Computer Vision - Assignment 5

Odysseas Papakyriakou
o.papakyriakou@students.uu.nl (2716283)

May 1, 2023

1 Description of the CNN models

This section describes the four CNN models that are used for the different action recognition tasks. The results for the training, validation and test sets are displayed in the results section 2. Two datasets have been used: the Stanford 40, and the Human Motion Database (HMDB) 51. The Stanford 40 is an image dataset, consisting of 40 action classes, where each class is represented by 180-300 images, with a total of 9532 images. Stanford 40 also contains annotations for the bounding boxes of the human caring the action [Hom20], but these annotations were not used in this implementation. The HMDB51 is a video dataset with large variation, since the videos were sourced from the web. It has 51 classes, where each class is represented by 100-150 videos of a few seconds, with a total of 6849 videos [KJG+11]. The dataset comes with an annotation for the training and the test set, so it is implemented as such. For both datasets, 10% is used as a test set and 90% as a training set. A subset of 10% from the training set is used as a validation set, which is created with stratification to keep the proportion of each class equal across training and validation sets. For the purposes of these tasks, 12 overlapping classes between the datasets were used: *applauding*, *climbing*, *drinking*, *jumping*, *pouring*, *riding a bike*, *riding a horse*, *running*, *shooting*, *smoking*, *throwing*, and *waving*. The trained model weights are saved using Keras' checkpoint callbacks, allowing to save the model with the highest validation accuracy across all epochs. The weights can thus be easily loaded using Keras' *load_weights* function on the model instance. Both datasets have a more or less equal class distribution, so the metric that is used to evaluate the models' performance throughout is simple accuracy, defined by the ratio of the number of correctly classified images over the number of total images. All images and frames of the videos were resized to 72x72 using the $INTER\_AREA$ method with opencv's *resize* function.

1.1 Stanford frames: ResNet-like model

The model used for classifying the Stanford40 images resembles a ResNet18-like structure and was based on the ResNet paper published by Kaiming et al. [HZRS16] and on a tutorial implementation from Kaggle [Res]. The main advantage of residual networks is that they use residual connections to address the vanishing gradient problem that emerges when training deep networks. In short, when training deep networks with many layers, backpropagation computes the gradients of the loss function with respect to the weights, as shown in equation 1.

$$\frac{\partial L}{\partial W_l} = \frac{\partial h_l}{\partial h_{l-1}} \frac{\partial L}{\partial h_l} \quad \text{where } h_l \text{ is the output of layer } l \tag{1}$$

The gradients are computed recursively, as displayed in equation 2, showing that the gradient depends on the product of the weight matrices and the derivatives of the activation functions. If the derivatives are too small, then the gradients will become increasingly small, causing the network weights to update very slowly or not at all [HZRS16].

$$\frac{\partial h_l}{\partial h_{l-1}} = W_l \frac{\partial act(h_{l-1})}{\partial h_{l-1}} \quad \text{where } act(h_{l-1}) \text{ is the non-linearity applied to the output of layer } l-1 \tag{2}$$

Residual connections, or identity blocks, skip over some layers in the network, allowing the gradients to flow directly to the earlier layers [HZRS16]. Additionally, ResNet introduced skip connections, or convolution blocks, that combine the output of one layer with the input of a subsequent layer, thus improving information flow between layers and allowing the network to learn more complex features.

