INFOMCV Assignment 5 Report

Martino Fabiani – Taher Jarjanazi (Group 20)

I. ARCHITECTURE CHOICES

1. Frame CNN architecture:

In constructing the structure of a neural network suitable for frame classification, we started with the GoogLeNet network. Compared to other popular networks like ResNet18 or ResNet50, we found that GoogLeNet gives similar or higher performance with less parameters, which makes it ideal for a starting network for this project. The network has the following structure:

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 64, 112, 112]	9,408
BatchNorm2d-2	[-1, 64, 112, 112]	128
MaxPool2d-3	[-1, 64, 56, 56]	0
Conv2d-4	[-1, 64, 56, 56]	4,096
BatchNorm2d-5	[-1, 64, 56, 56]	128
Conv2d-6	[-1, 192, 56, 56]	110,592
BatchNorm2d-7	[-1, 192, 56, 56]	384
MaxPool2d-8	[-1, 192, 28, 28]	0
Conv2d-9	[-1, 64, 28, 28]	12,288
BatchNorm2d-10	[-1, 64, 28, 28]	128
Conv2d-11	[-1, 96, 28, 28]	18,432
BatchNorm2d-12	[-1, 96, 28, 28]	192
Conv2d-13	[-1, 128, 28, 28]	110,592
BatchNorm2d-14	[-1, 128, 28, 28]	256
Conv2d-15	[-1, 16, 28, 28]	3,072
BatchNorm2d-16	[-1, 16, 28, 28]	32
Conv2d-17	[-1, 32, 28, 28]	4,608
BatchNorm2d-18	[-1, 32, 28, 28]	64
⋮	⋮	
Conv2d-115	[-1, 384, 7, 7]	319,488
BatchNorm2d-116	[-1, 384, 7, 7]	768
Conv2d-117	[-1, 192, 7, 7]	159,744
BatchNorm2d-118	[-1, 192, 7, 7]	384
Conv2d-119	[-1, 384, 7, 7]	663,552
BatchNorm2d-120	[-1, 384, 7, 7]	768
Conv2d-121	[-1, 48, 7, 7]	39,936
BatchNorm2d-122	[-1, 48, 7, 7]	96
Conv2d-123	[-1, 128, 7, 7]	55,296
BatchNorm2d-124	[-1, 128, 7, 7]	256
MaxPool2d-125	[-1, 832, 7, 7]	0
Conv2d-126	[-1, 128, 7, 7]	106,496
BatchNorm2d-127	[-1, 128, 7, 7]	256
AdaptiveAvgPool2d-128	[-1, 1024, 1, 1]	0
Dropout-129	[-1, 1024]	0
Linear-130	[-1, 1000]	1,025,000

Total params: 6,624,904
Trainable params: 6,624,904
Non-trainable params: 0

Input size (MB): 0.574219
Forward/backward pass size (MB): 60.065002
Params size (MB): 25.272003
Estimated Total Size (MB): 85.911224

This network is structured to receive RGB frames of size (3,224,224). To achieve this structure, during the middle frame extraction process, a frame is preprocessed to make it suitable for the network used, by means of resizing and normalization.

The GoogLeNet network consists of multiple inception blocks. Within this structure, the input data passes through 127 convolutional layers with a kernel size of 1x1, which are used to extract information from the data through convolution without reducing its size but only varying its depth. Each of these is followed by a batch normalization layer, which enables faster and more stable training, improving the model's performance. Additionally, the network comprises several pooling layers, allowing for the reduction of the input data's size while maintaining its depth, thereby reducing processing complexity. The pooling layers progressively reduce the input data's dimensions throughout the structure, starting from a size of (112,112), which is the output of the first convolutional layer processing frames of size (224,224), and reducing them to (7,7).

Finally, the structure includes a flatten layer that reduces the data to a single dimension, followed by fully connected layers to reach a dimension of (1,1000).

