Real-time Action Recognition in Livestock Surveillance feeds

Aniket Shirke
anikets@illinois.edu

Jay Joshi
jajoshi2@illinois.edu

Presentation link: https://uofi.box.com/s/2vej9k2ycge08gr78xvmm56y0r1vvz6k

1. Introduction

In recent years, video-based action recognition has become a popular field of research in deep learning. There are a large number of applications in the domain of robotics, healthcare, human computer interaction systems, traffic monitoring and control, etc. Over the decade, this technology has evolved to produce reliable outcomes in real-time human activity recognition based on large collection of data sets such as UCF101 and Sports1M. Another concurrent trend has been the emergence of edge computing. The computational power available at the source of data collection, commonly referred to as "the Edge", can be used in performing reasonably complex operations on-site.

1.1. Problem Description

In this project, we are specifically focused on bringing together the ideas from Edge Computing and Computer Vision to implement a livestock monitoring tool for animal welfare using live activity feeds. We intend to build a real-time system, using a Raspberry Pi and Intel's Neural Compute Stick, to recognize the actions of animals in livestock surveillance feeds. In order to build a prototype, the Department of Animal Sciences have provided us with an annotated video dataset of pigs carrying out a task called Novel Object Recognition (NOR), wherein a pig either explores the area in which it is contained, or investigates the novel object of interest. Using this dataset, we aim to classify the movements of the pig into two distinct categories: *"Explore"* and *"Investigate"*, using action recognition techniques adopted in Computer Vision.

1.2. Summary of Approach

We started by learning about the theory related to action recognition by completing an exhaustive literature survey, and it is summarized in section 1.2. We then built a custom Long-term Recurrent Convolutional Network (LRCN)[10] model for the UCF101 dataset [14], and deployed this model on a Raspberry Pi powered by Intel's Neural Compute Stick as described in section 3.2. We trained three action recognition models, namely LRCN, Convolutional 3D (C3D) [15] and Temporal Shift Module (TSM) [13] on our NOR dataset and compared the models on analytical metrics such as accuracy, precision-recall, frame-level hit-miss rates, and computational metrics such as frames-per-second, multiply-accumulate operations (GMACs), model parameters and memory.

1.3. Related Work

Traditional Approach: Initial approaches in Computer Vision algorithms can be majorly branched into three conventional steps to predict object or actions in image frames. First, high dimensional features in local region of a video are extracted densely or sparsely from points of interest. Second, these extracted features are then combined using hierarchical or k-means clustering into fixed size video level feature encoding. Third, use a classifier like Random Forest or Support Vector Machines to predict using those encoders. A state-of-the-art technique called improved Dense Trajectories (iDT) [16] was devised using densely sampled trajectory features. Later, 3D convolutions gave breakthrough and made major changes in the way spatio-temporal information was combined.

Deep Learning Approach: In 2014, Andrej Karpathy [12] introduced multiple ways to fuse temporal dimension using 2D pre-trained network of convolutions. Experimentation with late fusion, early fusion, single frame fusion, and slow fusion did not produce significant results due to inability to capture motion features. To deal with this challenge, Simmoyan and Zisserman [7] explicitly modeled two network streams; one of them stacked optical flow vectors and one for spatial context. This method reflected results with improved performance but it had some drawbacks. Predictions were obtained by taking average of predictions from sampled clips. This process misses out on long term temporal information which were supposed to be stored in learned features.