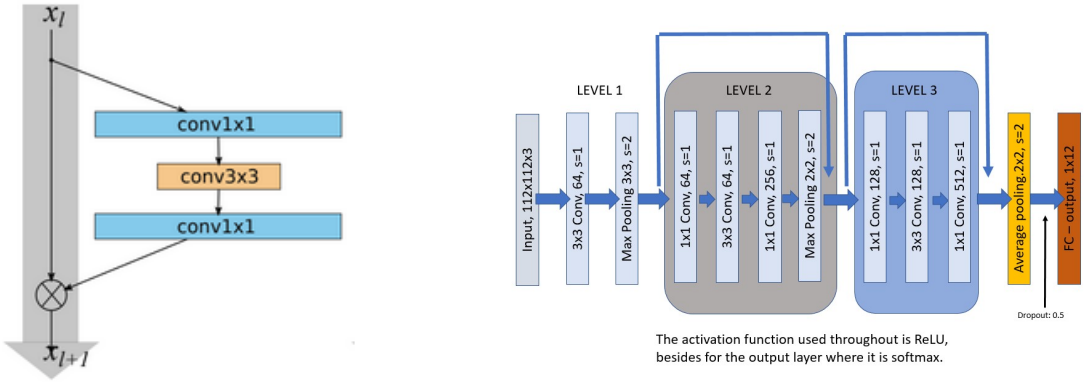
The difference between residual and skip connections is that skip connections have another convolution layer on the skip connection path, allowing to resize the input so that its dimensions match the output and can be added together. Figure 1a illustrates an example of a residual (identity) block and figure 1b shows an example of a skip (convolution) block.

(a) Identity block

(b) Convolution block

Figure 1: Residual (identity) and skip (convolution) blocks

For the current task I tried to adapt the ResNet50 model. ResNet50 introduced the 3-layer bottleneck block to replace the 2-layer blocks of the ResNet34, as described in the original article by Kaiming et al. [HZRS16]. This means that the residual connections now skip 3 layers, while there is also an additional 1x1 convolution layer in each block, thus reducing the number of parameters and decreasing the computational cost [HZRS16]. An illustration of a bottleneck block is shown in figure 2a, and a general description of the model's architecture is shown in figure 2b. There are several modifications to the structure and the hyperparameter values of the model as it was described in the original article [HZRS16]. Particularly, the fourth and fifth levels were removed completely, while the identity blocks that were included were only used once and not multiple times, as originally. This was because their addition resulted in high fluctuations of accuracy and loss without improving the model's performance. This is quite reasonable, since the dataset is not that large and including more layers doesn't help in learning more complex functions from the data. Moreover, the original architecture includes a fully connected layer of 1000 units, but this is completely omitted here, as it decreased the network's performance, probably due to the same reasons. Additionally, a dropout of 0.5 was introduced before the final output layer, while batch normalization was omitted completely. ReLU activation functions were kept as in the original structure after each convolution block. The optimization algorithm was Adam, since it has been shown to usually outperform other optimization algorithms, while its initial learning rate was set to 0.001 after some experimentation.



(a) Bottleneck block

(b) My ResNet architecture

Figure 2: Bottleneck and ResNet architecture

1.2 HMDB frames: Finetuning the ResNet model

After the model was trained on the Stanford data, it was finetuned on the middle frame of the HMDB51 data. The middle frame was selected on the assumption that it shows the action distinctively. For example, a video for drinking and smoking might start with two people, both having their hands down, thus looking very similar. Towards the middle of the videos the people drink and smoke which can now be more easily distinguished.

The model's weights were loaded and all layers besides the last two were kept frozen. This means that the model starts from the pre-trained weights, while only training the last two layers from the beginning. The learning rate now was decreased from 0.001 to 0.00001.

1.3 HMDB optical flow

Optical flow refers to the estimated motion calculations between two frames which are taken at times t and $t + \Delta t$, at every pixel position. As was explained in section 1.2, the middle frame is an intuitively reasonable choice to represent the video. Thus, 16 optical flow frames were calculated, centered around the middle frame. To avoid the frames looking identical, resulting in poor optical flow estimates, frames were selected with a step of two. There were a few too short videos, where it was not possible to take frames with steps of two, so in these cases the frames were consecutive.

Optical flow was calculated using opencv's function $calcOpticalFlowFarneback$, which implements Farnebäck's algorithm. In comparison to the Lucas Kanade method, which looks at corner points, the Farnebäck method considers all points and detects the pixel intensity changes between the two frames. Initially, the input images are converted to grayscale, and then the partial derivatives with respect to the spatial and temporal coordinates are computed using a Gaussian derivative filter. It then fits a polynomial function to the spatiotemporal gradients in a local neighborhood around each pixel, where the coefficients of the polynomial are used to estimate the motion of the pixel [Far03]. The Farnebäck algorithm was preferred because it is relatively faster and more robust to noise and occlusions compared to other algorithms.

