM cell. Both of these models are a special kind of RNN, capable of learning long-term dependencies.

ConvLSTM cell in LRCN replaces matrix multiplication with convolution operation at each gate in the LSTM cell. By doing so, it captures underlying spatial features by convolution operations in multi-dimensional data (Shi et al, 2015). Similarly, combining CNN and LSTM networks can capture both spatial and long-term dependences in a video.

4. Methods

Research design and modeling methods

For this research, simple neural networks, Long Short-Term Memory (LSTM) and Convolutional Neural, CONLSTM model and hybrid model of CNN-LSTM are analyzed and compared with varying architectures for each.

A video consists of an ordered sequence of frames. Each frame contains spatial information, and the sequence of those frames contains temporal information. To model both of these aspects, I have used architecture that consists of convolutions (for spatial processing) as well as recurrent layers (for temporal processing) and I have also used hybrid architecture that combines convolution and recurrent layers for video classification (ConvLSTM and CNN-LSTM models).

In addition to different architectures, I have experimented with image size, image sequence length and regularization techniques.

All models use the ReLU activation function in the dense and convolution layers, which is one of the popular activation functions in neural networks because it is computationally efficient and fixes the problem of vanishing gradients. I have used Tanh in ConvLSTM network as tanh activation is the go-to activation function in RNN architectures. I used validation accuracy as the metric to evaluate the performance. Finally, I used categorical cross entropy to measure the loss between labels and predictions and SoftMax activation function for the output layer.

Data preparation, exploration, visualization

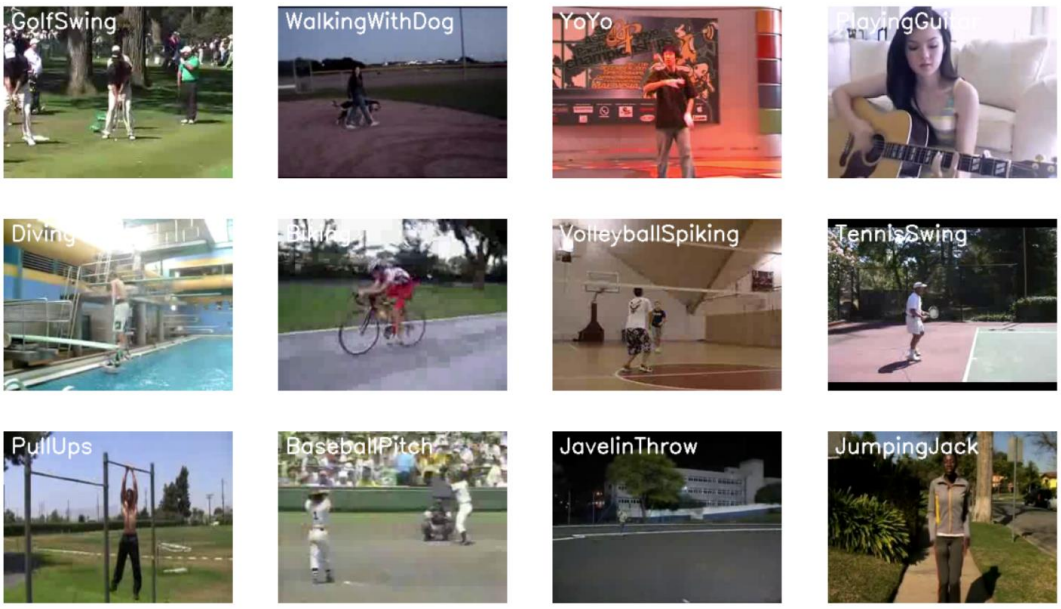
The dataset used in this research is UCF50, which consists of realistic videos taken from YouTube. UCF50 has a total of 6081 videos that fall into 50 action categories. Each action class contains a minimum of 100 videos.

The Dataset summary:

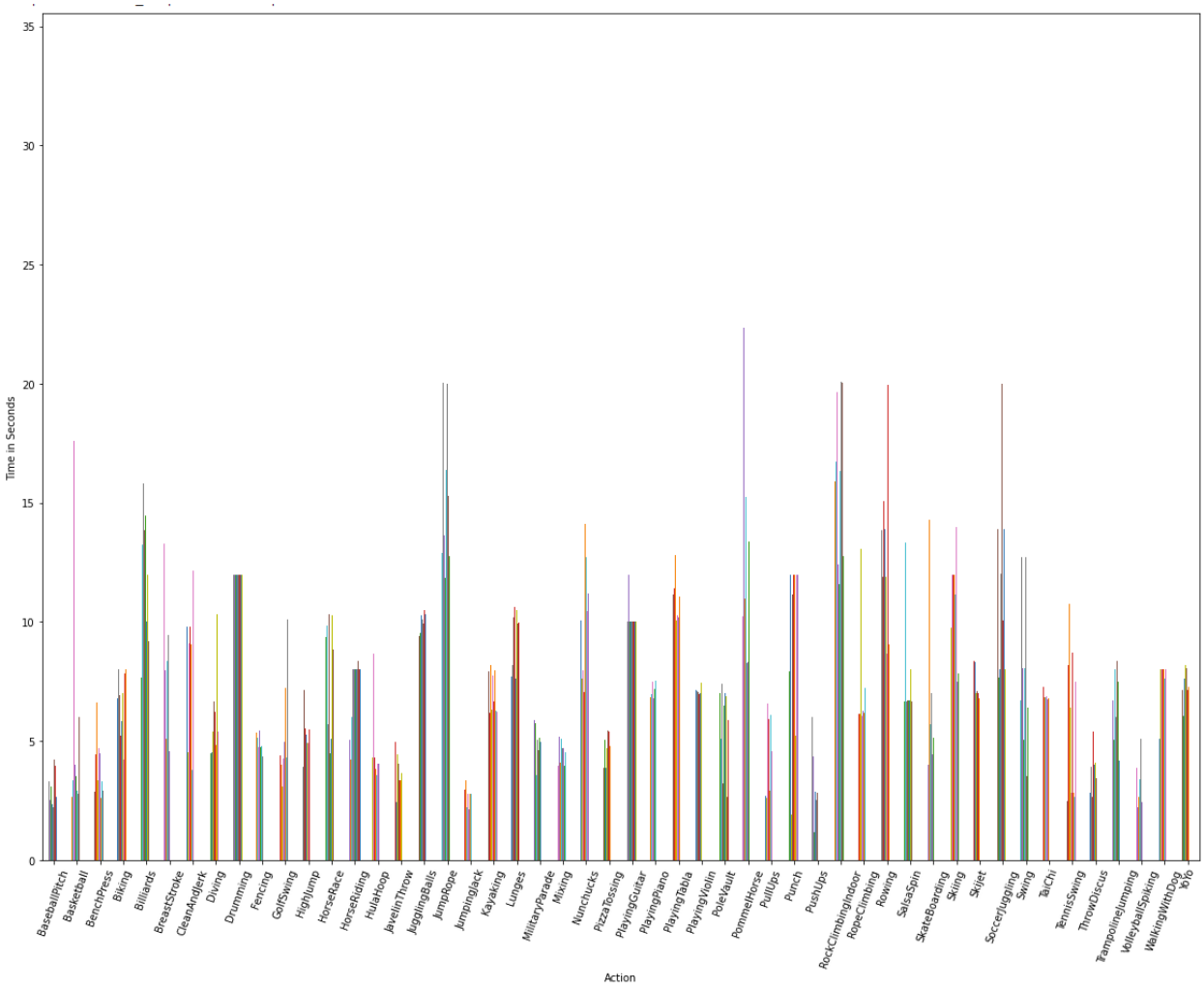
- **50** Action Categories
- **133** Average Videos per Action Category
- **199** Average Number of Frames per Video
- **320** Average Frames Width per Video
- **240** Average Frames Height per Video
- **26** Average Frames Per Seconds per Video

(See Appendix A for dataset summary overview)

Examples:



The bar graph below provides an overview of the length of all videos in the UCF50 dataset.



50 Action Classes:

```
['BaseballPitch', 'Basketball', 'BenchPress', 'Biking', 'Billiards',
 'BreastStroke', 'CleanAndJerk', 'Diving', 'Drumming', 'Fencing', 'GolfSwing',
 'HighJump', 'HorseRace', 'HorseRiding', 'HulaHoop', 'JavelinThrow',
 'JugglingBalls', 'JumpRope', 'JumpingJack', 'Kayaking', 'Lunges',
 'MilitaryParade', 'Mixing', 'Nunchucks', 'PizzaTossing', 'PlayingGuitar',
 'PlayingPiano', 'PlayingTabla', 'PlayingViolin', 'PoleVault', 'PommelHorse',
 'PullUps', 'Punch', 'PushUps', 'RockClimbingIndoor', 'RopeClimbing', 'Rowing',
 'SalsaSpin', 'SkateBoarding', 'Skiing', 'Skijet', 'SoccerJuggling', 'Swing',
 'TaiChi', 'TennisSwing', 'ThrowDiscus', 'TrampolineJumping',
 'VolleyballSpiking', 'WalkingWithDog', 'YoYo']
```

Preprocessing data:

On average, each video contains 199 frames of 320 x 240-pixel size i.e. converting 3 RGB channels into a numpy array would be 320 x 240 x 3 that would be multiplied by 199 frames. Multiplied by 50 categories, this would create a computationally inefficient array; therefore, I have resized the image shape taken the sequence length.

Given the computation limitations, I experimented by resizing the image by 64 X 64 or 32 x 32 and chose between 20 or 10 frames evenly per video for training the model. The data was then normalized by dividing the NumPy array by 255.

The data was split between Train (75%) and Test Set (25%) and 20% of training data was used for validation while training each model.

Implementation and programming

Model research was done using Python programming language and TensorFlow. TensorFlow is a free and open-source software library for machine learning and artificial intelligence and TensorFlow can be implemented in python by Keras. Keras is a deep learning API written in Python, running on top of the machine learning platform TensorFlow which is extremely user-friendly.

The dataset was downloaded In google Colab directly from UCF Center for Research in Computer Vision, using the Linux !wget command

I used the sci-kit learn library to create the Train (75%) and Test Set (25%) and to get a confusion matrix and report on precision, accuracy and F1 score.

I used OpenCV library to extract frames from video, and python multithreading to increase performance.

Additionally, I used NumPy and Pandas library for data for EDA and finally, I also used seaborn package and matplotlib library to visualize the data and results.