Advancing research in action recognition led to development of fusion algorithms such as Long-Term Recurrent Convolutional Network (LRCN) [11] which combines RNN with CNN instead of stream-based network to improve encoder-decoder architecture for video representa-

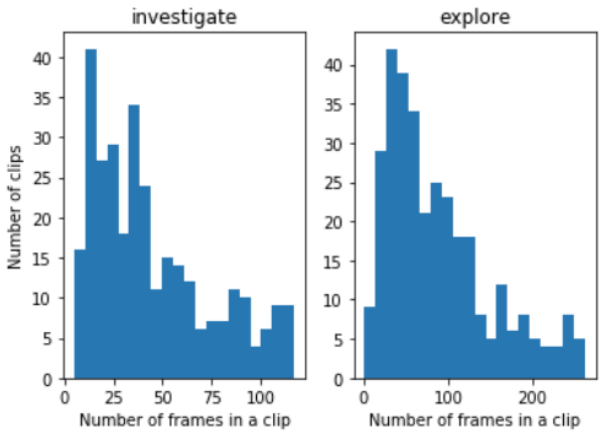

Figure 1. Histogram of number of frames per action clip

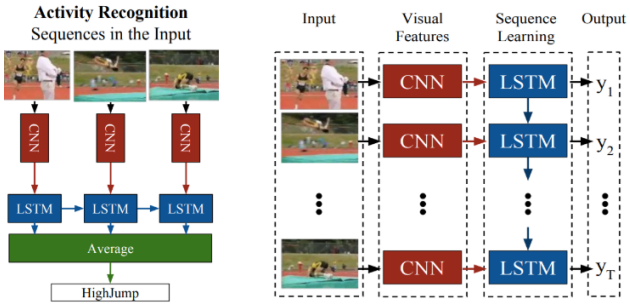

Figure 2. LRCN pipeline [11]

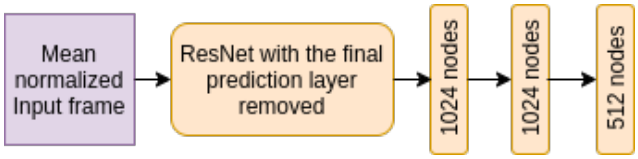

Figure 3. LRCN: CNN encoder

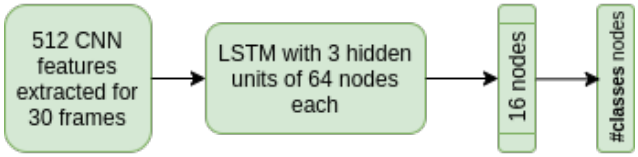

Figure 4. LRCN: RNN decoder

tion. Yao et al. [18] introduced a mechanism using CNN-RNN encoder decoder framework for capturing global context and local spatiotemporal features. Later, algorithms such as I3D[8], T3D[9], TSN[17], and others contributed heavily in improving temporal 3D convolutional networks.

We specifically wanted to pick models which did not require any optical flow computation in real-time and only required RGB frames as input during inference, hence we picked LRCN (RNN + 2D CNN features), C3D (Classifier + 3D CNN features) and TSM (Temporal shifting of 2D CNN features) as our model architectures.

2. Details of approach

2.1. Dataset

To have a consistent notation, a single clip is defined as a continuous set of frames (images) and a video is defined as a continuous set of clips.

Pig behavioral dataset: We have used the Pig behavioral dataset provided by the Department of Animal Sciences at the University of Illinois, Urbana-Champaign. We have 20 videos, which are 5-6 minutes long each, annotated with the time ranges for two actions, namely 'Exploring' and 'Investigating' in a csv file. The video dataset is available at https://uofi.box.com/s/1tihqo6sxwh1f0g6413rw12ij0o8tnk0

Analysis: We pre-processed the raw videos to extract clips of relevant actions using the csv file containing the annotations. We completed an analysis of how many clips existed for a given type of activity and plotted a histogram of the number of frames per clip, as depicted in Figure 1. To maintain consistency while training the deep learning models, we removed all the clips which contained less than 60 frames and fragmented all the bigger clips into smaller clips having exactly 60 frames each. We effectively have 1134 clips for class 'explore' and 1109 clips for class 'investigate', and every clip contains 60 frames each. Since the dataset is balanced, there is no need to remove bias by boosting the samples of the data under consideration.

