HUMAN ACTION RECOGNITION IN DARK VEDIOS

Anonymous authors
Paper under double-blind review

ABSTRACT

In recent years, Human Action Recognition (HAR) in videos has garnered extensive research interest, particularly in applications such as surveillance and autonomous driving. However, the majority of these efforts have been concentrated on well-lit video data, resulting in State of the Art (SOTA) algorithms that often perform poorly in low-light or dark conditions. This paper focuses on addressing the challenge of enhancing action recognition performance in dark or low-light videos. We begin by comparing various frame sampling methods, including uniform sampling, MinMax-Sampling, and random sampling. Furthermore, we investigate image enhancement techniques such as Zero-Reference Deep Curve Estimation (Zero-DCE) and Contrast Limited Adaptive Histogram Equalization (CLAHE) to improve the contrast and brightness of the extracted image frames. Subsequently, the enhanced images are fed into a ResNet-18 (2+1)D neural network for feature extraction, followed by classification using a Support Vector Machine (SVM) and accuracy assessment. The paper also discusses the integration of SOTA methods from each stage into an end-to-end model. By combining these methodologies, this study provides new insights and practical solutions for effective action recognition in low-light conditions.

1 INTRODUCTION

Human action recognition (HAR) has emerged as a pivotal area in the domain of computer vision, boasting significant implications across various applications including human-computer interfaces (Meng et al., 2007), military operations, autonomous vehicles, and intelligent home systems (Zhang et al., 2007). Conventional State of the Art (SOTA) methods, encompassing various techniques, have predominantly concentrated on utilizing high-definition video data for both training and validation phases (Ghadiyaram et al., 2019) (Bello et al., 2021). These methods necessitate an environment with optimal lighting and clear visibility. However, such approaches exhibit suboptimal performance when applied to videos captured in nocturnal or low-light settings, highlighting a critical gap in their applicability under diverse lighting conditions.

In practical scenarios, the efficacy of computer vision transcends human capabilities, particularly in contexts where human visual perception is limited, such as nocturnal or dimly lit environments. This is notably evident in applications like nocturnal security surveillance and military rescue missions conducted under minimal lighting conditions. Achieving precise human action recognition in such low-light settings presents a formidable challenge, demanding extensive research and innovative solutions.

In recent times, the academic community has shown growing interest in refining the quality of low-light video content (Chen et al., 2019) Jiang & Zheng (2019). Certain conventional methodologies have pivoted towards enhancing the visibility in dimly lit footage. However, these methods occasionally compromise the integrity of the data, inadvertently impacting classification accuracy owing to suboptimal enhancement techniques. Concurrently, another research trajectory has been dedicated to devising effective algorithms capable of uniformly segmenting a diverse array of video lengths. This approach, though promising, faces significant challenges in accurately depicting the spatio-temporal attributes inherent in video data.

Tackling the challenge of extracting viable information from dimly-lit, unclear videos demands the selection of suitable enhancement techniques, coupled with the strategic segmentation of extracted

frames. By preprocessing and partitioning extraneous noise, image representation becomes more succinct, thereby streamlining the feature extraction process (Mourya et al., 2020).

Driven by this objective, our study delves into an effective approach for human action recognition in low-light videos. We employ the MinMax-Sampling Singh et al. (2022) method for sampling and the Zero-DCE (Guo et al., 2020) for image enhancement. The enhanced videos are then processed through a 3D-CNN-based ResNet-18 (2+1)D network (Tran et al., 2018), which is pivotal for feature extraction. Subsequently, these features are classified using an Support Vector Machine (SVM), yielding insightful inference results. To further refine our model, we integrate all these methods into a seamless end-to-end system. Here, the training dataset undergoes implicit sampling and enhancement before being inputted into the ResNet-18 (2+1)D network, allowing for a comprehensive training and inference validation.

2 Related Work

The recent landscape of human action recognition algorithms, as delineated in Kong & Fu (2022), categorizes them broadly into two distinct approaches: shallow techniques and deep architectures, as illustrated in Figure 1. The shallow approach encompasses methodologies like action representation, action classification, classifiers for human interaction, and classifiers tailored for RGB-D videos. These techniques primarily employ handcrafted feature engineering to capture local representations. On the other hand, deep architectures involve spatio-temporal networks, multi-stream networks, and hybrid networks, represented chiefly by 3D-CNNs and Two-stream methodologies.