The 16 optical flow frames that were calculated for every video ideally capture a distinctive movement for each class, thus allowing the model to improve the chance performance of 1/12. The data now are in 4 dimensions, with height and width having a value of 72, time being 16, and the number of channels being 2, namely the optical flow estimation in the x and y direction respectively. Therefore, a variant of a 3-dimensional convolutional neural network was developed, based on the paper by Tran et al. [TWT+18] and on a tensorflow tutorial [tut], describing a residual (2+1)D network. Although 2D convolutions can be applied to 4-dimensional data, the temporal ordering of the frames is ignored since the entire temporal information is collapsed in a single channel, thus preventing any temporal reasoning [TWT+18]. In contrast, with 3D convolutions the kernel is able to slide in all three dimensions, both spatial and the temporal one, thus preserving temporal information and propagating it through subsequent layers.

Building upon 3D CNNs, Tran et al. [TWT+18] propose a residual (2+1)D convolution block, where the 3D convolution is decomposed into two separate steps, a 2D spatial and a 1D temporal convolution for each layer. Moreover, the number of filters is determined in a way that the number of parameters remains the same as when using a full 3D convolution. According to the authors [TWT+18], a residual (2+1)D CNN has several advantages over a full 3D CNN. Specifically, although the number of parameters is not changed, such an architecture doubles the number of non-linearities in the network, since there is an additional ReLU between the 2D and 1D convolution in each block. This increases the complexity of the functions that can be represented, thus allowing the model to learn more complex features. Additionally, decomposing the 3D convolution into separate spatial and temporal components facilitates the optimization, since it results in lower training and testing loss [TWT+18].

For the current task, a ResNet-like model was implemented based on the aforementioned tensorflow tutorial [tut], where each convolution is a (2+1)D convolution. However, there were several modifications to the model to fit the task. Particularly, a max pooling operation with a kernel size of 2 was introduced after each block to downsample the input and bring it to the appropriate shape for the residual connections. Additionally, batch normalization was omitted completely, while a dropout of 0.35 was added between the two fully connected layers to reduce overfitting and improve generalization. The model's architecture is illustrated in figure 3.

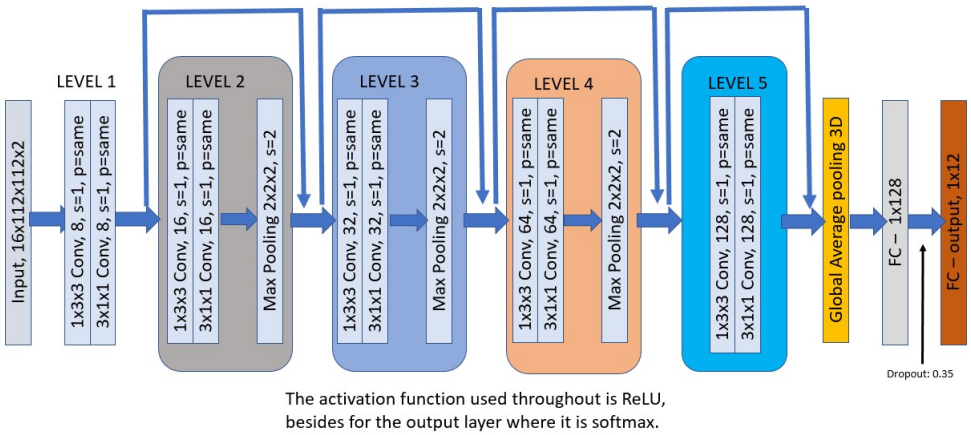


Figure 3: My residual (2+1)D architecture

1.4 HMDB: Two-stream network

Two-stream networks consist of two separate inputs to a model and one common output, and they have been shown to substantially improve performance in action recognition videos [SZ14]. The idea is that each input captures complementary information from the frames: the still RGB frames capture spatial information, while optical flow captures the motion between a series of frames. Initially, the two streams are trained separately, until they are fused and continue on a common path [SZ14].