2.2. Architectures used

2.2.1 LRCN

We adopt the approach described in [11]. The general idea is to encode every frame using a Convolutional Neural Network to capture spatial features in the frame and then pass these features from consecutive frames through a Recurrent Neural Network to capture the temporal variations in the frame. Figure 2 summarizes our approach.

Figure 3 depicts the architecture we have used for our CNN encoder and Figure 4 depicts the RNN decoder. We remove the final prediction layer of a pretrained ResNet18 network to use it as a CNN feature extractor. We modified the Github tutorial [1] to train our own LRCN model.

2.2.2 C3D

We use a C3D [15] network pretrained on the Sports1M dataset to capture the spatiotemporal variation in a clip, and use the output of the fc6 layer to extract a single 4096 di-

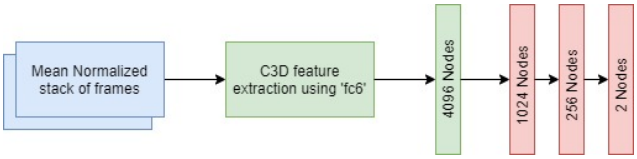

Figure 5. C3D architecture

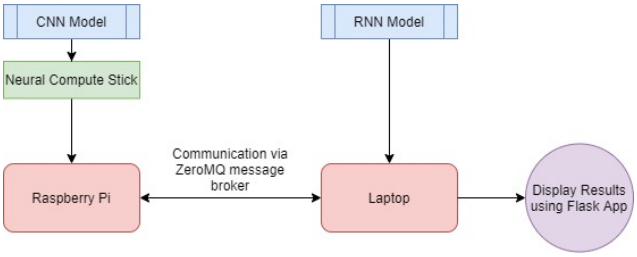

Figure 6. Deployment using Raspberry Pi

mensional feature vector for a clip containing 16 frames. We then build a Binary Classifier which uses these feature vectors as input to predict the class label, as depicted in Figure 5

2.2.3 TSM

We adopt one of the various approaches described in [13]. The key idea is similar to LRCN, where we extract features for each frame in a clip using a backbone Convolutional Neural Network. But instead of training a Recurrent Neural Network, the features are pooled using a temporal shift module to obtain a prediction for the clip. We modified the open-source Github code [2] to train our TSM network using with ResNet18 as the backbone, using only the RGB channels.

2.3. Real time web application

For our prototype system, we have used the LRCN architecture to build a proof of concept real-time web application. Once the LRCN model is trained, we store the weights in a .pth file. We use Intel's Neural Compute Stick (NCS) 2 [3] to offload our CNN model onto a Raspberry Pi. We first convert the Pytorch model into an ONNX model file and then use Intel's model optimizer[4] to obtain an executable NCS binary.

Figure 6 depicts our system architecture for deploying our Real-time action recognition application. We use a Pi-Camera to extract frames and do a forward pass in the CNN encoder. We utilize the ImageZMQ [5] library to facilitate the communication of frames and extracted features between the Pi and the host laptop. The RNN decoder runs on host laptop to aggregate features and the predictions are displayed along with the frames using a Flask web application.

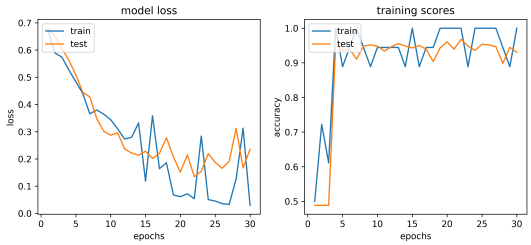

Figure 7. LRCN training curve

3. Results

To generate our results so far, we have used an Intel(R) Xeon(R) CPU E5-2697 machine with 128GB RAM and 28 cores. We were not able to setup the GPUs on campus cluster and hence decided to train our model on a CPU.

3.1. Training Procedure

For the training procedure, we maintain a consistent training and validation split of 75%-25%. Hence, we use 1682 clips for training and 561 clips for validation.