5. Results

Experiments:

11 models were evaluated by conducting experiments. The experiments were done by varying model architecture, image size, sequence length and regulation methods.

Results

Name	Type	Architecture	Frame Size	Number of Frames	Regularization	Batch size	Optimzer	Train Accuracy	Valid Accuracy	Test Accuracy	Time
Model 5C	CNN-LSTM	4 CON2d layer 8 + 16 + 32 + 32 Kernel MaxPool2D, Nested in TimeDistrubuted layer + 1 LSTM layer 32 Cells	64 x 64	20	Early stopping + Dropout		ADAM	89.75	73.22	70.92	9 min
Model 5B	CNN-LSTM	4 CON2d layer 8 + 16 + 32 + 32 Kernel MaxPool2D, Nested in TimeDistrubuted layer + 1 LSTM layer 32 Cells	64 x64	10	Early stopping + Dropout		ADAM	78	64	69	6 min
Model 3B	CNN	4 CON2d layer 16 + 32 + 64 + 64 Kernel, MaxPool2D, 1 Dense layer 2056 Nodes	64 x 64	20	Early stopping + Dropout + Batch Normalization	4	ADAM	84.79	67.27	67	25 min
Model 4B	LRCN	4 ConvLSTM Layer 4 + 8 + 14 +15 Kernel MaxPool3D	64 x64	10	Early stopping + Dropout	4	ADAM	95	67	61	146 min
Model 4C	LRCN	4 ConvLSTM Layer 4 + 8 + 14 +15 Kernel MaxPool3D	64 x 64	20	Early stopping + Dropout		ADAM	94	65	60.2	95 min
Model 3A	CNN	4 CON2d layer 16 + 32 + 64 + 64 Kernel, MaxPool2D Nested in TimeDistrubuted layer + 1 Dense layer 2056 Nodes	64 x64	20	Early stopping + Dropout	4	ADAM	72.83	60.23	57.72	20 min
Model 5A	CNN-LSTM	4 CON2d layer 8 + 16 + 32 + 32 Kernel MaxPool2D, Nested in TimeDistrubuted layer + 1 LSTM layer 32 Cells	32 x 32	20	Early stopping + Dropout		ADAM	74.58	57	55	26 min
Model 4A	LRCN	4 ConvLSTM Layer 4 + 8 + 14 +15 Kernel MaxPool3D	32 x 32	20	Early stopping + Dropout	4	ADAM	80.74	58	54	104 min
Model 1B	DNN	4 Dense layer 1024 +512 + 256 +128 Nodes	64 x 64	20	Early stopping	4	ADAM	65	40.9	39	23 min
Model 1A	DNN	2 Dense layer 512 + 512 Nodes	64 x 64	20	Early stopping	128	RMSprop	4.5	4.4	4.2	6 min
Model 2A	LSTM	1 LSTM Layer Bidirectional 64 Cells + 1 Dense Layer 1024 Nodes	64 x 64	20	Early stopping	128	RMSprop	3.9	3.7	3.4	3 min

Key Findings:

Image size and number of frames:

The larger 64 X 64 image size provided better results compared to 32 x 32 image size. On the other hand, increasing the number of frames from 10 to 20 frames per video did not significantly improve the model result. Larger frame size would give a moderate increase in model performance; however, due to computation limitations (e.g. running out of memory), I could not test with the larger image size.

Regularization using early Stopping, dropout and batch normalization:

Overfitting was observed with models that had multiple CNN, LRCN (ConvLSTM) and CNN+LSTM layers.

I applied early stopping and dropout to LRCN model. For CNN layers, I applied batch normalization in addition to early stopping and dropout.

DNN Models:

I experimented with the number of dense layers and batch size in models. The denser layers and lower batch size improved the model. However, the best DNN model test accuracy was 39%.

LSTM Models:

1-layer bidirectional LSTM model with 64 nodes performed poorly and it could possibly be due to the large batch size; however, reducing the batch size made the LSTM model computational very expensive and unviable to run in google Colab.

CNN Models:

I experimented with 4 Conv2d layers models with and without batch normalization and found that batch normalization provides significant improvement in performance. The best CNN model had a test accuracy of 67% (See Appendix C).

Long-term Recurrent Convolutional Network Models (LRCN):