Then an additional fully connected layer has been added to the standard GoogleNet structure to adapt its last layer's dimension to the number of present classes, and then return the probabilities obtained with Softmax, resulting in a network compatible with the addressed problem, presenting the following final structure:

MaxPool2d-125	[-1, 832, 7, 7]	0
Conv2d-126	[-1, 128, 7, 7]	106,496
BatchNorm2d-127	[-1, 128, 7, 7]	256
AdaptiveAvgPool2d-128	[-1, 1024, 1, 1]	0
Dropout-129	[-1, 1024]	0
Linear-130	[-1, 12]	12,300
Softmax-131	[-1, 12]	0

Total params: 5,612,204
Trainable params: 5,612,204
Non-trainable params: 0

2. Optical flow CNN architecture:

In building a CNN capable of receiving optical flow from different videos as input and providing one of the 12 selected classes from the HMDB51 database as output, the first step was the extraction of optical flow.

To do this, a function called 'extract_optical_flow' was implemented, which allows the extraction of these flows in stacks of 8 frames. This function, for each video used in the training and test dataset, starts by reading the first frame, resizing it (prev_frame = cv.resize(prev_frame, (224, 224))) to reduce the computational complexity of the data, and converting it to grayscale. Similarly, the next frame is extracted, and the same process is applied. At this point, the optical flow between these two frames is calculated using the function 'calcOpticalFlowFarneback'

This function performs a dense extraction of optical flow (represented by the parameter 0 of the function), considering the entire frame. The process is based on the processing of a three-level pyramid structure with a scale value of 0.5 for optical flow calculation. The operation is iterated three times using a 5x5 window. Finally, the optical flow is scaled by a value of 1.2 and returned as a result.

This process is carried out for the first 8 frames of the video, resulting in data of size (8, 224, 224, 2), where 2 represents the shift of x and y coordinates of the pixels between the subsequent pairs of images of the optical flow.

A CNN was then adapted to receive input data of this size. To do this, it was decided to combine the stack_size dimension at the input data with the depth of the optical flow channels, resulting in data compatible with the 2D convolutional layers of dimensions (16,224,224).

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 32, 224, 224]	4,640
MaxPool2d-2	[-1, 32, 112, 112]	0
Conv2d-3	[-1, 64, 112, 112]	18,496
MaxPool2d-4	[-1, 64, 56, 56]	0
Flatten-5	[-1, 200704]	0
Linear-6	[-1, 256]	51,380,480
Linear-7	[-1, 12]	3,084
Dropout-8	[-1, 12]	0
Softmax-9	[-1, 12]	0

Total params: 51,406,700
Trainable params: 51,406,700
Non-trainable params: 0

Input size (MB): 3.062500
Forward/backward pass size (MB): 24.502228
Params size (MB): 196.100998
Estimated Total Size (MB): 223.665726

The structure follows a standard CNN configuration. The input data first passes through a convolutional layer, which allows for an initial generalization of the input information using a 3x3 kernel size while maintaining the dimensions of the data but increasing its depth. Subsequently, the data proceeds through a pooling layer, which reduces the size of the data provided by the convolutional layer using a 2x2 kernel size while keeping the depth unchanged. The same convolutional and pooling process is then repeated, resulting in further increased depth and subsequent reduction in the size of the processed data.

The use of two convolutional layers was necessary to achieve efficient generalization of the input, yielding valid results without introducing an excessive number of parameters. The data is then flattened using a flatten layer, reducing its structure to a single dimension without introducing new parameters. Finally, the data passes through two fully connected layers, gradually reducing its dimensions, ending with a size matching the number of classes (12). To reduce the risk of overfitting encountered during training, a dropout layer of 0.1 was introduced. This value was chosen as it allows for effective reduction of overfitting while minimally affecting the network. Finally, probabilities attributed to the classifications made by the CNN are extracted through Softmax.

3. Combination into two-stream network:

The construction of a Two Stream Network followed the following procedure. The initial objective was to implement the previous structures used for optical flow and middle frame classification, namely OpticalFlowNet and GoogLeNet, by loading their corresponding weights obtained from the previous structures.

Before proceeding with a fusion of these two branches, it is necessary to process them. Specifically, it is evident that it is necessary to remove from each of the two networks the layers that perform the final flattening process and fully connected layers that lead to the attainment of the final output. This is necessary as the objective of a Two Stream Network is to merge these into a single flow following an initial phase of convolutions and pooling of both input branches, obtaining a single data and continuing with subsequent processing, generalizing the contained information through new layers until reaching a final output suitable for classification.

Following the removal of the last layers, the next step is to adjust the dimensions of the data from each branch to a common value. This is crucial to merge the information in a fusion layer. After observing the shapes of the data in our structures (respectively (1024,7,7) for GoogleNet and (64, 56,56) for OpticalFlowNet), a common dimension of (128,7,7) was set, and new convolutional and pooling layers were added to both networks to reach this dimension.