3.1.1 LRCN

Each frame is resized to 224x224 and normalized using the mean and standard deviation of the ImageNet dataset. We minimize the cross entropy loss for 30 epochs with a learning rate of 0.0001 using Adam optimizer. Figure 7 depicts the model performance obtained after 17 hours of training. We can see that the model is performing really well after 5 epochs itself, and we eventually obtain an accuracy of 93% on the validation set. In the discussion section, we discuss the different parameter tunings required to converge the model for LRCN, which was surprisingly difficult to train for our dataset.

3.1.2 C3D

Feature Extraction: We follow a different approach while training this model. Each frame is resized to 128x171 and normalized using the mean and standard deviation used for training the C3D model on the Sports1M dataset. We run a script to extract C3D features for all the clips in our dataset, and store the output, which is a 4096 dimensional vector per clip, in a numpy array. This pre-computation step took 15 minutes to run on our dataset which contains 2243 clips.
Binary Classifier: We train a binary classifier using the pre-computed features, which minimizes the binary cross entropy loss for 20 epochs with a learning rate of 0.001 using Adam optimizer. Since the majority of the computation is done in the Feature Extraction step itself, the model is trained within 4 minutes and the performance is depicted in

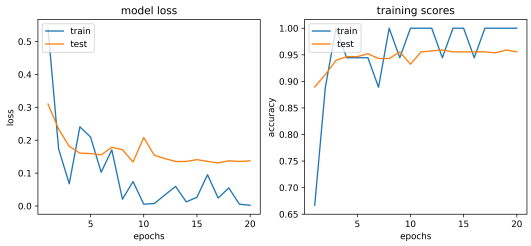

Figure 8. C3D training curve

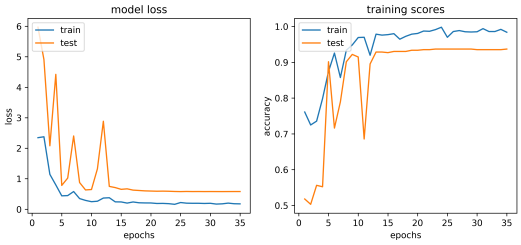

Figure 9. TSM training curve

Figure 8. We can see that the model reaches an accuracy of over 90% after the second epoch itself and we eventually obtain an accuracy of more than 95% on the validation set.

3.1.3 TSM

Since we are using ResNet18 as the backbone network, we apply similar preprocessing to the frames as mentioned in section 3.1. We minimize the cross entropy loss for 35 epochs with an initial learning rate of 0.02 using SGD optimizer with a momentum of 0.9 and a weight decay of 0.0005. We adjust the learning rate to 0.002 and 0.0002 after 12 and 25 epochs respectively. Figure 9 depicts the model performance obtained after 7.5 hours of training. The model saturates at an accuracy of 93% on the validation set after 15 epochs.

3.2. Precision Recall curve

We plotted the Precision-Recall curve for our model by evaluating it on 561 clips in the validation set. We set the class 'investigate' as our positive label, and from Figure 10, we can see that C3D and TSM produce very comparable results with a mean Average Precision(mAP) of 0.944 and 0.965 respectively. LRCN achieves a mAP of 0.892.

3.3. Raspberry Pi deployment

We train our custom LRCN model with a ResNet18 backbone on the UCF101 dataset. Figure 11 depicts the model performance obtained after 45 hours of training. We notice the model is converging to achieve a satisfactory point. We pick the checkpoint for the 28th epoch, which

