Two-Stream CNN Action Recognition

INFOMCV Assignment 5 Report

Marios Iacovou (1168533) – Christos Papageorgiou (9114343) (Group 74)

I. MODELS AND ARCHITECTURE CHOICES

The aim of this assignment is to classify actions in still images and videos with CNNs. To achieve this, we use 4 different models with 3 different architectures. The first and second model share the same architecture, with the second model being a fine-tuned iteration of the first model, and handle spatial cues using image inputs. The third model handles temporal cues using optical flow image stack inputs. The fourth model combines the abilities of the second and third model in a two-stream network architecture. To train these models, the image-based Stanford 40 Action dataset and the video-based Human Motion Database 51 (HMDB51) are used. Stanford 40 includes 40 action classes in total while HMDB51 includes 51, out of which 12 overlapping classes are chosen. After splitting the data for Stanford 40, we are left with 3516 images for the training and validation set with oversampling to account for the unequal number of images for every class and 304 images for the testing set. For HMDB51 we are left with just 840 videos for the training and validation set and 360 for the testing set. Due to the limited amount of data and computational resources at hand, we chose not to use overly-complex architectures to avoid overfitting and reduce the time it takes to train the models.

1. Frame CNN architecture:

We use GoogLeNet (Inception-v1) as implemented in PyTorch as our first architecture (Figure A.1 in Appendix section). We chose this architecture as GoogLeNet's use of inception modules, which employ parallel convolutional operations of different filter sizes within the same layer, make capturing features at multiple scales computationally efficient while maintaining high accuracy in image classification tasks. GoogLeNet also makes use of auxiliary classifiers at different depths to improve convergence and combat the vanishing gradient problem in training at the cost of additional parameters. The pre-trained implementation in PyTorch that we used as our base model, however, has them removed, which results in further parameter and complexity reductions. More specifically, the parameters of the model are reduced by half with this modification, while maintaining high accuracy due to pre-training on the ImageNet dataset. We use this pre-trained implementation with an adjusted dense output layer that matches our action classes to train our first model on Stanford 40 data and then fine-tune it on frames from the HMDB51 dataset and get our resulting second model.

2. Optical flow CNN architecture:

We use ResNet-18 as implemented in PyTorch as our second architecture (Figure A.2 in Appendix section). ResNet-18 is the simplest available version of the pre-trained ResNet family of networks available in PyTorch. We chose this architecture for its residual blocks that are designed to combat the vanishing gradient problem by employing skip connections, which retain and propagate important information more effectively through the layers. This helps in amplifying the motion cues from the image stacks attained from optical flow calculation, which in some cases can be minimal when the motion between frames is subtle. We use the pre-trained implementation of the network in PyTorch with an adjusted input layer that matches our RGB optical flow image stacks and an adjusted dense output layer that matches our action classes to train our third model on motion cues extracted from frame comparisons of videos in the HMDB51 dataset. In order to keep some information from the original pre-trained weights in the input layer, we average the weights across the RGB input channels of the original pre-trained model and replicate the results across every new input channel.

3. Combination into two-stream network:

Motivated by different examples of merging the spatial and temporal streams into a two-stream network in related work (Figure A.3 in Appendix section), we experimented with different fusion methods. After its final inception layer, our spatial GoogLeNet network ends with global average pooling on 1024x7x7 features which results in 1024x1x1 features that are flattened and passed to a dense layer after dropout is applied. The final classification is given by a softmax activation. Similarly, after its final residual block, our temporal ResNet-18 ends with a global average pooling on