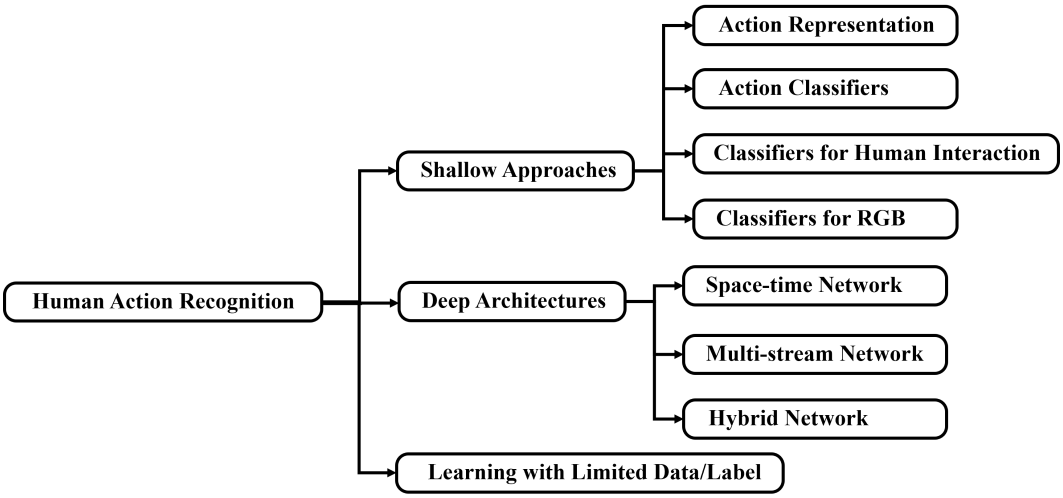


Figure 1: Classical Human Action Recognition Methods.

The mainstream approaches based on shallow architectures primarily consist of two types: holistic representation and local representation. In holistic representation, the method captures the overall motion information of HAR by capturing the global body structure and motion. Motion Energy Images (MEI) and Motion History Images (MHI), proposed by Bobick and Davis, exemplify this approach (Bobick & Davis, 2001). The main challenges of holistic representation are its sensitivity to noise, potential introduction of irrelevant information, the need for a static background, and difficulties in detecting detailed actions. On the other hand, local representation involves extracting local features from motion. Laptev's Spatio-Temporal Interest Points (STIP) method initially detects points of significant change, around which local descriptors are then constructed (Laptev, 2005). Despite its effective local feature extraction, this method sees limited practical application due to its complexity and computational intensity.

The field of human action recognition is advancing towards deeper architectures due to the powerful performance of deep learning, with several robust algorithms for action recognition already in existence. Simonyan and Zisserman (Simonyan & Zisserman, 2014) introduced a two-stream ConvNet architecture comprising spatial and temporal networks for action recognition. However, the

performance of this method was not entirely satisfactory, despite the authors further proposing the Two-Stream Inflated 3D ConvNet (I3D) (Carreira & Zisserman, 2017). To extract spatio-temporal features, attention shifted towards 3D structures, with the 3D-CNN model C3D pioneering this approach (Tran et al., 2015). This was followed by exciting developments like 3D-Resnet Hara et al. (2017) and 3D-Resnext. Moreover, the use of R(2 + 1)D (Tran et al., 2018) has been proposed for spatial and temporal convolution, enhancing the feature extraction for action recognition. The latest state-of-the-art (SOTA) methods revolve around the concept of Transformers, showcasing the prowess of self-attention techniques in this domain.

3 PROPOSED METHODOLOGY

The unique dark video human action recognition architecture proposed in this paper consists of four modules: **image frame sampling, image quality enhancement, feature extraction network and classification network.** The final composition of the end-to-end framework is shown in Figure 2. The framework takes the entire video as input and finally outputs the classification result.

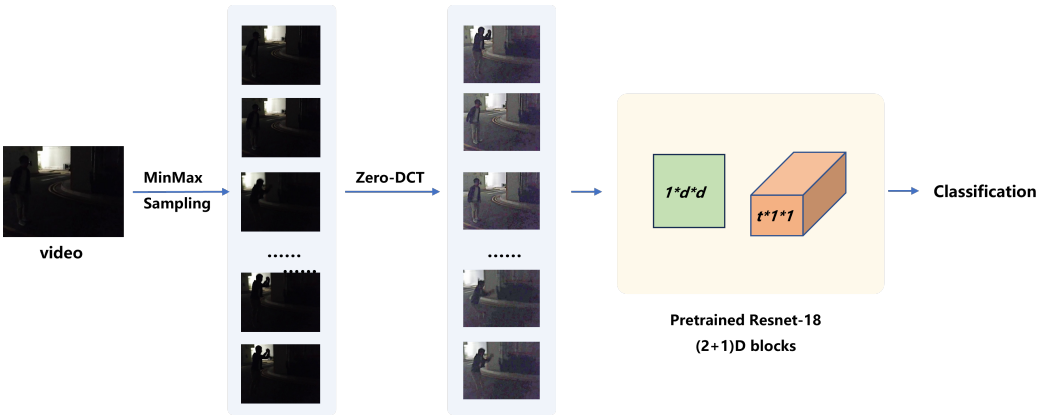


Figure 2: Framework of the proposed methodology.

3.1 IMAGE FRAMES SAMPLING

In human action recognition projects in low-light environments, choosing the correct video frame sampling method is crucial to ensure the effectiveness of model training. In this paper, three commonly used sampling methods are compared: max-min sampling, uniform sampling and random sampling, and the max-min sampling method is finally selected based on the sampling results.