Starting from GoogleNet, without its last layers, a 1x1 convolutional layer was added that keeps the input dimension unchanged but reduces its depth to 128. This brings the resulting structure to the following:

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 64, 112, 112]	9,408
BatchNorm2d-2	[-1, 64, 112, 112]	128
MaxPool2d-3	[-1, 64, 56, 56]	0
Conv2d-4	[-1, 64, 56, 56]	4,096
BatchNorm2d-5	[-1, 64, 56, 56]	128
Conv2d-6	[-1, 192, 56, 56]	110,592
BatchNorm2d-7	[-1, 192, 56, 56]	384
MaxPool2d-8	[-1, 192, 28, 28]	0
Conv2d-9	[-1, 64, 28, 28]	12,288
BatchNorm2d-10	[-1, 64, 28, 28]	128
Conv2d-11	[-1, 96, 28, 28]	18,432
BatchNorm2d-12	[-1, 96, 28, 28]	192
Conv2d-13	[-1, 128, 28, 28]	110,592
BatchNorm2d-14	[-1, 128, 28, 28]	256
Conv2d-15	[-1, 16, 28, 28]	3,072
BatchNorm2d-16	[-1, 16, 28, 28]	32
Conv2d-17	[-1, 32, 28, 28]	4,608
BatchNorm2d-18	[-1, 32, 28, 28]	64
MaxPool2d-19	[-1, 192, 28, 28]	0
Conv2d-20	[-1, 32, 28, 28]	6,144
BatchNorm2d-21	[-1, 32, 28, 28]	64
⋮	⋮	⋮
Conv2d-123	[-1, 128, 7, 7]	55,296
BatchNorm2d-124	[-1, 128, 7, 7]	256
MaxPool2d-125	[-1, 832, 7, 7]	0
Conv2d-126	[-1, 128, 7, 7]	106,496
BatchNorm2d-127	[-1, 128, 7, 7]	256
Conv2d-128	[-1, 128, 7, 7]	1,179,776

Total params: 6,779,680
Trainable params: 6,779,680
Non-trainable params: 0

Input size (MB): 0.574219
Forward/backward pass size (MB): 60.089600
Params size (MB): 25.862427
Estimated Total Size (MB): 86.526245

Continuing with the optical flow branch, a similar procedure was implemented. Indeed, a convolutional layer and a pooling layer were added. The first, using a kernel size of 2x2 and a stride of 2, allows for complete resizing of the data, bringing it to a shape of (128,28,28). Subsequently, through the pooling layer with a kernel size of 2x2 and a stride of 4, the depth remains unchanged while reducing the output spatial dimensions to the desired common value.

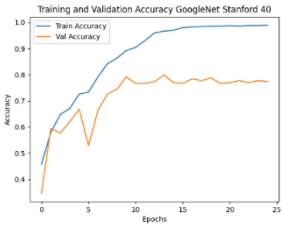
Therefore, the structure of the optical flow branch is as follows:

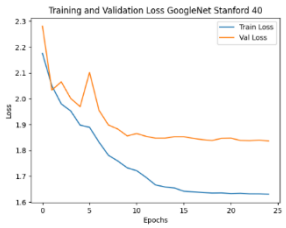
```
Layer (type)                 Output Shape              Param #
================================================================
Conv2d-1                [-1, 32, 224, 224]            4,640
MaxPool2d-2             [-1, 32, 112, 112]                0
Conv2d-3                [-1, 64, 112, 112]           18,496
MaxPool2d-4              [-1, 64, 56, 56]                 0
Conv2d-5                [-1, 128, 28, 28]            32,896
MaxPool2d-6              [-1, 128, 7, 7]                  0
================================================================
Total params: 56,032
Trainable params: 56,032
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 3.062500
Forward/backward pass size (MB): 23.782227
Params size (MB): 0.213745
Estimated Total Size (MB): 27.058472
```

Now that the data is in the same shape, it's possible to proceed with constructing the architecture of the Two Stream Network. First, it consists of a fusion layer, a layer responsible for combining the data provided by the two branches. In the network, this combination is achieved through depth-wise concatenation. The network structure then continues with a standard setup. Firstly, the obtained data passes through a convolutional layer that, using a 1x1 kernel size, conducts an initial convolution process while keeping the input dimensions unchanged, which effectively allows to extract features from the fused input of the two branches. Next is a pooling layer that, maintaining the depth, reduces the data size to (256,3,3). Finally, the data is flattened with a flattening layer, followed by two fully connected layers that transform the data from the shape of (2034) to (64), and ultimately to the number of present classes, which is (12). Through Softmax, the resulting probabilities of classification are obtained from this Two Stream Network.

