First Person Action Recognition

First Author
Institution1
Institution1 address
firstauthor@i1.org

Second Author
Institution2
First line of institution2 address
secondauthor@i2.org

Abstract

In this paper we explore different methods for performing a first person action recognition task on the GTEA61 dataset. We start by implementing a structure based on Ego-RNN: a two stream architecture that works separately on the frames of the videos and on motion features extracted from optical flow. We then try to improve on top of this architecture by implementing a self-supervised task capable of working jointly on spatial and temporal features.
Finally, we implement a two-in-one stream architecture, embedding RGB and optical flow into a single stream.

1. Introduction

While the classification of third-person human actions has seen a lot of progress over time, the more niche task of detecting first-person actions can still be considered as a less explored field, and has seen some interesting developments in recent years. For the peculiarities that characterize this specific task, and because of the fact that we are using a fine-grained dataset to train our model, it becomes important to take into account both the objects and the motions characterizing the user performing the action.

In 2018, a paper by Sudhakaran *et al.* [3] proposed to address the first person recognition task by combining an architecture for the analysis of the frames with a temporal network, capable of learning from image features with spatial attention. This architecture, called Ego-RNN, is used as the starting point for our experiments.

One of the problems of this model is that frames and optical flow features are learned separately, merging the two branches only at the end of the model. Taking this into account, we expand on this architecture by implementing the self-supervised task proposed by Planamente *et al.* [1].
We implement this step by creating a self-supervised motion segmentation task, and feeding the backbone of the Ego-RNN network to it. As a result of the added MS task, the backbone will try to learn features regarding the object movements, that are supposed to be beneficial to the classification task.

After implementing the MS task both as originally conceived in the paper and as a regression problem, we take it a step further, by applying a motion condition and a motion modulation layer to our model, with the goal of using the features from this layers to modulate RGB features.

The code for all of the steps described above is made available at `https://github.com/paoloalb/FPAR`.

2. Implementation of Ego-RNN

A ResNet-34 network pre-trained on imagenet is implemented as the backbone for this model. This network is used together with a spatial attention layer in order to obtain image features with spatial attention. In this layer, we calculate the class activation map of the winning class, and we convert it to a probability map by applying softmax.

The features obtained allow us to encode both temporal and spatial dimensions at the same time, by using a convLSTM module. The output of the convLSTM is then fed to an average pooling layer and finally to a fully connected module for classification.
For the first stage we keep the weights of the pre-trained ResNet-34 locked, while training only the parts of the model without any trained weights (the convLSTM module and the final classifier). In this stage, as stated in [3], the network is trained for 200 epochs, with an initial learning rate of 10^{-3}, and a step down policy characterized by a decay factor of 0.1 after 25, 75 and 150 epochs. The best accuracy obtained with this parameters, with 16 frame videos, is 46.5%.

For the second stage we train also the spatial attention layer, together with the last layer of the ResNet and

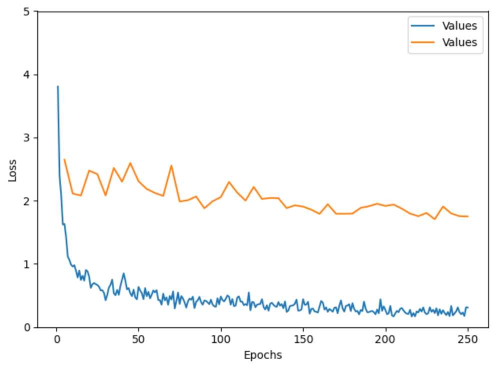
its fully connected layer. With this changes in place, we expect to see an increase in the network's performance, since in this stage our model will do a better job at learning spatial and temporal features.
For this phase, as done in the first stage, the hyperparameters used are the same of the original paper: 150 epochs of training with a learning rate of 10^{-4}, decayed after 25 and 75 epochs by a factor of 0.1. The increase in accuracy is consistent with the expected results, and in line with the results obtained by Sudhakaran *et al*. [3].

For comparison, the RGB network was also trained without the the spatial attention layer, obtaining 47.3% accuracy in the best of cases.

In combination with the above convLSTM-attention model, another method often adopted in literature ([3][2][4]) for action recognition tasks, is the use of stacked optical flow images, in order to train the temporal stream to recognise actions from motion. The flow images are taken from the provided dataset, they are arranged in stacks of 5 and are then fed to a temporal network based on a ResNet-34 model pretrained on ImageNet.
For the first run, the same parameters of the original paper were used, and the model was trained for 750 epochs, with a learning rate of 10^{-2}, decayed after 150, 300 and 500 epochs by a factor of 0.5. With this run, the best accuracy obtained on the validation set was slightly above 40%.