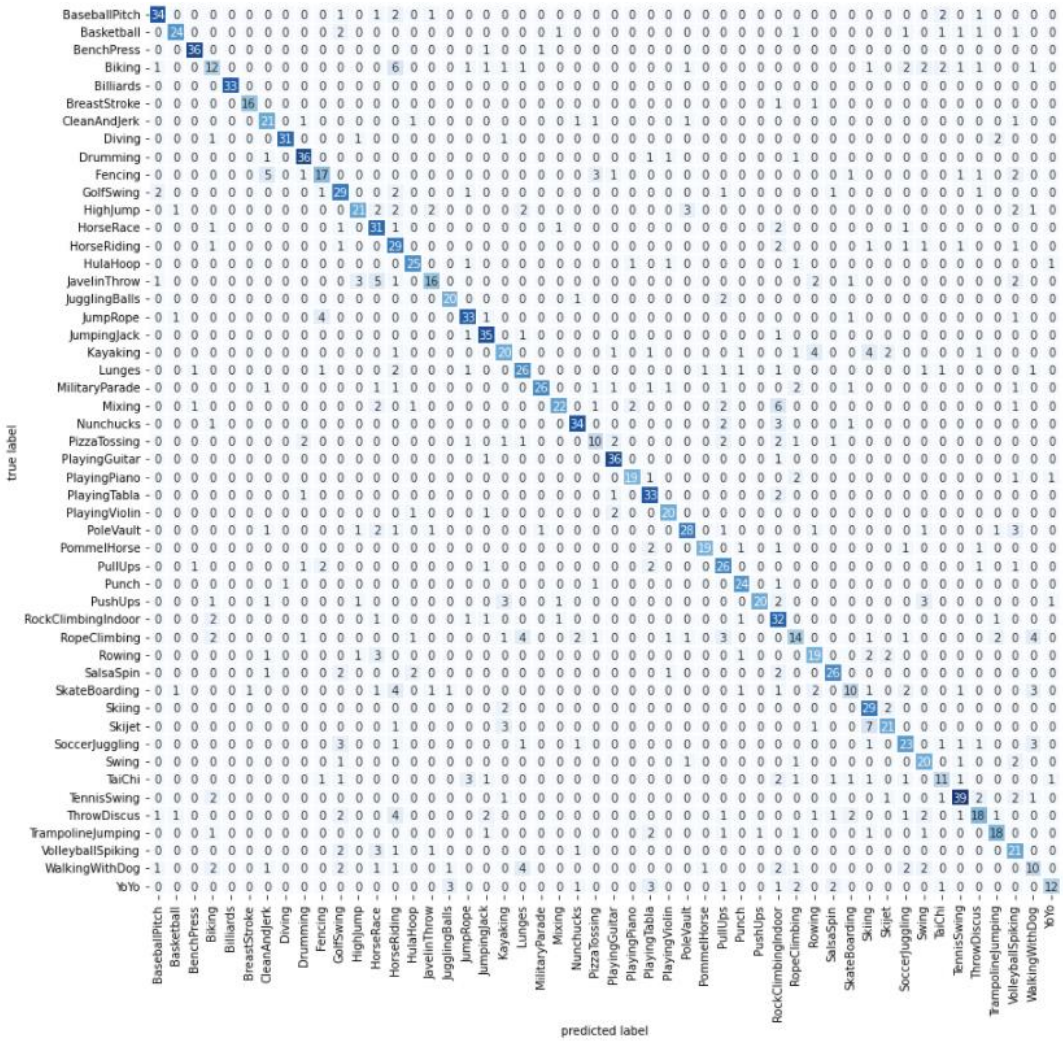
The best LRCN model had a test accuracy of 61% and training accuracy of 95%, which shows overfitting. The LRCN model has potential to improve; however, it had the longest training time and the RNN architecture has limited regularization techniques (See Appendix B).

Hybrid CNN-LSTM Model:

CNN-LSTM hybrid model had the best test accuracy and required significantly less training time compared to the LRCN model. The best CNN-LSTM model had a test accuracy of 70.92% and training accuracy of 89.75%. Given that this model had the highest test accuracy, it is the the model I would ultimately recommend my clients to pursue.

Best, Model 5C: CNN-LSTM model trained using image size 64 x 64 and sequence length of 20 frame per video.

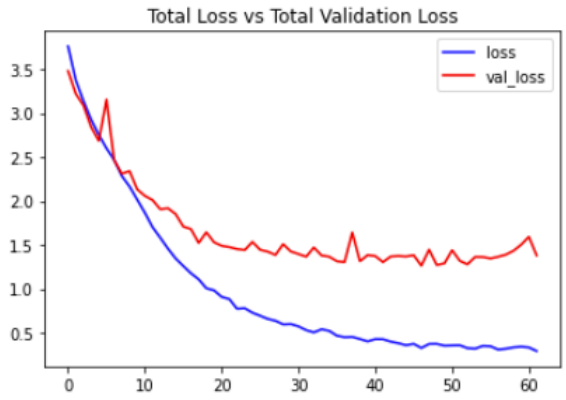
Name	Type	Architecture	Frame Size	Number of Frames	Regularization	Batch size	Optimizer	Train Accuracy	Valid Accuracy	Test Accuracy	Time
Model 5C	CNN-LSTM	4 CON2d layer 8 + 16 + 32 + 32 Kernel MaxPool2D, Nested in TimeDistrubuted layer + 1 LSTM layer 32 Cells	64 x 64	20	Early stopping + Dropout		ADAM	89.75	73.22	70.92	9 min