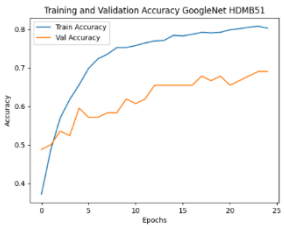
II. Results

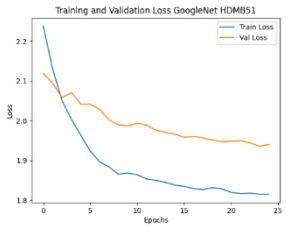
A. Stanford 40 Frames



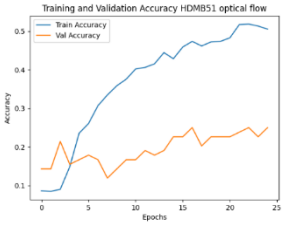


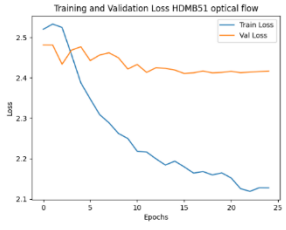
B. *HMDB51 Frames*



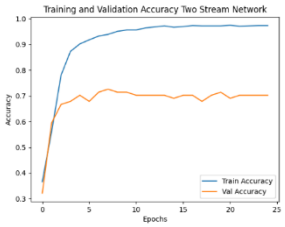


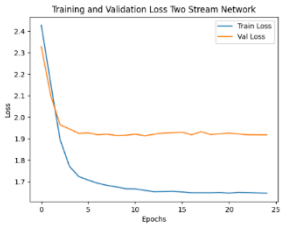
C. HMDB51 Optical flow





D. HMDB51 Two-stream





E. Result table

TABLE I. SUMMARY OF YOUR RESULTS PER MODEL

Dataset	Model specifications			
	Model	**Top-1 accuracy (train / test)**	**Top-1 loss train**	**Model parameters**
Stanford 40	Frames	98.779% / 80.592%	1.631	5,612,204
HMDB51	Frames	80.291% / 52.22%	1.815	5,612,204
HMDB51	Optical flow	51.323% / 22.78%	2.127	51,406,700
HMDB51	Two-stream	97.354% / 54.72%	1.646	6,001,228

III. Discussion of Results

The Stanford 40 frames model is trained using a pretrained GoogLeNet on the ImageNet dataset, which allows it to perform well on the Stanford 40 dataset, although it still overfits on the training data, probably due to the fact that there are relatively few training samples and many of them being similar to the images in ImageNet. Next, the Stanford 40 trained network is fine-tuned on the middle frames from videos in the HMDB51 dataset with corresponding action classes to those in Stanford 40. We observe that the resulting accuracy on the test set after training is significantly lower compared to the first model. This difference is common to tasks of transfer learning such as this where a model is fine-tuned on small number of data points from a different distribution, even with corresponding labels. The model also overfits on the training data, which is reflected by the disparity between the train and test accuracies. Again here, the lack of data quantity paired with an already capable model contributes to the overfitting. The optical flow model trained from scratch on the optical flow calculated from the same videos of the HMDB51 dataset is a custom architecture, which is simple in nature. We see that the performance tends to be quite poor with only 22.78% accuracy on the test dataset. This performance can be explained by the simplistic nature of the optical flow used, which boils down to the movement of pixels along the selected video frames. This feature by itself is not the most useful in identifying types of movements accurately. This might also explain the model overfitting on the training data, where we hypothesize that the network is most likely focusing on irrelevant movements such as those in the background and not on the main subject of interest. Finally, we look at the Two-stream network which combines the trained HMDB51 frames and optical flow models by using

mid-level fusion of the outputs of the last convolutional layers of each model. This model shows improved performance on the test dataset where it scores 2.5% higher accuracy than the HMDB51 frames model alone. This slight performance boost can be attributed to the optical flow model that provided, although less than stellar, movement features that helped improve the results. This network also tends to overfit on the training data, which is to be expected seeing how the models it is made of also do the same.