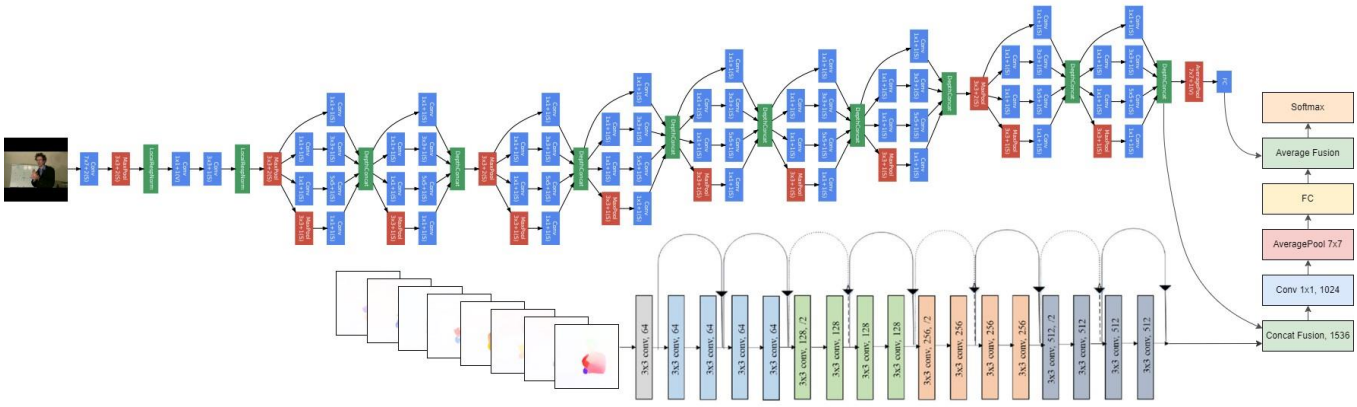


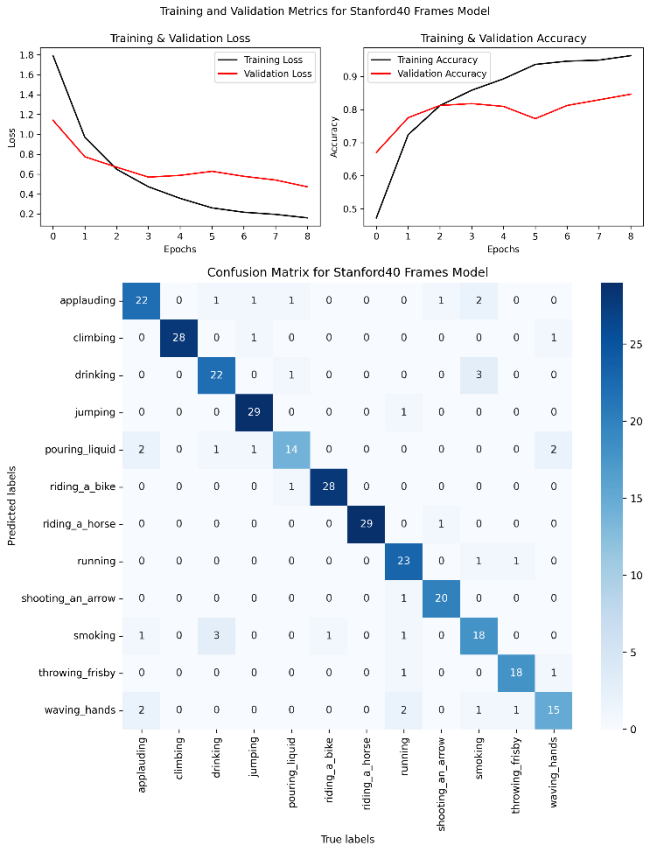
Figure 1: Two-stream network architecture combining spatial and temporal streams (ReLU and BN layers not shown)

512x7x7 features which results in 512x1x1 features that are flattened and passed to a dense layer, with the final classification given by a softmax activation. Our architecture of choice for the two-stream network follows the approach of merging the two streams before global average pooling is applied, but also keeping a separate stream where the spatial model continues to its trained final dense layer as before. On the merged stream we have a concatenated set of 1536x7x7 features that go through 1024 1x1 convolutions that effectively reduce their number to 1024x7x7 and select the most useful ones. The selected features go through a batch normalization layer, a ReLU activation, and global average pooling that results in 1024x1x1 features. These features are then flattened and passed to a dense layer after dropout is applied. We keep the dropout of the merged stream the same as the original dropout from the spatial model, which has a probability of 0.2. The final prediction of the model is given by a softmax activation after averaging the dense layer scores of the merged stream and the spatial stream, which showed better results than selecting the maximum dense layer scores. This approach allows us to utilize the spatial model with all its trained layers, while also combining its features with the temporal stream and getting new scores from a new dense layer. Our two-stream model with frames and optical flow image stacks from the HMDB51 dataset as inputs to its spatial and temporal streams can be seen in Figure 1 with just its batch normalization and ReLU activation layers omitted.