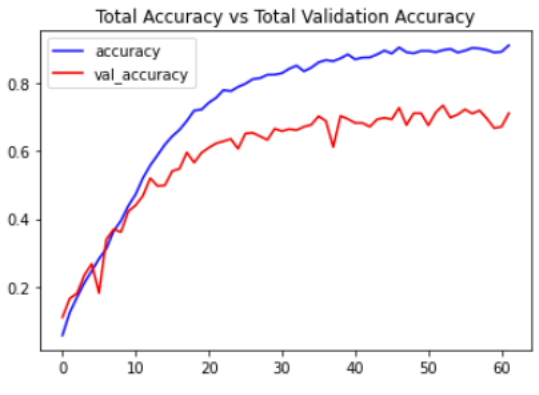


BaseballPitch	0.85	0.81	0.83	42
Basketball	0.86	0.73	0.79	33
BenchPress	0.92	0.95	0.94	38
Biking	0.46	0.35	0.40	34
Billiards	1.00	1.00	1.00	33
BreastStroke	0.94	0.89	0.91	18
CleanAndJerk	0.64	0.78	0.70	27
Diving	0.97	0.86	0.91	36
Drumming	0.84	0.90	0.87	40
Fencing	0.65	0.53	0.59	32
GolfSwing	0.62	0.76	0.68	38
HighJump	0.75	0.58	0.66	36
HorseRace	0.58	0.82	0.68	38
HorseRiding	0.48	0.76	0.59	38
HulaHoop	0.81	0.83	0.82	30
JavelinThrow	0.73	0.52	0.60	31
JugglingBalls	0.80	0.87	0.83	23
JumpRope	0.77	0.80	0.79	41
JumpingJack	0.76	0.92	0.83	38
Kayaking	0.61	0.56	0.58	36
Lunges	0.65	0.68	0.67	38
MilitaryParade	0.93	0.68	0.79	38
Mixing	0.85	0.58	0.69	38
Nunchucks	0.83	0.83	0.83	41
PizzaTossing	0.56	0.43	0.49	23
PlayingGuitar	0.82	0.95	0.88	38
PlayingPiano	0.86	0.79	0.83	24
PlayingTabla	0.72	0.89	0.80	37
PlayingViolin	0.80	0.83	0.82	24
PoleVault	0.80	0.67	0.73	42
PommelHorse	0.90	0.76	0.83	25
PullUps	0.59	0.74	0.66	35
Punch	0.80	0.89	0.84	27
PushUps	0.95	0.61	0.74	33
RockClimbingIndoor	0.49	0.80	0.61	40
RopeClimbing	0.48	0.36	0.41	39
Rowing	0.61	0.66	0.63	29
SalsaSpin	0.81	0.76	0.79	34
SkateBoarding	0.56	0.33	0.42	30
Skiing	0.59	0.88	0.71	33
Skijet	0.75	0.64	0.69	33
SoccerJuggling	0.64	0.64	0.64	36
Swing	0.61	0.77	0.68	26
TaiChi	0.55	0.42	0.48	26
TennisSwing	0.81	0.80	0.80	49
ThrowDiscus	0.62	0.47	0.54	38
TrampolineJumping	0.72	0.67	0.69	27
VolleyballSpiking	0.50	0.72	0.59	29
WalkingWithDog	0.42	0.32	0.36	31
YoYo	0.75	0.46	0.57	26
accuracy			0.71	1671
macro avg	0.72	0.71	0.70	1671
weighted avg	0.72	0.71	0.71	1671

```
Accuracy Score: 0.7091561938958707
Root Mean Square Error: 10.830421281494543
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 time_distributed_12 (TimeDi  (None, 20, 64, 64, 16)   448
 stributed)

 time_distributed_13 (TimeDi  (None, 20, 16, 16, 16)   0
 stributed)

 time_distributed_14 (TimeDi  (None, 20, 16, 16, 16)   0
 stributed)

 time_distributed_15 (TimeDi  (None, 20, 16, 16, 32)   4640
 stributed)

 time_distributed_16 (TimeDi  (None, 20, 4, 4, 32)     0
 stributed)

 time_distributed_17 (TimeDi  (None, 20, 4, 4, 32)     0
 stributed)

 time_distributed_18 (TimeDi  (None, 20, 4, 4, 64)     18496
 stributed)

 time_distributed_19 (TimeDi  (None, 20, 2, 2, 64)     0
 stributed)

 time_distributed_20 (TimeDi  (None, 20, 2, 2, 64)     0
 stributed)

 time_distributed_21 (TimeDi  (None, 20, 2, 2, 64)     36928
 stributed)

 time_distributed_22 (TimeDi  (None, 20, 1, 1, 64)     0
 stributed)

 time_distributed_23 (TimeDi  (None, 20, 64)           0
 stributed)

 lstm_1 (LSTM)               (None, 32)                12416

 dense_2 (Dense)             (None, 50)                1650

=================================================================
Total params: 74,578
Trainable params: 74,578
Non-trainable params: 0
_________________________________________________________________
Model Created Successfully!
```





6. Conclusions

The results of the 11 experiments show that the hybrid CNN-LSTM model performs better than CNN and LRCN models. CNN models performed surprisingly well and given the short video clip, it makes intuitive sense that spatial correlation would be more significant than sequential correlation. The experiments also showed that image size and sequence length (number of frames per video) have an impact and, in general, larger images and sequence length provided better results, though this had a diminishing rate of return and was computationally very expensive.

LRCN models performed best on training data (overfitting) and while there was potential to get better test results with hyperparameter tuning, although, the long training time was its biggest drawback. A hybrid CNN-LSTM model architecture was also capable of capturing the spatial and sequential information and was computationally efficient.

In conclusion, because of the highest test accuracy and computational efficiency, I would recommend the hybrid CNN-LSTM model 5C to build a video classifier model for human activity recognition.