IV. Link to Model Weights

Assignment 5 Models

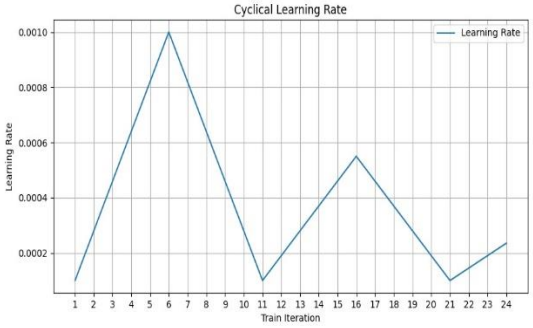
V. Choice Tasks

CHOICE 3: Create a Cyclical Learning rate schedule

This Choice task has been implemented within the classification model of the HMDB51 database by using the mid frame. For the development of a Cyclical Learning Schedule, the procedure was as follows. Inside a PyTorch code, it is possible to initialize a scheduler. Within it, the range of values to vary the learning rate within one training epoch, the desired type of cyclical learning, and the step size are established. Without using this method, the learning rate used was 0.0001; consequently, it was decided to observe the classifier's behavior by varying this value from 0.0001 to 0.001. The "triangular2" mode was selected, which allows for a linear change of the learning rate from the minimum set value to the maximum and then back to the minimum, continuing in a triangular pattern. What makes this mode unique is that at each cycle, the maximum learning rate value is halved, progressively halving in a triangular pattern.

Subsequently, the step size was set to 5. This experimentally chosen parameter represents the number of iterations ('scheduler.step()') after which the learning rate is changed. It was selected to allow for 2 complete cycles of learning rate modification within one training epoch, enabling better observation of its variations. The last parameter set is 'cycle_momentum', set as False since momentum is not being used within the code.

This structure results in a learning rate trend within a training epoch as represented in the following graph:



The implementation of this learning rate update method has resulted in a variation of the final results as follows. In fact, in the absence of this method, the accuracy values obtained were 80.29% during training, 69.05% during validation, and 52.23% during testing. However, following a cyclical variation of the learning rate, the accuracy reached 90.6% during training, 69.05% during validation, and 55.28% during testing.

This evident improvement is due to the variation in the learning rate value, which allows the model to try different values, drastically reducing the risk of reaching local minima and also reducing the risk of overfitting within the training process. Additionally, this variation allows the model to dynamically explore the input data since, with the same complexity, the model will try different learning rate values, finding the most suitable one for the current input data, thus achieving greater generalization capacity and avoiding getting stuck in a local minimum.

CHOICE 4: Use 1x1 convolutions to connect activations between the two branches

Within the Two Stream Network, following the concatenation of the two branches, a convolutional layer with a kernel size of 1x1 was introduced. This layer does not alter the spatial dimensions of the input data, allowing for the extraction of information from the combined input data.

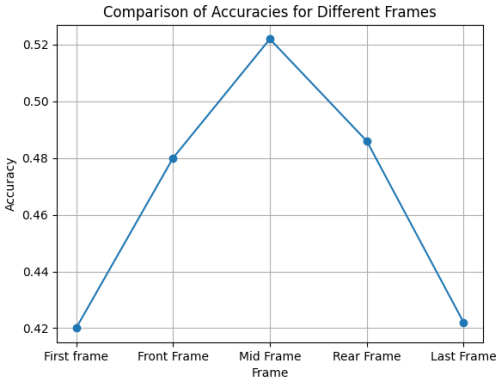
CHOICE 5: Systematically investigate how your results change when using other frames than the middle frame in your HMDB51 frame model

For the systematic investigation of variation within the frame classifier in the HMDB51 database, the procedure was as follows. Once the entire code of a functioning network capable of classifying actions represented by an input frame was developed, instead of using only the mid frame as required by the project baseline, a complete cycle of training and testing was conducted each time using a frame at a different position in the video, observing how this influences the obtained results.

The reasoning behind this research concerns the amount of information that each frame can represent. It is expected that, since the videos are very short, most of the characteristic information of each class (action) represented is likely contained towards the middle of it.