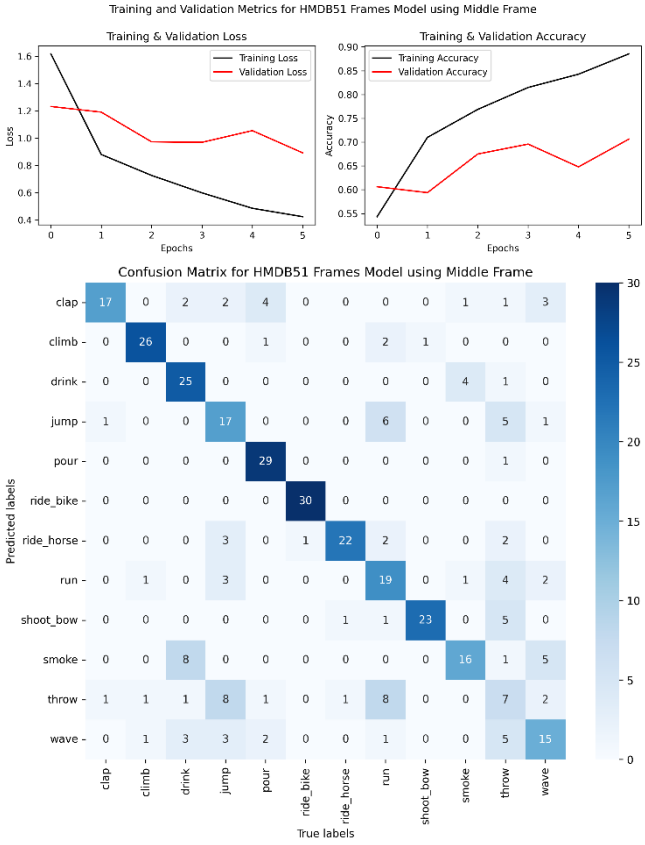
II. RESULTS

Below are the training and validation results of training our models, along with the resulting confusion matrices from evaluating them. The plots end at the best training epoch.

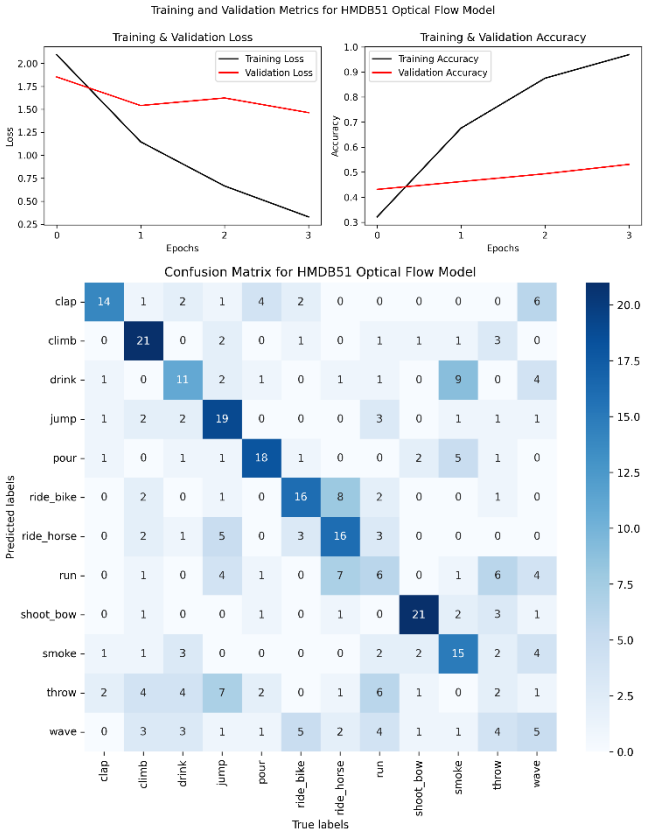
1. Stanford 40 Frames:



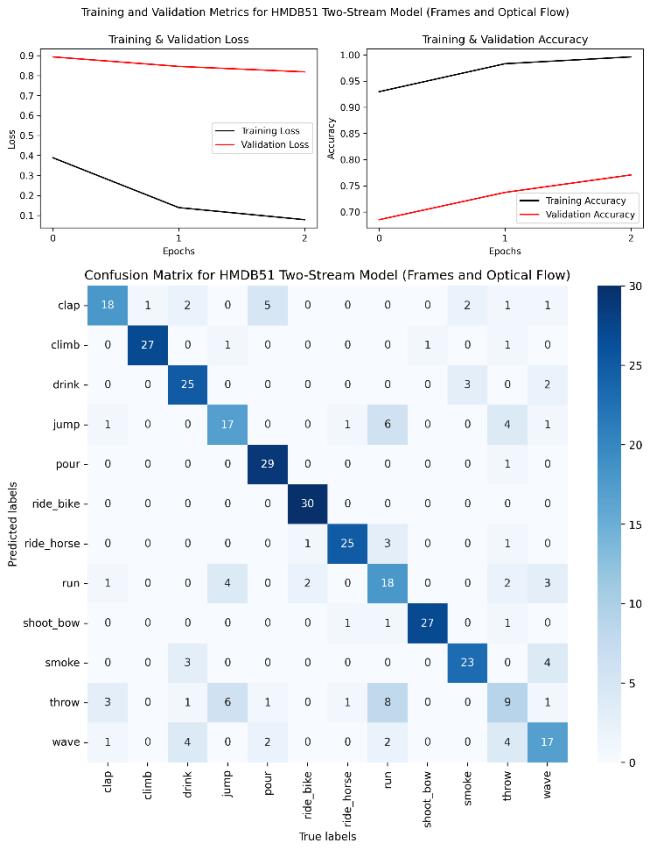
2. HMDB51 Frames:



3. HMDB51 Optical flow:



4. HMDB51 Two-stream:



Result table:

Dataset	Model specifications			
	Model	Top-1 accuracy (train / val / test)	Top-1 loss train	Model parameters
Stanford 40	Frames	0.963 / 0.847 / 0.869	0.161	5,612,204
HMDB51	Frames	0.885 / 0.706 / 0.664	0.424	5,612,204
HMDB51	Optical flow	0.969 / 0.531 / 0.435	0.331	11,248,524
HMDB51	Two-stream	0.996 / 0.771 / 0.714	0.079	18,441,784

III. DISCUSSION OF RESULTS

1. Stanford 40 Frames:

To train our first model, we combine the training and testing splits that come with the Stanford 40 dataset and isolate the 12 classes we need: “applauding”, “climbing”, “drinking”, “jumping”, “pouring liquid”, “riding a bike”, “riding a horse”, “running”, “shooting an arrow”, “smoking”, “throwing frisbee”, and “waving hands”. Then we create our own training and testing sets, with the testing set being 10% of the complete set. The testing set is generated using stratification to resemble the original distribution of the data. The training set is further split into a training set and a validation set, with the validation set being 10% of the training set before splitting. Since the distribution of the classes is uneven, our training and validation sets are then oversampled with random duplicates of their existing data to account for the imbalance. This is done after splitting them initially as to not have data spill between the different sets. Our resulting sets include 2868 images in the training set, 324 images in the validation set, and 304 images in the testing set.

The images in the dataset have varying sizes, so before feeding them to the model we pad them into square images using zero filling, and then resize them to 224x224 pixels. This maintains the original aspect ratios of the images while giving them the size that the first model requires. Further transformations are applied to the images during training to avoid overfitting, which is discussed in the Choice Tasks section. Finally, our images are normalized using the mean and standard deviation of ImageNet, which our first model is pre-trained on. The labels of the images are changed from strings to numbers from 0 to 11.

To train our first model with this data, we first imported a pre-trained GoogLeNet, as discussed in the first section of the report. We changed the output dense layer to 12 classes and trained the model with a small learning rate of 0.0001 on the new images. This learning rate showed to converge better than others we tested. We employ an adaptive learning rate scheme that results in the learning rate being halved every 5 epochs. We also experimented with a cyclical learning rate schedule which is discussed and compared against the decaying step schedule in the Choice Tasks section. For this model and for all the other models we keep the batch size as 32. Our model is set to train for 21 epochs but is reverted to the best epoch of its training to avoid overfitting. The best epoch is determined based on validation loss and accuracy during training, being updated only when both improve.

As can be seen from the training and validation graphs, our model reached its best state on epoch 9/21. The testing accuracy is similar to the validation accuracy and close to the training accuracy, which indicates that the model has trained without overfitting on our new data and can generalize well on unseen data. This also shows us that the oversampled training and validation sets with their augmentations allowed the model to generalize enough to classify the testing set that was not oversampled. Looking at the confusion matrix we can see that the model performed well on all different classes overall.

2. HMDB51 Frames:

To train our second model, we use the first train-test split that comes with the HMDB51 dataset and isolate the 12 classes that match the classes from the Stanford 40 dataset: “clap hands”, “climb”, “drink”, “jump”, “pour”, “ride bike”, “ride horse”, “run”, “shoot bow”, “smoke”, “throw”, and “wave”. Videos with tag 1 are used as training videos and videos with tag 2 are used as testing videos. We take 10% of the training videos as the validation split. No oversampling is required as the distribution of data is equal across classes. Our resulting sets include 756 videos in the training set, 84 videos in the validation set, and 360 videos in the testing set.