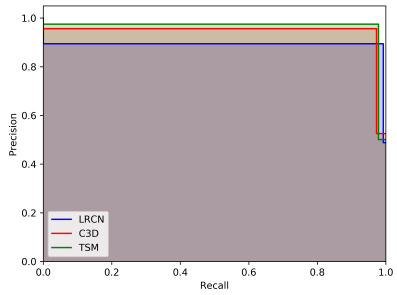

Figure 10. Precision-Recall curves on the testing dataset

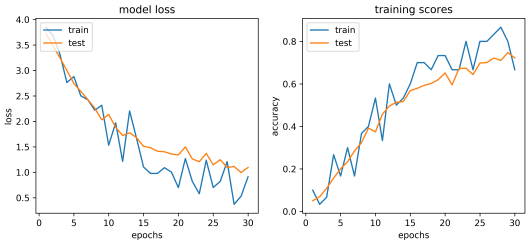

Figure 11. CNN+RNN Model performance on UCF101

achieved a fairly high accuracy on both the training and the validation set, to offload on our Raspberry Pi.

We deployed our model on a Raspberry Pi 4 and used Intel's Neural Compute Stick 2 to accelerate the computations for our ResNet18 model. Including network latency, we achieved a rate of 5 frames per second in the end-to-end computation pipeline. One can view the result on this link: https://shorturl.at/vAMUX

3.4. Annotation Metric

To measure the visual performance of different models, we annotated the 20 surveillance videos in our dataset using LRCN, C3D and TSM. The results can be found here https://uofi.box.com/s/fi42b26vc1y87n989deh3tpzi7ru3blr. We devise a new metric to evaluate the annotations obtained by the three models.

Consider each video V in our dataset D. Let $Gt(f)$ and $Pred(f)$ be functions which output the groundtruth and predicted action class, i.e. 0 (explore) or 1 (investigate) for each frame f in video V. $FC(V)$ returns the frame count for video V.

We define a hit $h(f)$ and a miss $m(f)$ on frame f as follows:

$$h(f) = \begin{cases} 1 & \text{if } Gt(f) = 1 \text{ and } Pred(f) = 1 \\ 0 & \text{otherwise} \end{cases}$$

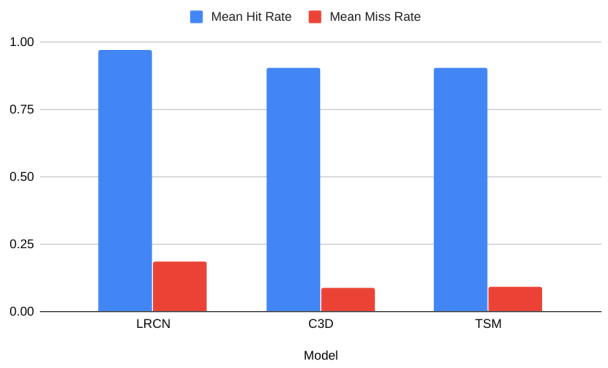

Figure 12. Annotation Metric

$$m(f) = \begin{cases} 1 & \text{if } Gt(f) = 0 \text{ and } Pred(f) = 1 \\ 0 & \text{otherwise} \end{cases}$$

We define the hit rate $H(V)$ and the miss rate $M(V)$ for a video V as follows:

$$H(V) = \frac{\sum_{f \in V} h(f)}{\sum_{f \in V} Gt(f)} \quad M(V) = \frac{\sum_{f \in V} m(f)}{FC(V) - \sum_{f \in V} Gt(f)}$$

Intuitively, we can think of hit rate as the rate at which our model correctly predicted the action 'investigate', and miss rate as the rate at which falsely labelled a frame belonging to the 'explore' class as 'investigate'. Hence, we would like to maximize the hit rate as much as possible while keeping the miss rate low. Note that this metric is evaluated at the frame-level and is different from a clip-level metric like accuracy and precision-recall.

We can see that LRCN has the highest mean hit rate of 97.2%, but it suffers from a high mean miss rate as well, 18.6%. This means that LRCN has a higher tendency to label a frame as an 'investigate' frame and has a weaker discriminative power compared to C3D or TSM. Both C3D and TSM have comparable hit rates of 90.5% and 90.7% respectively, and very low miss rate of 8.7% and 9.0% respectively.