To verify this, it was decided to use 5 different frames, specifically, in addition to the middle one required by the baseline, corresponding to the total number of frames /2, the first frame, the front frame (total number of frames / 4), the rear frame (total number of frames / 4 * 3), and the last frame were also used.

The results obtained are as follows:



It can be observed how the increase in accuracy starts from the lowest value of 42%, corresponding to the results obtained using the first and last frames. This is due to the information contained within the frame. The first and last frames captured will indeed have fewer graphic information regarding the type of action performed in the video (since, considering, for example, the 'clap' class, the action might not have started yet

or already ended), leading the model to attempt to generalize information from these inputs but with not optimal results.

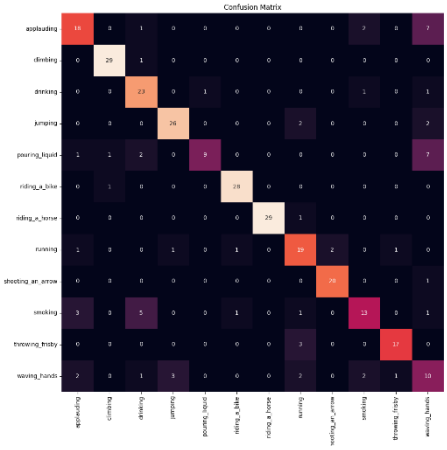
Proceeding with the front frame and the rear frame, the same reasoning as before is applied. It is observed, therefore, how these new frames within the video contain more specific information about the class they represent, confirming the expectation and logic used for the ongoing systematic research. This allows the model to generalize the input frames better, resulting in a 6% improvement for the front frame and a 6.22% improvement for the rear frame during testing, highlighting the correctness and necessity of a frame selection approach during data preprocessing to achieve performance improvement in the model.

Finally, as shown by the graph, it is confirmed that using the mid frame is the optimal choice for achieving optimal results during testing, reaching 52% accuracy.

It is therefore confirmed that the short duration of the videos translates into a greater concentration of visual information regarding the action performed towards the central frames of it, and this graph supports this correctness. The greater presence of characteristic components of each class allows the model to generalize these efficiently, resulting in a slight improvement compared to other intermediate frames (front frame, rear frame), but especially a significant improvement compared to the first and last frames

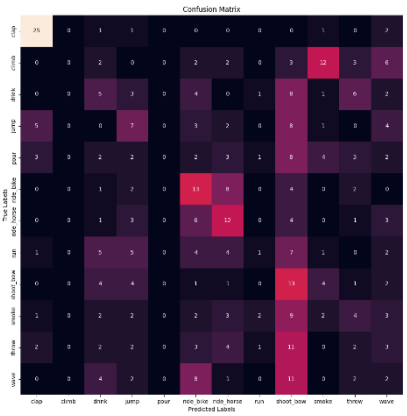
<u>CHOICE 8:</u> Present and discuss a confusion matrix for your (1) Stanford 40 Frames and (2) HMDB51 Optical flow models

Standford 40 Frames:



From the confusion matrix obtained after training the model and testing it on the StandFord 40 Frames dataset, it is immediately clear to observe an excellent level of accuracy. It is evident to see that the majority of classifications are done correctly, representing an accuracy of 80.59%. There is no clear error correlation, but it is possible to notice that the model tends to misclassify the 'waving_hands' class.

HMDB51 Optical Flow:



From the confusion matrix obtained after testing the model based on action recognition through optical flow, it is immediately observed that the performance is not optimal. Indeed, this represents a test accuracy of 22.78%. It is evident that the majority of classifications are incorrect, particularly noting how the model prioritizes the classification of the 'shoot_bow' class, followed by 'ride_bike' and 'ride_horse'.

<u>CHOICE 6:</u> Show class activations (e.g., using Grad-CAM) for two classes in the HMDB51 frame, optical flow and two-stream models

We make use of the GradCAM method implemented in the pytorch-grad-cam library where the activation of last convolutional layer for each model is recorded and displayed as a heat map on top of the original images for the action classes “clap” and “climb”. We see that for the HMDB51 frames model (first row), there is moderate, although not great focus on the center of the action in the image. For the optical flow model (second row), the focus on the movement is very minor, although for the “clap” class the focus seems to be better. Finally, the two-stream network’s (third row) activation shows much better focus on the action class “climb” and a slight improvement on “clap”.