After training spatial and temporal features separately, the original paper proposes to concatenate the output of the two networks, adding a new fully connected layer to get the class scores, and performing a fine-tuning for 250 epochs. The best accuracy on the validation set, with 16 frame videos as inputs, was of about 63%.
However, when looking at the loss, we can see that a lot of overfitting is happening on the training set. We can speculate that, since the original model was trained on a larger quantity of data (25 frames instead of 7/16), the original Ego-RNN model had more features to learn with respect to ours. Because of that, our implementation of Ego-RNN takes less time before it begins to overfit.

Moreover, if we analyze the behaviour of the two models that compose the two-stream model, we can see that the temporal network has the same problem. This model, in fact, starts to overfit when less than half of the training is completed, and after about 300 epochs the accuracy on the validation doesn't change too much anymore, with just some random fluctuations. Since the original model saves just the model with the highest accuracy on the validation set, it may happen that, due to random fluctuations, the saved model is found in the last epochs, even if at that point the model is overfitting, and there are no relevant improvements.



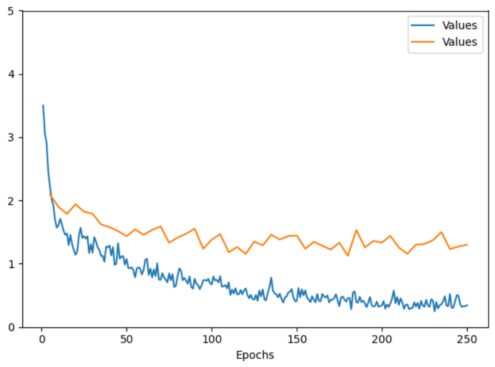


Figure 1. Loss of the two stream model before and after the overfitting reduction performed on the temporal network

Assuming that this behaviour of the temporal-warp flow could have repercussions on the whole two-stream model, we decided to train it again for just 300 epochs. This approach proved to be really beneficial, and was able to mitigate the overfitting effect on the two-stream model. After this changes were put in place, great improvements on the two-stream metrics were achieved, with the accuracy reaching 73.7% in the 16 frame case.
Complete results for the different phases are shown in the table below.

Configurations	7 Frames	16 frames
ConvLSTM	37.7%	46.5%
ConvLSTM-attention	55.2%	64.9%
Temporal-warp flow	42.2%	42.2%
Two-stream (joint train)	57%	73.7%

Table 1. Results of the experiments

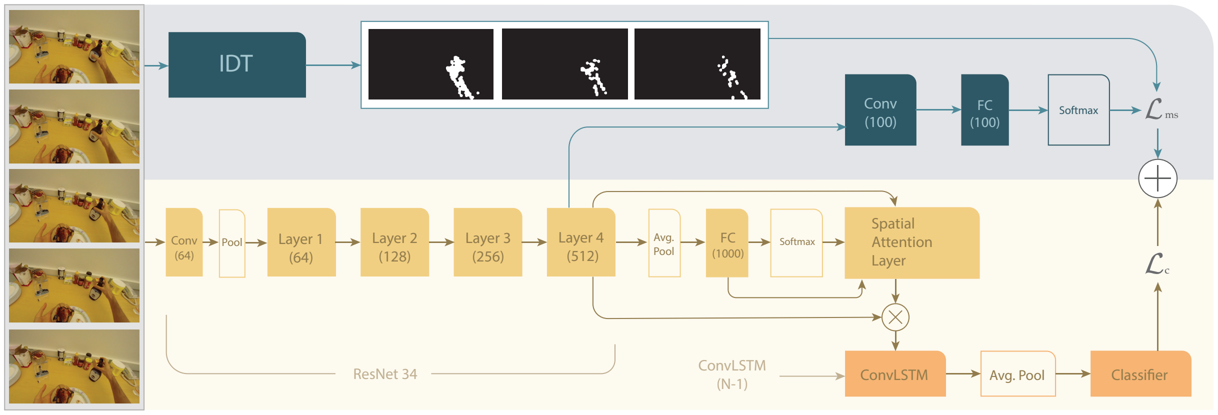


Figure 2. Self-supervised network architecture

3. Adding a self-supervised task

Until now, the motion and temporal networks implemeted above were trained separately. As suggested in [1], we try to implement a motion segmentation self-supervised task on top of the second stage of the Ego-RNN network, in order to learn jointly informations related to motion and frames. Another advantage of this model is that, differently from Ego-RNN, we can train it end-to-end in a single run.