Post analysis

Best Model: Incorrect vs correct classification examples:

GolfSwing

TrampolineJumping

GolfSwing

Swing

GolfSwing

JumpingJack

Punch

PoleVault

PlayingGuitar

JumpingJack

Skiing

RockClimbingIndoor

TennisSwing

PullUps

HighJump

ThrowDiscus

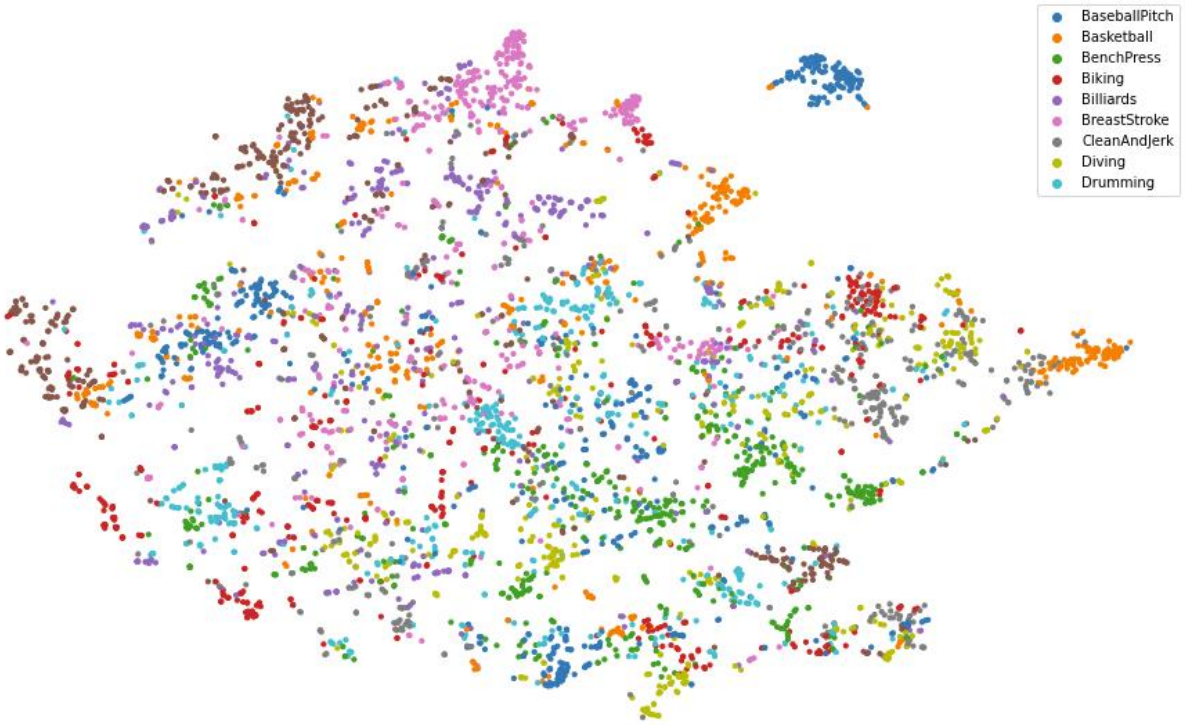
Skiing

RockClimbingIndoor

VolleyballSpiking

SoccerJuggling

T-SNE plot of LSTM layer in CNN-LSTM model.



References:

Rehman A, Belhaouari SB (2021) Deep Learning for Video Classification: A Review. https://doi.org/10.36227/techrxiv.15172920.v1.

Ng, J.Y., Hausknecht, M.J., Vijayanarasimhan, S., Vinyals, O., Monga, R., & Toderici, G. (2015). Beyond short snippets: Deep networks for video classification. 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 4694-4702.

Shi, X., Chen, Z., Wang, H., Yeung, D., Wong, W., & Woo, W. (2015). Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting. NIPS.