4. Discussion

4.1. Deployment issues on Raspberry Pi

We initially intended to use C3D to extract spatiotemporal features from a video stream. But we carried a small proof of concept to arrive at the conclusion that the Intel Neural Compute Stick does not support 3D convolutions yet, and our suspicions were backed by their official website [6]. Hence, we could not deploy our C3D model on a Raspberry Pi. We were not able to offload the Temporal Shift Module as there was no clear documentation available to deploy it on Intel's Neural Compute Stick. The author provided the deployment procedure on an Nvidia Jetson Nano, but we did not have that particular hardware to deploy our custom TSM model.

Table 1. Computational Performance Metrics

Model	FPS	GMACs	# Parameters	Memory (in MB)
LRCN	18.20	109.43	13.30 M	**51.85**
C3D	110.84	17.5	65.67 M	286.69
TSM	**125.19**	**14.57**	**11.18 M**	86

4.2. Computational Performance

Due to the shortcomings described in the previous sections, we evaluated the different computational metrics on the server described in section 2.3. TSM achieves the maximum frames rate per second due to its computational efficiency obtained due to lesser number of parameters. During our literature survey, we noticed that multiply-accumulate operations and number of parameters are critical computational metrics, and hence we profiled our model in Pytorch to obtain GMACs (multiply-accumulate operations in billions) and the number of model parameters. TSM once again outperforms C3D and LRCN in these metrics. The only positive aspect about LRCN is the compact memory requirements for the model checkpoint.

4.3. LRCN training iterations

Training a custom LRCN model for our dataset turned out to be challenging and we modified various parameters to obtain a satisfactory model convergence. We performed a forward ablation study to improve our model incrementally. We initially started out with the model architecture used for training on the UCF101 dataset and obtained the following training curve as depicted in Figure 13.

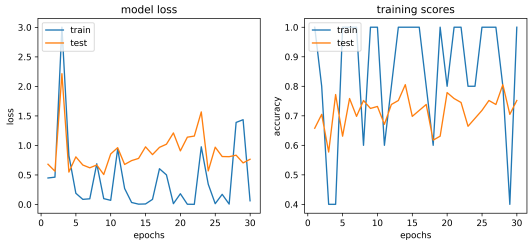

Figure 13. Trained using model architecture used for UCF101

Since no clear convergence was observed, we decreased the learning rate from 0.001 to 0.0001. The result was not promising, as we can see that the model overfitted the underlying dataset, and as a result, the model loss increased as the number of epochs progressed as depicted in Figure 14.

To combat overfitting, we added a dropout of 0.3 in the fully connected convolutional layers of our ResNet18 backbone and the LSTM network. As seen in Figure 15, adding

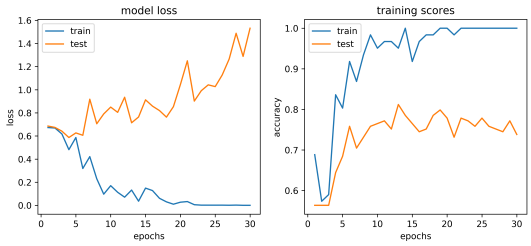

Figure 14. Result obtained by decreasing the learning rate

dropout clearly did not help in decreasing the loss on our validation set.

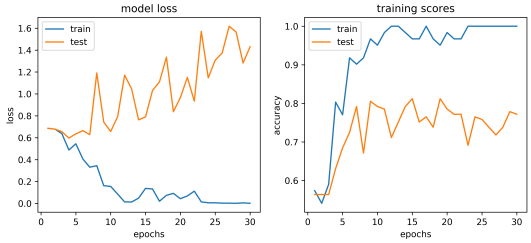

Figure 15. Adding dropout did not help in resolving overfitting

We adopted to simplify our model by decreasing the number of hidden nodes in the LSTM network from 512 to 256 and reducing the number of nodes in the fully connected layer of RNN from 128 to 32. This partially resolved the problem of overfitting as seen in Figure 16. We further tweaked some parameters to obtain the final training curve as depicted in Figure 7.