As we can see from the schema above, the output of layer 4 is fed to the MS task, which is a simple branch composed of a convolutional block capable of reducing the channels from 512 to 100, and a fully connected layer of size 49, followed by a softmax. The output of the MS task is used to calculate the per-pixel cross-entropy between the computed 7x7 image and the ground truth, obtained from the motion maps already available in the original dataset.
These mmaps are downsampled to a 7x7 matrix, and each of the resulting 49 pixels is set to 0 or 1 depending on a given treshold. For the first run, the treshold was set to 0.5. However, setting it to 10^{-2} proved to be beneficial and increased the performance of the self-supervised task. For the calculation of the $\mathcal{L}_{ms}$ loss, we used the binary_cross_entropy function provided by PyTorch.

In the beginning, the results obtained by training this model were not comparable to the ones of the two-stream network implemented in the previous section, and even doing some hyperparameter optimization didn't provide great improvements. However, a turning point was reached by reducing the kernel size of the convolutional layer in the MS task from 7 to 1. This value is admittedly an unconventional choice, and the improvement obtained by using it can be explained by the fact that, with this value, the MS task doesn't lose information about the motion clues that we are trying to encode in the model with this self-supervised task.

After this changes were put in place, the best accuracy obtained on the validation set was 67.5%, obtained with 150 epochs of training, a batch size of 32, an initial learning rate of 10^{-4} and a step down policy characterized by a decay factor of 0.1 after 25 and 75 epochs.

When looking at these metrics (and more generally at the other accuracy values reported in this paper), an important thing to remember is that the validation was done on a small number of samples, since the provided dataset is relatively small for a deep learning task of this kind, expecially if compared with other similar datasets (*e.g.* Epic Kitchens).

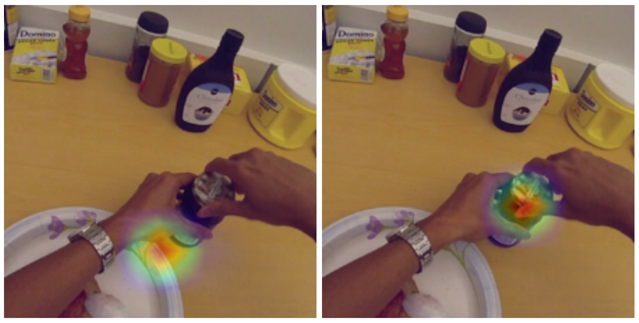


Figure 3. Class activation maps before and after the introduction of the MS task.

4. CAM visualization

Inside the spatial attention layer of Ego-RNN we take advantage of the average pooling of the ResNet34 model to compute class activation maps as proposed by Zhou *et al.* [6], in order to identify the regions of the image that were used to predict the winning class. To do it, we just need to map the predicted class score back to the previous convolutional layer, thus highlighting the class-specific regions that are being activated the most. While these maps have a useful purpose in the inner workings of the network, they can also be visualized, and used to find out the importance that the classificator gives to the different regions in the image.

We can visualize the effect of the self-supervised task by plotting the class activation maps before and after the implementation of the MS task. As we can see from the image above, before training the self-supervised task the model is focusing on the plate, while afterwards the network is capable of better identifying the jar.

The complete animations related to the displayed images, together with some other examples on self-made videos, are made available in the provided Github repository, together with the code.

5. Self supervised task as a regression problem

Since we are using a treshold to arbitrarly decide if a pixel of the motion maps is switched on or off, we could argue that we are losing some useful information during this process. A better idea would be to use a regression technique instead.
To achieve this goal, we add a sigmoid to the head of the MS task. We need also to remove the transformation based on a treshold that we implemented in the original self-supervised task, and we change the loss function for calculating the $\mathcal{L}_{ms}$.
We performed experiments with different functions for this loss, and the best accuracy was found with the use of the Kullback–Leibler divergence, that for two discrete probability distributions P and Q defined on the same probability space $\mathcal{X}$, is defined as:

$$D_{\mathrm{KL}}(P \parallel Q) = \sum_{x \in \mathcal{X}} P(x) \log \left(\frac{P(x)}{Q(x)} \right) \tag{1}$$

The best validation accuracies found with the different loss functions can be found in the table below.

Loss function	Accuracy
MSE	65.7%
Soft-margin	54.3%
Kullback–Leibler	70.1%

Table 2. Accuracy with different loss functions

6. Hyperparameter optimization

Some hyperparameter tweaking was done on epochs, batch size, step size (which is the number of epochs before the application of a step down policy on learning rate), type of optimizer and also the use of a relu activation function in the MS task. The results are shown in the table below.