In the current implementation, the two streams consist of the middle of the video, described in section 1.2, and the optical flow information, described in section 1.3. The output of the trained models' weights were loaded and fused by taking the average, as this method was shown to provide the best result according to the paper by Simonyan and Zisserman [SZ14]. All layers were frozen, and the two models were fused late, on the penultimate layer, from where they continued on a common path to a fully connected layer with ReLU activation, a dropout of 0.5, and the output layer with a softmax activation. The model was trained with a learning rate of 0.001 using the Adam optimizer.

Late fusion was decided based on its highest validation and test accuracy. Another option that seemed appealing, but did not beat the chosen option, was to fuse the models on the output of a convolution towards the end of the model, where the height and width dimension had a size of 14. Two variants were tried for the common path: 1) the fully connected path that was described above, and 2) a path with 2 (2+1)D convolutions, followed by average pooling, followed by the fully connected path. The second variant performed slightly better than the first, but as was mentioned, it did not beat late fusion.

2 Results

Table 1 shows the top-1 train, validation and test accuracy across all epochs, as well as the top-1 train loss, based on the minimum validation loss. It also shows how large the model weights are in MB.

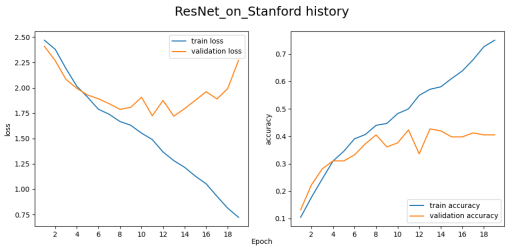
Dataset	Model	top-1 acc.: train/val./test	top-1 train loss	Model size (MB)
Stanford40	frames	0.48 / 0.423 / 0.418	1.491	10.3
HMDB51	frames	0.493 / 0.5 / 0.347	1.513	10.3
HMDB51	optical flow	0.21 / 0.226 / 0.264	2.174	5.2
HMDB51	two-stream	0.606 / 0.548 / 0.4	1.642	5.2

Table 1: Accuracies and loss per model

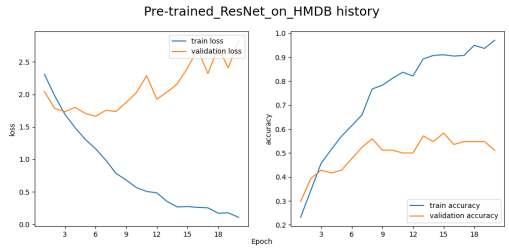
As becomes apparent, test performance on the HMDB51 videos is lower than the Stanford40 images, especially when considering the optical flow between frames or the middle frame individually. However,

when combining the middle frame with the optical flow information there is a substantial increase in test performance.

Figure 4a and figure 4b visualize train and validation loss and accuracy for the ResNet model described in sections 1.1 and 1.2. The models is first trained on Stanford images, and the weights are transferred for training the model anew on the middle frame of the HMDB51 videos, where only the last 3 layers are unfrozen.



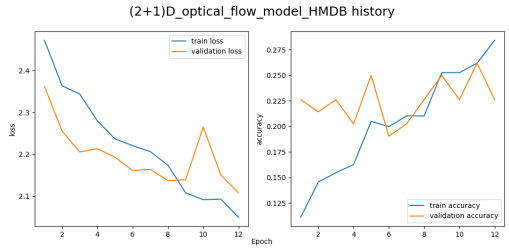
(a) ResNet model on Stanford frames



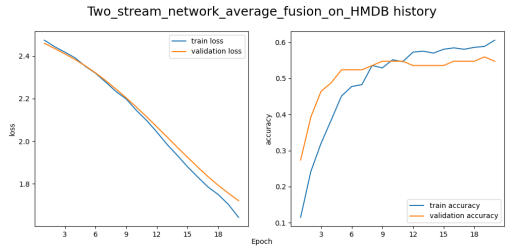
(b) Pretrained ResNet model on HMDB middle frames

Figure 4: ResNet model

Figure 5a visualizes train and validation loss and accuracy for the optical flow model on 16 frames of the HMDB51 videos. Figure 5b visualizes the same for the two-stream model, which combines input from the network on the middle frame of the HMDB51 videos and the optical flow network.