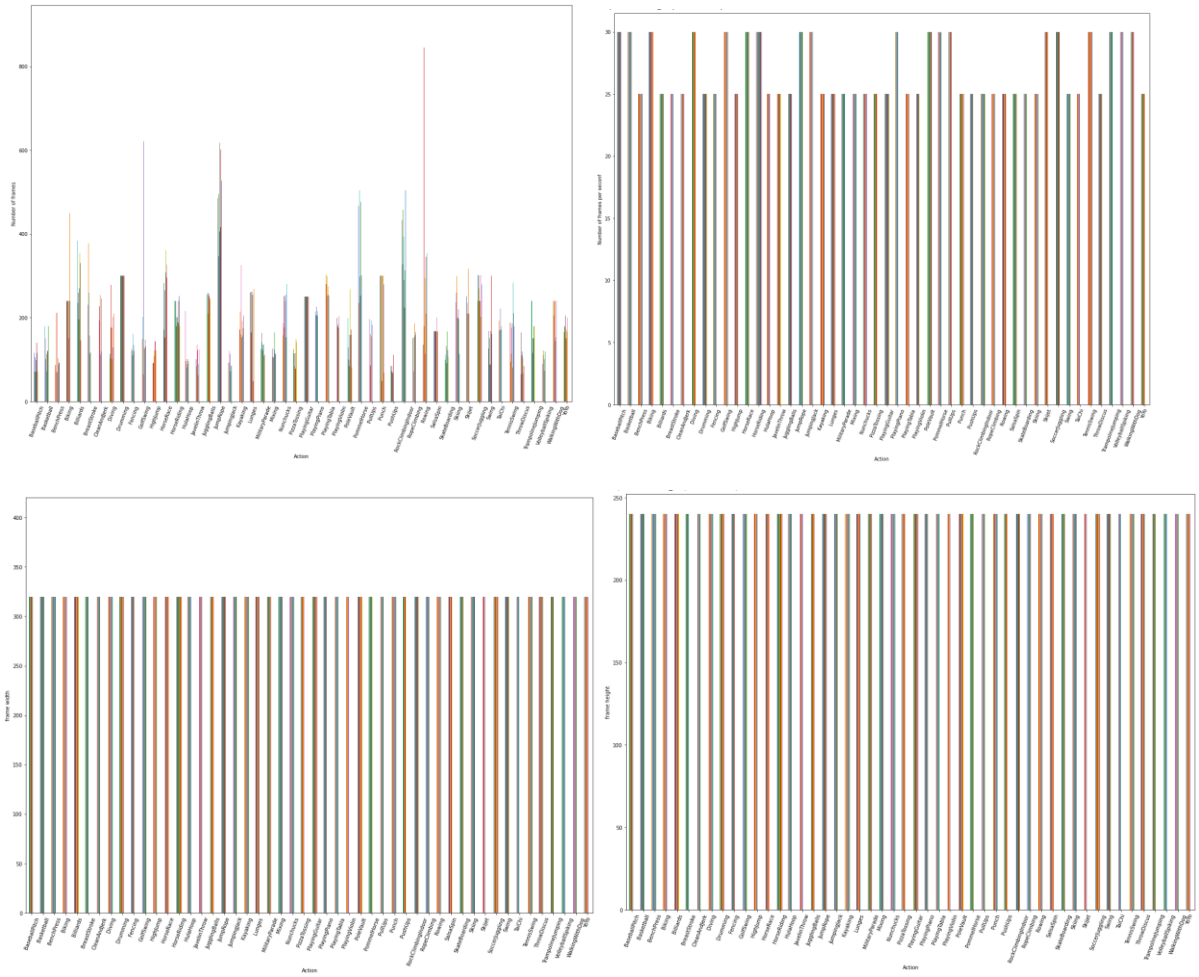
Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun: Deep Residual Learning for Image Recognition, Dec 2015, DOI: https://arxiv.org/abs/1512.03385

Frahcois, C. (2018). Deep Learning with Python. Shelter Island, New York: Manning Publications Co.

Geìron, A. (2019). *Hands-on machine learning with Scikit-Learn, Keras and TensorFlow: concepts, tools, and techniques to build intelligent systems* (2nd ed.). O'Reilly.

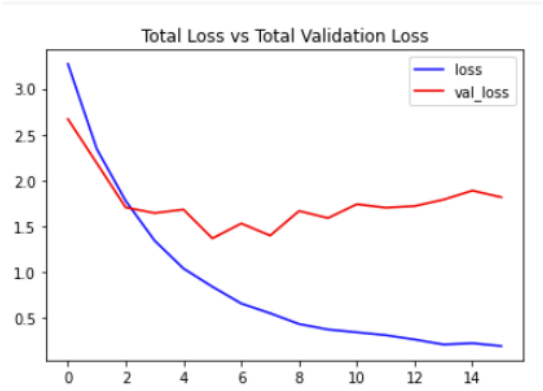
Appendix A

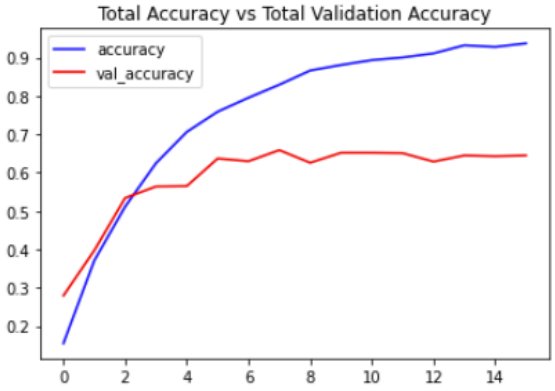
Dataset Summary



Appendix B

LRNC (ConvLSTM) Best model





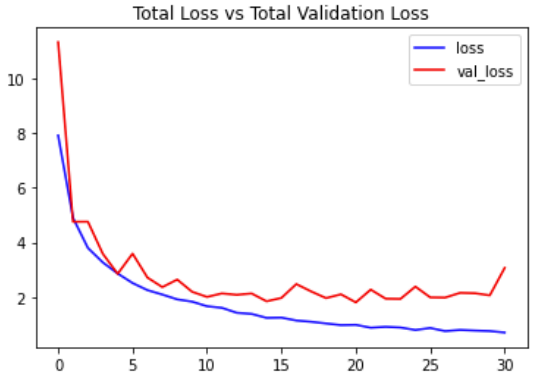
Model: "sequential"