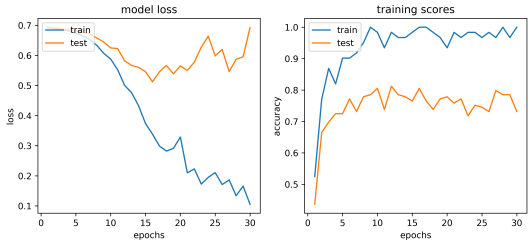

Figure 16. Simplifying the RNN model helped in resolving overfitting

4.4. Annotation Performance

In the above sections, we have discussed different challenges of model deployment in Raspberry Pi and quantitatively compared the computational efficiency of C3D, TSM, and LRCN. Now, we introduce a detailed analysis of resulting performance on annotated videos. We evaluate classification trends in 20 processed videos.

Classification accuracy: Annotation metrics introduced in section 3.4 is used for measuring performance index of trained activity recognition models. We calculate mean hit rate and mean miss rates for every 30 frames. As seen in Figure 12 annotation metrics of LRCN has the worst miss rate as compared to models obtained from C3D and TSM. Also, we have real-time accuracy per frame for each model which denotes the confidence score of predicted classification. However, both of these metrics do not help us understand what action(s) in the video contributed to poor confidence score for a given prediction. So, we further carry out an experiment to understand model limitations and identify challenges for recognising activity in certain situations. With granular visualization of results (given in Appendix), we see specific trends in hit-miss rate observed. Annotated videos with higher miss rates are chosen to find conclusive evidence and investigate some of the common pitfalls of model.

Based on the sniffing behavior of pig in the annotated video, we branch out recurring issues into the following two hypothesis:

1. Recognising constantly changing activity: In this scenario, we expect pig to rapidly toggle between two actions. Such a situation has come out to be one of the prominent reason for observed dip in accuracy for all the models.

2. Consistency in recognising constant activity: Here, we vouch for the robustness of model to predict continuing action accurately. This is another major challenge faced by obtained action recognition models. Usually, this is random and occurs for both the categories of classification.

For example:

- **Video 543**[Timestamp: 1:10s, 1:18s]: Issue 1 and 2 are relevant to both LRCN and C3D as models wrongly tagged the action as 'Investigate.' TSM outperformed others because it accounts for smooth temporal pooling of features.
- **Video 704:** [Timestamp: 2:04s] TSM and C3D does not cope up with rapidly changing actions to predict a wrong classification. LRCN works well for constantly toggling activity.
- **Video 862:** [Timestamp: 1:20s] Constant activity has variations in classification confidence score. TSM and LRCN observes less fluctuations as compared to C3D.

Additionally, there are many more examples to validate the above mentioned hypothesis and evaluate performance of annotated videos. In sum, LRCN and TSM shows better results in both the cases. The only downside of LRCN is that its accuracy is on a lower side than the other two models but with more stability. In order to find optimal model, we can use the information given in Table 1 along with analysis from hypothesis testing. Thus, it can be concluded that

TSM achieves superior output at a higher FPS, low computational latency, and generates reasonably fair results in classifying multiple activity scenario for our pig dataset.

5. Conclusion

Livestock monitoring is crucial for the effective functioning of modern farms, but is bottlenecked on the ability to analyze enormous volumes of information. Edge Computing and Computer Vision is widely viewed as a key technology to relieve this bottleneck. Through this project, we built a real-time web application framework which deploys a Long-term Recurrent Convolutional Network for action recognition. We compared LRCN, C3D and TSM on the basis of various analytical and computational metrics, and also explored some key challenges involved in the deployment of these models using state-of-the-art Neural Network accelerators on a Raspberry Pi.