For our second model, we pick the middle frame from each video as input. We also experimented with 4 other frames

throughout the video which resulted in the middle frame giving the best results. The different results from using each frame to train the model are discussed in the Choice Tasks section. Before feeding the frames to the model we apply the same transformations to them as with the images from the Stanford 40 dataset and normalize them using the mean and standard deviation of ImageNet. The labels are again changed to 0-11 to be consistent with the labels of the first model.

The second model is a fine-tuned version of the first model, so it is only trained on 11 epochs and reverted back to its best state. The model trained best using the same learning rate and adaptive learning scheme as the first model. Additionally, fine-tuning the whole model without freezing any layers gave the best results.

As can be seen from the training and validation graphs, our model reached its best state on epoch 6/11. The training accuracy, validation accuracy, and testing accuracy are all lower than the first model's scores. We also see a higher loss on this model. We can see that this dataset results in some overfitting, despite the data augmentations that we had employed. This can be attributed to the limited amount of data the model has to adapt its weights or the complexity of the model for the task. We can also see that the validation accuracy differs from the testing accuracy by more points. When examining the different frames taken at different points of the video, we saw that for certain videos the action did not take place during the selected frame. This would result in the model trying to learn from things that do not resemble an action when facing these frames. This would also explain to some degree why the testing accuracy might differ from the validation accuracy, as more frames in the testing set might not display an action. We can also attribute this discrepancy to the validation set having only 7 images per class due to the limited amount of data, which makes it less indicative of the testing distribution of 30 images per class. Looking at the confusion matrix we can see that the model struggles most with the "throw" class, which can be an indication of lower quality videos for that class, or the actions presented in the videos being very similar to other classes. The rest of the class results are similar to the results from the first model.

3. HMDB51 Optical flow:

To train our third model, we use the same training, validation, and testing splits as the second model. Optical flow image stacks need to be extracted from the videos in the different sets to train and evaluate the model. Following the investigation into the results of training the second model using frames from different parts of the videos, we decided to select frames around the middle frame. We first split each video in 9 frames including the middle frame from its start up to its middle and keep the last 4 frames without including the middle frame. We then split the video into an additional 9 frames including the middle frame from its middle to its end and keep the first 4 frames not including the middle frame. In total this gives us 4 frames at intervals relative to the video length before the middle frame, the middle frame, and 4 frames at intervals after the middle frame. Using this approach, we aimed to select frames away from the start and end of the video and closer to the middle. After getting the 9 frames, we pair them in order with each other into 8 pairs. These 8 pairs will give us a calculation of the optical flow between them. To make this calculation we used the pre-trained RAFT model as implemented in PyTorch. RAFT uses a deep-learning approach to estimating optical flow which makes it more robust and able to capture more complex motion between frames compared to traditional computer vision methods. As RAFT iteratively improves its predictions, we pick the last prediction as our optical flow calculation and convert it to RGB. When calculating the optical flow on different frames, the frames are first square padded and resized to 224x224 pixels. Then they are rescaled to [-1, 1] for RAFT. Before being fed to the third model, calculated optical flow frames are first normalized on the means and standard deviation of ImageNet.

To train out third model with the optical flow image stacks, we first imported a pre-trained ResNet-18, as discussed in the first section of the report. We changed the input layer channels to 24 (8 stacked RGB images) and the output dense layer to 12 classes. As mentioned in the first section of the report, we average the weights of the original input layer of the pre-trained model across the RGB input channels. We then replicated the averaged results across every new input channel. This resulted in an improvement in our model accuracy as it allowed us to keep some information from the original pre-trained weights in the new modified input layer. The model is trained over 21 epochs as the first model with the same learning rate of 0.0001 and the same adaptive learning rate scheme.

As can be seen from the training and validation graphs, our model reached its best state on epoch 4/21. The training accuracy, validation accuracy, and testing accuracy are all lower than both the previous models. We can see that this model clearly overfits on the training data with a big difference between the testing accuracy and the training accuracy, as well as a big difference between the validation accuracy and the training accuracy. We experimented with adding an additional dropout layer before the final dense layer of the model to mitigate the overfitting, but the results were almost identical. Looking at the confusion matrix we can see that this model also struggles with the "throw" class as the second model, which can indicate that this class data is indeed hard to classify. Different confusions can be observed, such as "drink" and "smoke" which we expect to appear in videos together, "ride bike" and "ride horse" which look similar when looking at the posture of a person without any other spatial cues, "wave" and "clap" which feature similar hand movements, or "throw" and "run" that can look similar when looking at only specific movements in the motion.