Epochs	Batch size	Step size	Optimizer	Accuracy
150	32	[25, 75]	Adam	67.5%
150	32	[25, 75]	SGD	53.5%
150	16	[50, 100]	Adam	67.5%
200	32	[50, 100]	Adam	64.9%
150	16	[25, 75]	Adam	51.7%

Table 3. Hyperparameter optimization on self-supervised task

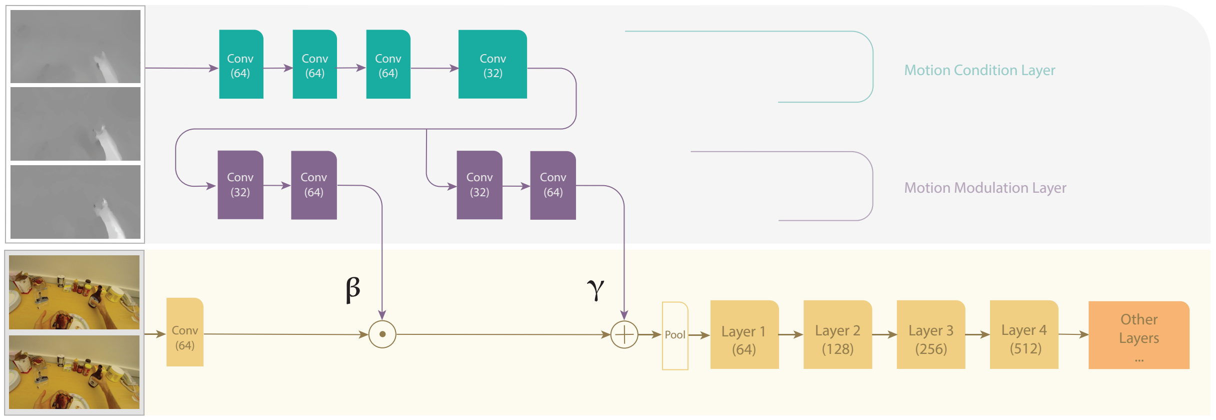


Figure 4. Two-in-one network architecture

7. Two-in-one stream action detection

In 2019, Zhao and Snoek [5] proposed a method to embed RGB and optical-flow into a single two-in-one stream, with the goal of performing spatio-temporal action detection with a model that can be trained end-to-end in a single run and can be easily implemented on top of a two-stream action detection model.
This allows us do conduct some more experiments with the optical flow images (that we stopped using in favour of mmaps computation), while reducing the higher computational costs required by the two separate streams that come as a disadvandage with the implementation of Ego-RNN.
This is achieved with the addition of two new layers: a motion condition and a motion modulation layer. The first layer applies a series of convolutions to the initial flow images, with the goal of generating some simple features from them. After this, the motion modulation layer learns a pair of affine transformations parameters β and γ through the use of two parallel series of convolutional layers. These parameters are then used in order to modulate the informations coming from the first convolutional layer of the RGB network, by applying to the RGB features F^{rgb} the following transformation:

$$\mathcal{M}^2(F^{rgb}) = \beta \odot F^{rgb} + \gamma \qquad (2)$$

With the first implementaiton of this model, we obtained an accuracy of 67.5%, reaching an accuracy similar to the one obtained with the first self-supervised task that we implemented before, and showing the effectiveness of this kind of approach.
In order to visualize better the mechanics involved in the implemented motion layers, we can plot the output of layer 1 before and after the application of the two motion layers, and also the inputs coming from β and γ.

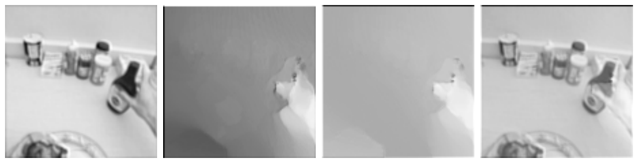


Figure 5. Gray-scale visualizations of pre motion layers, beta, gamma, post motion layers

8. Conclusion

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum

References

[1] M. Planamente, A. Bottino, and B. Caputo. Joint encoding of appearance and motion features with self-supervision for first person action recognition, 2020.

[2] K. Simonyan and A. Zisserman. Two-stream convolutional networks for action recognition in videos, 2014.

[3] S. Sudhakaran and O. Lanz. Attention is all we need: Nailing down object-centric attention for egocentric activity recognition, 2018.

[4] L. Wang, Y. Xiong, Z. Wang, Y. Qiao, D. Lin, X. Tang, and L. V. Gool. Temporal segment networks: Towards good practices for deep action recognition, 2016.

[5] J. Zhao and C. G. M. Snoek. Dance with flow: Two-in-one stream action detection. 2019.

[6] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba. Learning deep features for discriminative localization, 2015.