(a) Optical flow (2+1)D model



(b) Two stream model on HMDB videos

Figure 5: Optical flow and two-stream models

The weights for the trained model's can be found on the following links:
stanford40, finetuned_hmdb_middle_frame, hmdb_optical_flow, two-stream_model

3 Discussion

This section discusses the results presented in the previous section. Importantly, all models seem to quickly overfit, but especially the ResNet frame model, as becomes clear when noticing the big difference between the train and validation loss and accuracy. This is not very surprising since the implemented ResNet model described in section 1.1 is quite deep and complex for this small amount of data. It would thus be interesting to see how the model performs on a larger dataset.

Additionally, it seems that optical flow information alone is not as distinctive as the middle frame of the video, since accuracy is quite smaller. However, optical flow seems to provide a remarkable improvement when combined with the middle frame information, as the two-stream network performs around 5% better than the middle frame network.

4 Choice tasks

4.1 Choice task 3: Create a cyclical learning rate schedule

Cyclical learning rates were introduced by Leslie Smith in 2017 [Smi17] in order to eliminate the need to experimentally find the best values and schedule for the learning rate. Common learning rate schedules often monotonically decrease the learning rate during training, but this method lets the learning rate cyclically vary between some set boundaries [Smi17]. Importantly, it has been shown to be useful for adaptive optimizers as well. For example, Adam adjusts the learning rate on a per-parameter basis based on the first and second moments of the gradient [Rud16], while cyclical learning rate adjusts the learning rate based on a cycle. Thus, by combining the two, cyclical learning rate can escape local minima and plateaus, while an adaptive optimizer can reduce the effects of noisy gradients. Theoretically, this can result in faster convergence and better generalization performance [Smi17].

For demonstration, a cyclical learning rate schedule was implemented on the ResNet model using the Stanford40 data. The schedule made use of tensorflow's addons optimizers, implementing a triangular cyclical learning rate, where the minimum value was set to 0.01 and the maximum to 0.00001. An example visualization is shown in figure 6, where the learning rate cycles between 0.01 and 0.0001.

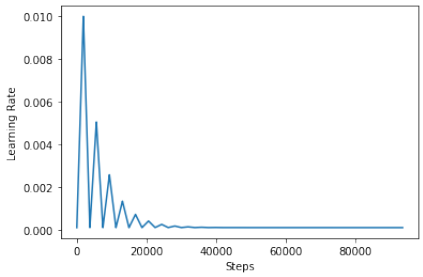


Figure 6: Example of a Cyclical Learning Rate schedule

The function used to scale up and down the learning rate is $\frac{1}{2^{x-1}}$, where x is the given cycle, and is named *triangular2* in the paper [Smi17]. The step size defines the duration of a single cycle and it was set to $2 * steps\_per\_epoch$, where $steps\_per\_epoch = \frac{train\_obs}{batch\_size}$. Contrary to what was expected, implementing a cyclical learning rate did not improve performance, as shown in table 2. Not only is the train and validation accuracy lower, but the model also shows a decreased testing accuracy from 0.418 to 0.336.