The overfitting can be attributed to the lack of data transformations employed when extracting optical flow frames. Additionally, another issue can be the quantity and the quality of the data that we extract the optical flow frames from. As a considerable number of videos in the dataset are in low-resolution with a lot of camera shake present, we expect the optical flow calculations to be suboptimal. Additionally, the selection of frames we used may not be optimal for some optical flow calculations, with some selections only resulting in subtle motion cues and missing the motion change that we expect to see due to their interval.

This can be investigated by using different intervals when extracting the frames to compute the optical flow and re-training the model using different optical flow image stacks. Due to time constraints, we could not investigate the overfitting further and continue using the results presented.

4. HMDB51 Two-stream:

Our fourth model combines models 2 and 3, as described in the first section, and is fed the same data that models 2 and 3 would use. The data inputs undergo the appropriate transformations discussed for models 2 and 3 before being given to model 4.

The model is trained over 6 epochs with the same learning rate of 0.0001 as the previous models. The adaptive learning rate schedule is changed to scale the learning rate by a factor of 0.3 every 2 epochs. The models used in this combined architecture are already fine-tuned and should undergo minor changes during this training. Most of the training should be related to the new layer with 1x1 convolution that selects features from the concatenated features of spatial and temporal stream and the new dense layer after it.

As can be seen from the training and validation graphs, our model reached its best state on epoch 3/6. This combined model outperforms the fine-tuned spatial model and the temporal model, giving us the best overall results on the data from the HMDB51 dataset. We can see some overfitting when comparing the training accuracy to the validation and testing accuracy. The loss on this model is the lowest of all models, which is a positive sign about the confidence of the predictions the model gives. Overall, we can see that the testing accuracy is a bit lower than the validation accuracy, which as with previous models can be attributed to the quality and quantity of the selected frames. Looking at the confusion matrix we can see that the model performed well on all different classes, with “throw” being its weakness. We have observed this class be problematic across all models using data from the HMDB51 dataset.

IV. LINK TO MODEL WEIGHTS

Our trained models can be accessed from our GitHub repository:
https://github.com/miacov/Two-Stream-CNN-Action-Recognition/tree/main/models

V. CHOICE TASKS

1. Choice 1 and 2: Do data augmentation for Stanford 40 and HMDB51:

To tackle different types of invariances for our spatial models and avoid overfitting, we perform various transformations:

transforms.ColorJitter(brightness=0.4, contrast=0.4, saturation=0.4, hue=0.3): randomly changes the brightness, contrast, saturation, and hue of the image, helping the model to handle variations in lighting and color that occur in different environments

transforms.RandomGrayscale(p=0.1): converts images to grayscale, trains the model to recognize patterns without relying on color information

transforms.RandomAutocontrast(p=0.1): enhances the contrast of the image by scaling the pixel values to use the full color range, particularly effective for images that are either too dark or too light.

transforms.RandomAdjustSharpness(sharpness_factor=2, p=0.1): increases the sharpness of the image, which can make subtle details more noticeable

transforms.RandomHorizontalFlip(p=0.25): flips images horizontally, ensuring the model does not develop a bias towards the direction of the objects and can generalize across different orientations

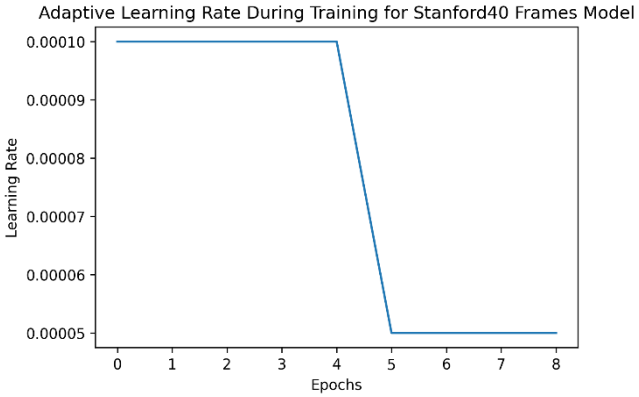
transforms.RandomPerspective(distortion_scale=0.3, p=0.1): modifies the perspective of images, simulating different angles and depths, helps in understanding objects from varied viewpoints

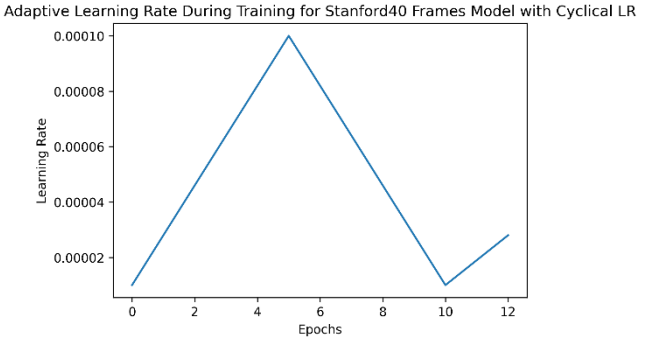
transforms.RandomRotation(degrees=15): randomly rotates the images, training the model to maintain accuracy even when images are taken from slightly tilted angles