References

[1] https://github.com/HHTseng/video-classification.

[2] https://github.com/mit-han-lab/temporal-shift-module.

[3] https://software.intel.com/en-us/neural-compute-stick.

[4] https://docs.openvinotoolkit.org/latest/_docs_MO_DG_Deep_Learning_Model_Optimizer_DevGuide.html.

[5] https://github.com/jeffbass/imagezmq.

[6] https://docs.openvinotoolkit.org/latest/_docs_IE_DG_supported_plugins_Supported_Devices.html.

[7] Karen Simonyan Andrew and Zisserman. Two-stream convolutional networks for action recognition in videos. In *arXiv:1406.2199v2*, 2014.

[8] Joao Carreira and Andrew Zisserman. Quo vadis, action recognition? a new model and the kinetics dataset, 2017.

[9] Ali Diba, Mohsen Fayyaz, Vivek Sharma, Amir Hossein Karami, Mohammad Mahdi Arzani, Rahman Yousefzadeh, and Luc Van Gool. Temporal 3d convnets: New architecture and transfer learning for video classification, 2017.

[10] Jeff Donahue, Lisa Anne Hendricks, Sergio Guadarrama, Marcus Rohrbach, Subhashini Venugopalan, Kate Saenko, and Trevor Darrell. Long-term recurrent convolutional networks for visual recognition and description. In *CVPR*, 2015.

[11] Donahue et al. Long-term recurrent convolutional networks for visual recognition and description. In *arXiv:1411.4389*, 2014.

[12] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. Large-scale video classification with convolutional neural networks.

[13] Ji Lin, Chuang Gan, and Song Han. Tsm: Temporal shift module for efficient video understanding, 2018.

[14] Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. Ucf101: A dataset of 101 human actions classes from videos in the wild, 2012.

[15] Du Tran, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. Learning spatiotemporal features with 3d convolutional networks, 2014.

[16] Heng Wang and Cordelia Schmid. Action recognition with improved trajectories. In *ICCV*, 2013.

[17] Limin Wang, Yuanjun Xiong, Zhe Wang, Yu Qiao, Dahua Lin, Xiaoou Tang, and Luc Van Gool. Temporal segment networks: Towards good practices for deep action recognition, 2016.

[18] Li Yao, Atousa Torabi, Kyunghyun Cho, Nicolas Ballas, Christopher Pal, Hugo Larochelle, and Aaron Courville. Describing videos by exploiting temporal structure, 2015.

A. Annotation Metric for every model

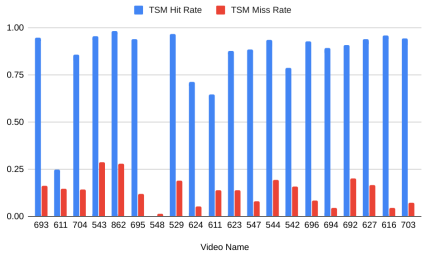

Figure 17. Hit and Miss Rate for TSM

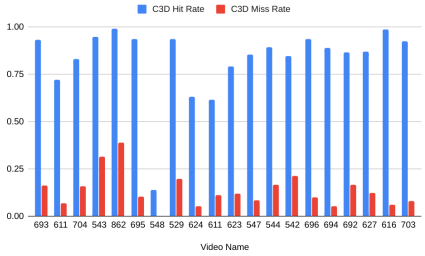

Figure 18. Hit and Miss Rate for C3D

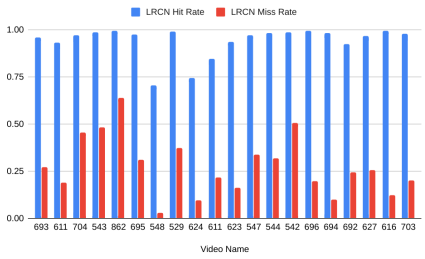

Figure 19. Hit and Miss Rate for LRCN