Dataset	CLR	top-1 acc.: train/val./test	top-1 train loss
Stanford40	no CLR	0.48 / 0.423 / 0.418	1.491
Stanford40	with CLR	0.399 / 0.306 / 0.336	1.739

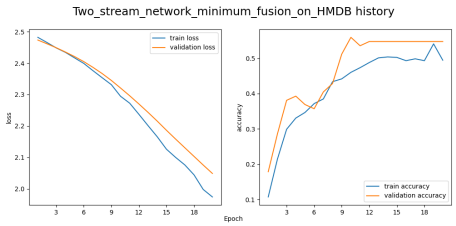
Table 2: Accuracies and loss for the Stanford40 data with and without CLR

4.2 Choice task 4: Use 1x1 convolutions to connect activations between the two branches

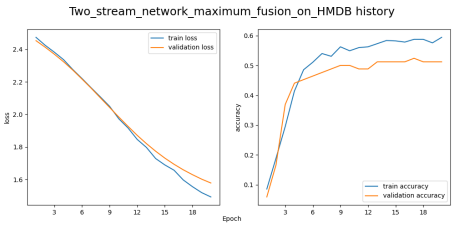
As described in section 1.4, fusion was also attempted after a convolution towards the end of the networks, where the height and width of the activation maps was 14. However, the optical flow model had 128 activation maps, while the ResNet model had 512 activation maps. To reduce the activation maps of the ResNet model from 512 to 128, a 1x1 convolution was performed on the output of that layer using *keras.layers.Reshape*, and then a convolution of 128 filters with a $1x1$ kernel.

4.3 Choice task 5: Experiment with different types of cooperation for the two networks

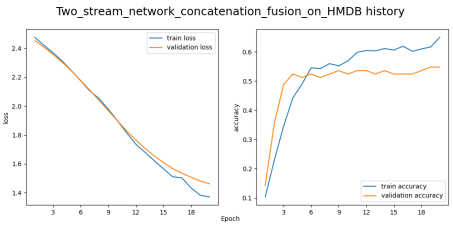
Figure 7 shows the training and validation accuracy and loss for different types of fusion. Particularly, the methods that were examined were minimum, maximum, concatenation, and average. The visualization for the average fusion is shown in section 1.4.



(a) Minimum fusion history



(b) Maximum fusion history



(c) Minimum fusion history

Figure 7: Cooperation types

As shown in table 3, the best performing method on the test set was the concatenation. However, from the remaining three, the average seemed to perform better than the minimum or the maximum.

Cooperation type	**top-1 acc.: train/val./test**	**top-1 train loss**
Minimum	0.495 / 0.548 / 0.383	1.974
Maximum	0.594 / 0.512 / 0.347	1.49
Concatenation	0.65 / 0.548 / 0.403	1.371
Average	0.606 / 0.548 / 0.4	1.642

Table 3: Results for different types of cooperation between the two networks

References

[Far03] Gunnar Farnebäck. Two-frame motion estimation based on polynomial expansion. volume 2749, pages 363–370, 2003.

[Hom20] Wendel Hom. Activity recognition with still images and video. 2020.

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[KJG+11] H. Kuehne, H. Jhuand, E. Garrote, T. Poggio, and T. Serre. Hmdb: A large video database for human motion recognition. *IEEE International Conference on Computer Vision*, pages 2556–2563, 2011.

[Res] Resnet and keras. https://www.kaggle.com/code/mishki/resnet-keras-code-from-scratch-train-on-gpu. Accessed: 2022-04-12.

[Rud16] Sebastian Ruder. An overview of gradient descent optimization algorithms. *ArXiv*, 2016.

[Smi17] Leslie Smith. Cyclical learning rates for training neural networks. pages 464–472, 2017.

[SZ14] Karen Simonyan and Andrew Zisserman. Two-stream convolutional networks for action recognition in videos. *Advances in Neural Information Processing Systems*, 1, 2014.

[tut] Video classification with a 3d convolutional neural network. https://www.tensorflow.org/tutorials/video/video_classification. Accessed: 2022-04-12.

[TWT+18] Du Tran, Heng Wang, Lorenzo Torresani, Jamie Ray, Yann LeCun, and Manohar Paluri. A closer look at spatiotemporal convolutions for action recognition. pages 6450–6459, 2018.