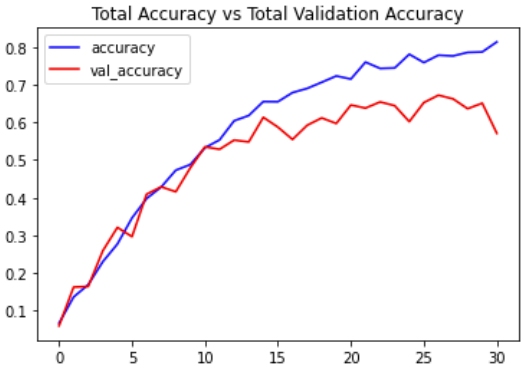
Layer (type)	Output Shape	Param #
conv_lstm2d (ConvLSTM2D)	(None, 20, 62, 62, 4)	1024
max_pooling3d (MaxPooling3D)	(None, 20, 31, 31, 4)	0
time_distributed (TimeDistributed)	(None, 20, 31, 31, 4)	0
conv_lstm2d_1 (ConvLSTM2D)	(None, 20, 29, 29, 8)	3488
max_pooling3d_1 (MaxPooling3D)	(None, 20, 15, 15, 8)	0
time_distributed_1 (TimeDistributed)	(None, 20, 15, 15, 8)	0
conv_lstm2d_2 (ConvLSTM2D)	(None, 20, 13, 13, 14)	11144
max_pooling3d_2 (MaxPooling3D)	(None, 20, 7, 7, 14)	0
time_distributed_2 (TimeDistributed)	(None, 20, 7, 7, 14)	0
conv_lstm2d_3 (ConvLSTM2D)	(None, 20, 5, 5, 16)	17344
max_pooling3d_3 (MaxPooling3D)	(None, 20, 3, 3, 16)	0
flatten (Flatten)	(None, 2880)	0
dense (Dense)	(None, 50)	144050

Total params: 177,050
Trainable params: 177,050
Non-trainable params: 0

Appendix C

```
=================================================================
 time_distributed (TimeDistr  (None, 20, 64, 64, 16)   448
 ibuted)

 time_distributed_1 (TimeDis  (None, 20, 16, 16, 16)   0
 tributed)

 time_distributed_2 (TimeDis  (None, 20, 16, 16, 16)   0
 tributed)

 batch_normalization (BatchN  (None, 20, 16, 16, 16)   64
 ormalization)

 time_distributed_3 (TimeDis  (None, 20, 16, 16, 32)   4640
 tributed)

 time_distributed_4 (TimeDis  (None, 20, 4, 4, 32)     0
 tributed)

 time_distributed_5 (TimeDis  (None, 20, 4, 4, 32)     0
 tributed)

 batch_normalization_1 (Batc  (None, 20, 4, 4, 32)     128
 hNormalization)

 time_distributed_6 (TimeDis  (None, 20, 4, 4, 64)     18496
 tributed)

 time_distributed_7 (TimeDis  (None, 20, 2, 2, 64)     0
 tributed)

 time_distributed_8 (TimeDis  (None, 20, 2, 2, 64)     0
 tributed)

 batch_normalization_2 (Batc  (None, 20, 2, 2, 64)     256
 hNormalization)

 time_distributed_9 (TimeDis  (None, 20, 2, 2, 64)     36928
 tributed)

 time_distributed_10 (TimeDi  (None, 20, 1, 1, 64)     0
 stributed)

 batch_normalization_3 (Batc  (None, 20, 1, 1, 64)     256
 hNormalization)

 flatten (Flatten)           (None, 1280)              0

 dense (Dense)               (None, 2056)              2633736

 output (Dense)              (None, 50)                102850

=================================================================
Total params: 2,797,802
Trainable params: 2,797,450
Non-trainable params: 352
_________________________________________________________________
Model Created Successfully!
```





Classification Report

	precision	recall	f1-score	support
0	0.90	1.00	0.95	36
1	1.00	0.69	0.82	13
2	1.00	0.19	0.32	31
3	0.96	1.00	0.98	27
4	0.88	0.91	0.89	23
5	0.95	0.79	0.86	24
6	0.86	0.86	0.86	35
7	0.86	0.63	0.73	30
8	0.88	0.85	0.86	26
9	1.00	0.50	0.67	12
10	0.86	0.82	0.84	22
11	0.61	0.85	0.71	26
12	0.58	0.52	0.55	21
13	0.94	0.50	0.65	34
14	0.54	0.81	0.65	27
15	0.87	0.68	0.76	40
16	1.00	0.60	0.75	10
17	0.68	0.75	0.71	36
18	0.37	0.53	0.43	19
19	0.73	0.86	0.79	37
20	0.53	1.00	0.69	10
21	1.00	0.24	0.39	25
22	0.71	0.95	0.81	41
23	0.55	0.88	0.67	32
24	0.84	0.73	0.78	22
25	0.60	0.60	0.60	5
26	0.62	0.82	0.71	22
27	0.95	0.71	0.82	28
28	0.75	0.89	0.81	27
29	0.52	0.65	0.58	20
30	0.22	0.42	0.29	19
31	0.00	0.00	0.00	4
32	0.79	0.43	0.56	35
33	0.55	0.84	0.67	19
34	0.60	0.14	0.23	21
35	0.00	0.00	0.00	5
36	0.35	0.82	0.49	11
37	0.67	0.50	0.57	28
38	0.33	0.31	0.32	13
39	0.44	0.78	0.56	18
40	0.67	0.80	0.73	5
41	0.32	0.47	0.38	15
42	0.93	0.52	0.67	25
43	0.35	0.30	0.32	20
44	0.00	0.00	0.00	6
45	0.41	0.58	0.48	12
accuracy			0.67	1017
macro avg	0.66	0.62	0.61	1017
weighted avg	0.72	0.67	0.66	1017

Accuracy Score: 0.672566371681416
Root Mean Square Error: 10.247526502040847