2. Choice 3: Create a cyclical learning rate schedule:

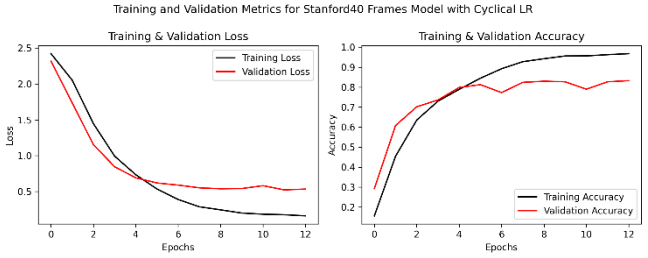
When training our first model, we used a decaying learning rate and a cyclical learning rate schedule. Our decaying learning rate schedule started with a learning rate of 0.0001 and halved it every 5 epochs. Our cyclical learning rate starts with a base learning rate of 0.00001 and sets the learning rate of 0.0001 as maximum learning rate for the first cycle. After this maximum learning rate is reached, the maximum for the next cycle is set to half of the previous maximum. A cycle takes 5 steps to get from the base learning rate to the maximum learning rate, and another 5 cycles to get from the maximum learning rate back to the base learning rate. This schedule can help the model make bigger moves in the search space using a bigger learning rate and then try to converge with a smaller learning rate, which can help in escaping local minima.

The results of training using the decaying learning rate were shown earlier in the report. We present the graphs of the learning rates through epochs up to their best-epoch results:





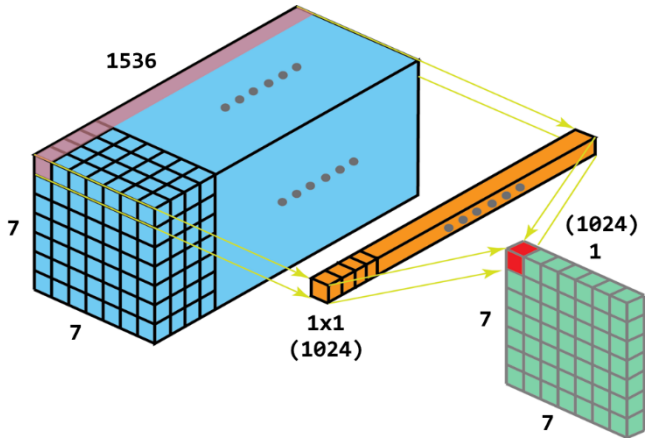
The training and validation graphs of training using the cyclical learning rate are the following:



The final testing accuracy achieved with the cyclical learning rate schedule was 0.859, which is very similar to the accuracy of 0.869 that was achieved using the decaying learning rate schedule. The test loss was 0.497 which is slightly higher than the loss of 0.419 we had with the decaying learning rate. Overall, we can see that the model using the cyclical learning rate schedule took some more epochs to reach its best-epoch results, but the results are very comparable to the ones achieved with the decaying learning rate schedule.

3. Choice 4: Use 1x1 convolutions to connect activations between the two branches:

In our two-stream network, we concatenate the 1024x7x7 features of our spatial model and the 512x7x7 features of temporal model into 1536x7x7 features. These features are then reduced to 1024x7x7 using 1024 1x1 convolutions, was discussed earlier in the report. The following figure is a visualization of these 1x1 convolutions:



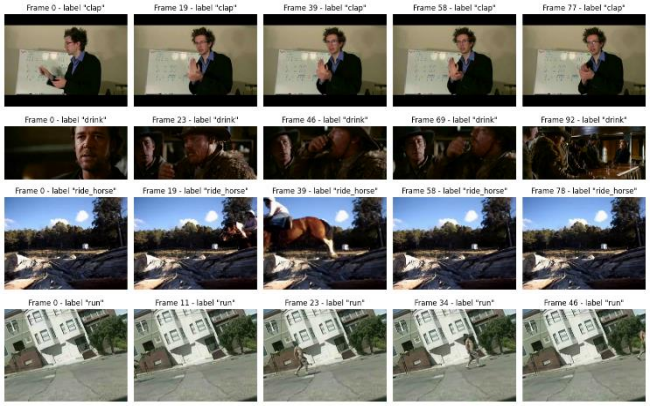
4. Choice 5: Systematically investigate how your results change when using other frames than the middle frame in your HMDB51 frame model:

From every video we sample the following frames: the frame in the middle, the first frame, the last frame, the frame in the center of the first and middle frame, and the frame in the center of the middle and the last frame. This will give as a curve of frame accuracy. The following are the training, validation, and testing accuracies for each of the frame selections:

Frame	Top-1 accuracy (train / val / test)
First	0.870 / 0.635 / 0.502
First-middle	0.915 / 0.731 / 0.593
Middle	0.885 / 0.706 / 0.664
Middle-last	0.883 / 0.756 / 0.622
Last	0.909 / 0.740 / 0.586

We can see from the best-epoch results that the training results are similar between the frames. However, when it comes to validation results and testing results, we can clearly see that the worst candidate out of the frames is the first frame. This can be due to videos having a completely different first frame at their start before switching to the expected motion video, which is the case with some of the videos in the dataset. Additionally, as the motion takes place over various frames, we expect that some frames are missing the motion, as discussed earlier in the report. The best overall results are achieved from frames around the middle frame, with the middle frame having the best testing results. This was considered when selecting parts of the video to take frames from when computing the optical flow image stacks.

Below are some examples of different frames in videos showcasing the phenomenon of the motion missing in some frame selections:



5. Choice 8: Present and discuss a confusion matrix for your (1) Stanford 40 Frames and (2) HMDB51 Optical flow models:

The confusion matrices for all our models were previously discussed and compared in the report. For each model, its weaknesses regarding certain classes and the reasons behind it were addressed.

VI. APPENDIX

Figure A.1: GoogLeNet architecture

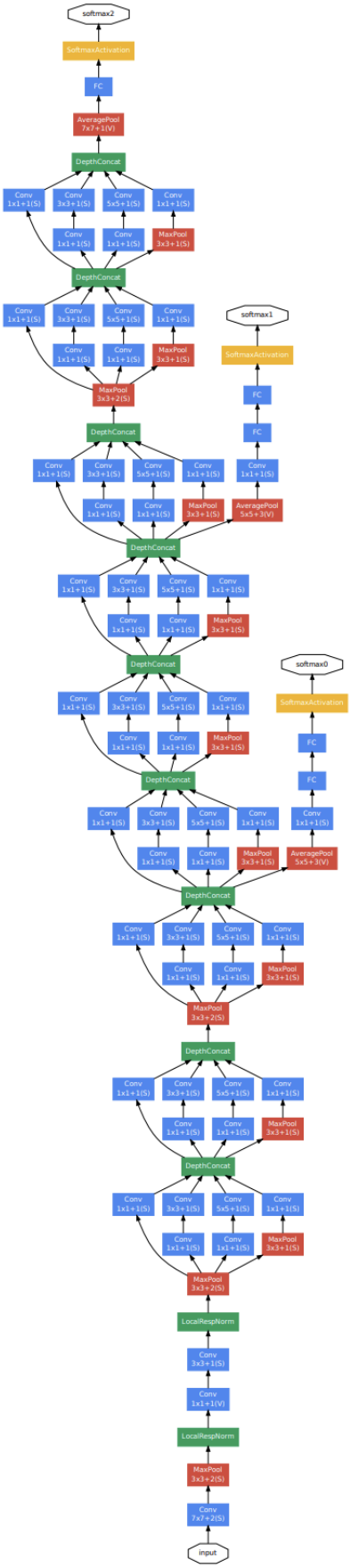


Figure A.2: ResNet-18 architecture

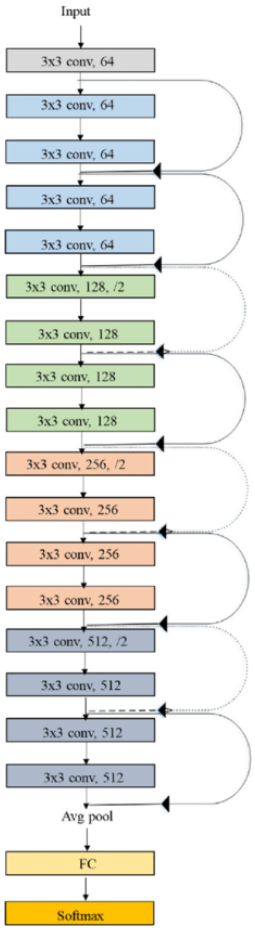


Figure A.3: Fusion examples for two-stream networks