Adrian Marinovich
Springboard Data Science Career Track
Capstone Project #2 - Final Report
February 20, 2019

Project: Classification of human actions in videos

Problem statement: Why it's a useful question to answer and details about the client:

What is the problem you want to solve?
The problem is to classify human actions as seen in videos.

Who is your client and why do they care about this problem?
A range of clients may be interested in human action classification in videos. Potential applications include the development of instructional tools for robots, accident/injury prevention in industrial sites, pools and recreational areas, and monitoring of secure locations. Such a tool might also allow robots and self-driving cars to better predict the intentions and possible outcomes of human actions and thus allow safer response decisions.

Description of the dataset, how you obtained, cleaned, and wrangled it
The data are obtained from the UCF101 Action Recognition Data Set (http://crcv.ucf.edu/data/UCF101.php), which consists of 13320 videos labelled with 101 action categories. The videos were collected from YouTube, and consist of a diverse set of realistic examples in varied settings. There are five types of action categories: human-object interaction, body-motion only, human-human interaction, playing musical instruments, and sports. There are 25 unique groups of videos per action, with each group consisting of 4-7 video clips segmented from a larger video. The videos have a frame rate of 25 fps, and range in length from 1.06 to 71.04 seconds, with a mean length of 7.21 seconds.

Python code was adapted from https://github.com/harvitronix/five-video-classification-methods to extract the UFC101 data into an appropriately created directory structure, and obtain frame-by-frame JPEG still image file sequences from each video using the FFmpeg multimedia framework. The resulting JPEG images are in 3 colors, with 240 x 320 pixel dimensions.

The data were split into train and test sets using pre-made lists prepared by the UFC research group, which were designed ensure that groups of video clips obtained from the same original video did not get split between sets. The train set comprised 9537

videos, and the validation set comprised 3783 videos. As the data contain arbitrary action categories without an immediate use-case application, and are instead intended for development of models for benchmarking and later training on ‘real-world’ datasets with more relevant classifications, no hold-out set was considered necessary.

Video classification was performed using a variety of both Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRU) recurrent neural network (RNN) models that were fed features extracted from the image sequences using a pretrained InceptionV3 model. The image data, originally with 240 x 320 x 3 shape, were reshaped to 299 x 299 x 3 shape to fit the required input dimensions of the InceptionV3 convolutional neural network (CNN) model. The feature extraction stage produced arrays with a 2048 feature dimension and additional dimension according to sequence length (number of images or video frames). To reduce computational load, a fixed sequence length was used during hyperparameter tuning. The code referenced above approached this constraint by setting 40 frames as the lower limit, and subsampling down all videos to this limit, with a top limit for subsampled videos of 300 frames, resulting in exclusion of 1,306 videos. To allow use of all videos, and to examine a range of fixed sequence lengths for their impact on model accuracy, the code was further modified to replace the standard 40-frame subsampled sequence length with a variable sequence length, of 50, 80 and 100 frames. This was achieved by subsampling longer sequences as before by regular skipping of frames, and additionally by interpolating shorter sequences using repetition of frames at regular intervals. In addition, the code was modified to allow features extraction from the original variable sequence lengths. After tuning, these variable-sequence-based features were fed to the top-performing tuned model using a batch size of 1.

Machine learning was performed using Python, primarily in Keras with a TensorFlow backend, on a stand-alone server equipped with an NVIDIA GTX 1060 GPU with 6 GB memory.

Initial findings from exploratory analysis - Summary of findings:
Overall, the fixed-sequence-length LSTM and GRU models gave top-1 validation accuracies between 64% and 75%. While there was worse performance by the LSTM model, GPU memory constraints limited development of LSTM models. The GRU models were used for building deeper models, and further hyperparameter tuning was performed using these models.

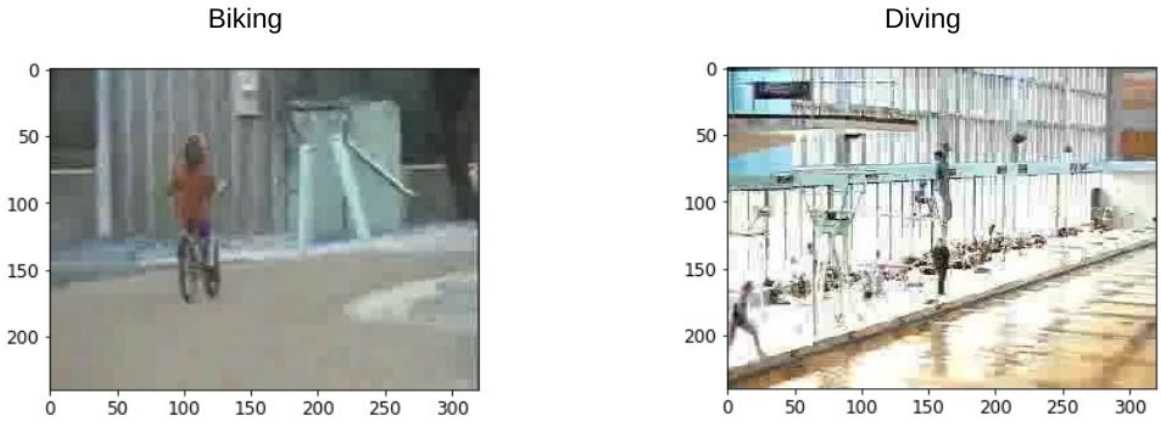
As shown in Table 1, the highest fixed-sequence-length validation accuracy of 75% was attained using a model with 3 GRU layers, using a 50-frame sequence length and batch size of 8. While tuning of batch sizes using the 50-frame sequence reveals improvement in validation accuracy with decreasing batch sizes down to 8, the 80-frame sequence tuning did not show a pronounced association with batch size, scoring 74% accuracy at batch sizes of both 8 and 32.

It is interesting that adding a fourth GRU layer did not improve accuracy, instead resulting in a validation accuracy of 72% when using the 50-frame sequence length and a batch size of 8. Likewise, as shown in Figure 1, the number of trainable parameters does not show a clear association with validation accuracy.

Feeding the top-performing model (using batch size set to 1) with features extracted from variable sequence lengths from original video image sets, resulted in 80% validation accuracy.

Visuals and statistics to support findings

Example images from videos:



Model template:

Layer (type)	Output Shape
RNN input layer	(None, sequence length, 2048)
Additional RNN layers	
dense_1 (Dense)	(None, 512)
dropout_1 (Dropout)	(None, 512)
dense_2 (Dense)	(None, 101)

Top performing model:

Layer (type)	Output Shape	Param #
gru_1 (GRU)	(None, sequence length, 2048)	25171968
gru_2 (GRU)	(None, sequence length, 2048)	25171968
gru_3 (GRU)	(None, 2048)	25171968
dense_1 (Dense)	(None, 512)	1049088
dropout_1 (Dropout)	(None, 512)	0
dense_2 (Dense)	(None, 101)	51813

Trainable params: 76,616,805

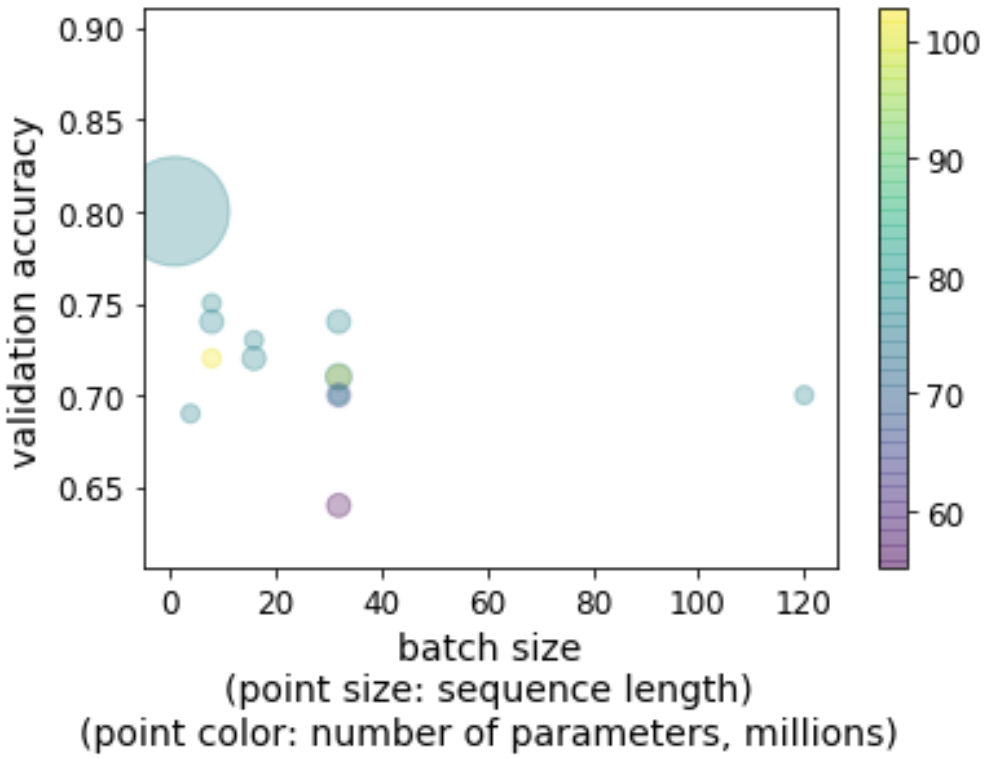
Table 1. Hyperparameter tuning results.

RNN type	Node number by RNN layer[1]				Trainable parameter total, in millions	Sequence length, in frames	Batch size	Epochs[2]	Validation accuracy
	1	2	3	4					
LSTM	2048	1024	1024		55.1	80	32	14	0.64
GRU	2048	4096			102.8	80	32	12	0.71
GRU	2048	2048	2048		76.6	50	4	4	0.69
GRU	2048	2048	2048		76.6	50	8	13	0.75
GRU	2048	2048	2048		76.6	50	16	10	0.73
GRU	2048	2048	2048		76.6	50	32	10	0.70
GRU	2048	2048	2048		76.6	50	120	26	0.70
GRU	2048	2048	2048		76.6	80	8	13	0.74
GRU	2048	2048	2048		76.6	80	16	16	0.72
GRU	2048	2048	2048		76.6	80	32	26	0.74
GRU	2048	2048	2048		76.6	100	32	17	0.71
GRU	2048	2048	2048		76.6	Variable (29-1776)	1	3	0.80
GRU	2048	2048	1024	1024	66.7	80	32	20	0.70
GRU	2048	2048	2048	2048	101.8	50	8	10	0.72

[1]Layer 1 is input layer

[2]Number of epochs until validation loss stopped improving

Figure 1. Plot of validation accuracy against selected tuning hyperparameters.
(Variable sequence length is represented by maximum length of 1776)



Future directions:
Due the limitations of the above approach, with single stream training, and lack of readily available visualization tools for LSTM/GRU models, another modeling approach may be considered, which implements two-stream temporal-spatial neural network models using videos of variable lengths within an end-to-end training framework that contains convolutional layers that can be visualized.

Project link:

https://github.com/adriatic13/springboard/tree/master/dsct_capstone2

References:

https://crcv.ucf.edu/data/UCF101.php

https://github.com/fchollet/deep-learning-with-python-notebooks

https://github.com/harvitronix/five-video-classification-methods