3.1.1 UNIFORM SAMPLING

We first converted the raw input video into a sequence of images, and then applied uniform sampling to acquire one frame in every three image frames until 30 frames were collected (statistically, the number of frames in the training and validation videos varied between 55-100 frames). For less than 90 frames (30×3), a solid black image is used to fill the empty sample. This method is simple, straightforward, and ensures that frames are uniformly sampled from the overall length of the video to represent the action stream at constant intervals, capturing important spatio-temporal information. The frames obtained by uniform sampling are shown in Figure 3. **And for a more straightforward demonstration of the sampling result, the images are selected directly after enhancement.**

3.1.2 RANDOM SAMPLING

The random sampling method is to select frames randomly from the video. This method is simple, captures unexpected and important frames in the video, and is suitable for obtaining diversity and randomness in video content. However, it is very unfriendly to the continuity of actions and may ignore temporally important continuous actions, which is not taken in this paper.

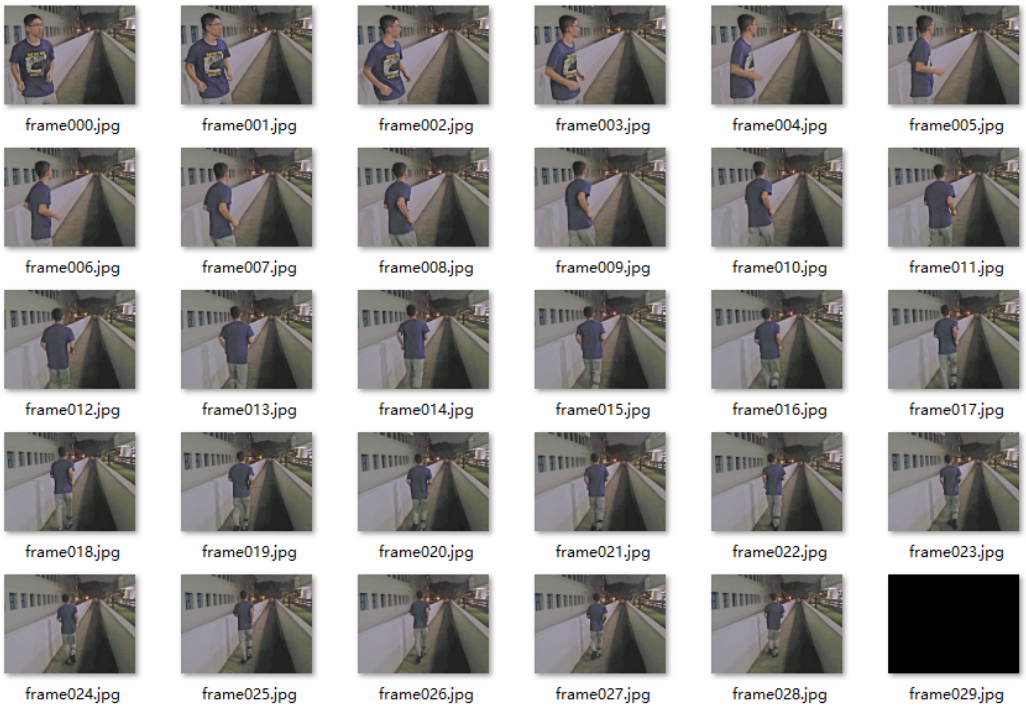


Figure 3: Uniformly sampled image frames from validation video 183.mp4.It can be seen that the last frame is supplemented by a blank frame as the total number of frames is less than 90.

3.1.3 MIN-MAX SAMPLING

Action videos are composed of action preparation, action start and action finish. Different videos have different lengths and distribution of action phases, and general sampling methods may lead to model overfitting. This project used the min-max sampling method to overcome this problem. Consider the input video $F = (F_1, ..., F_L)$, F_L denotes the F^{th} frame, L denotes the total number of frames. F_r denotes the number of frames required, and the maximum step length is calculated first:

$$S_{\text{max}} = \left\lfloor \frac{F}{F_r} \right\rfloor . \tag{1}$$

Thus the sampling step size S_{req} is as follows:

$$S_{\text{req}} = \text{Max}\left[\text{Min}\left[S_{\text{max}}, 4\right], 1\right], \tag{2}$$

The principle shows that the max-min sampling method considers the overall length of the video while selecting the frames. **This method ensures that frames are extracted from different parts of the video without over-concentrating on specific parts of the video.**The frames sampled for the training set video $Jump\_8\_1.mp4$ are shown in Figure 4.

In summary, max-min sampling is the most appropriate choice. Firstly, **it provides a more comprehensive coverage of the whole video**, including the onset, execution and end phases of the action, which is important to ensure that the model learns the different phases of the action when it is trained. Secondly, **max-min sampling omits the trial-and-error process of determining the sampling interval for uniform sampling and avoids uneven distribution of key actions**.

3.2 IMAGE QUALITY ENHANCEMENT

Since the overall video of the provided dataset has a mean value of approximately: [0.08149717 0.07725237 0.08080856] and a standard deviation of: [0.17944118 0.17953332 0.1681924], it is evident that the capture environment is very poor, so implementing image enhancement to improve the visual quality is a crucial step.

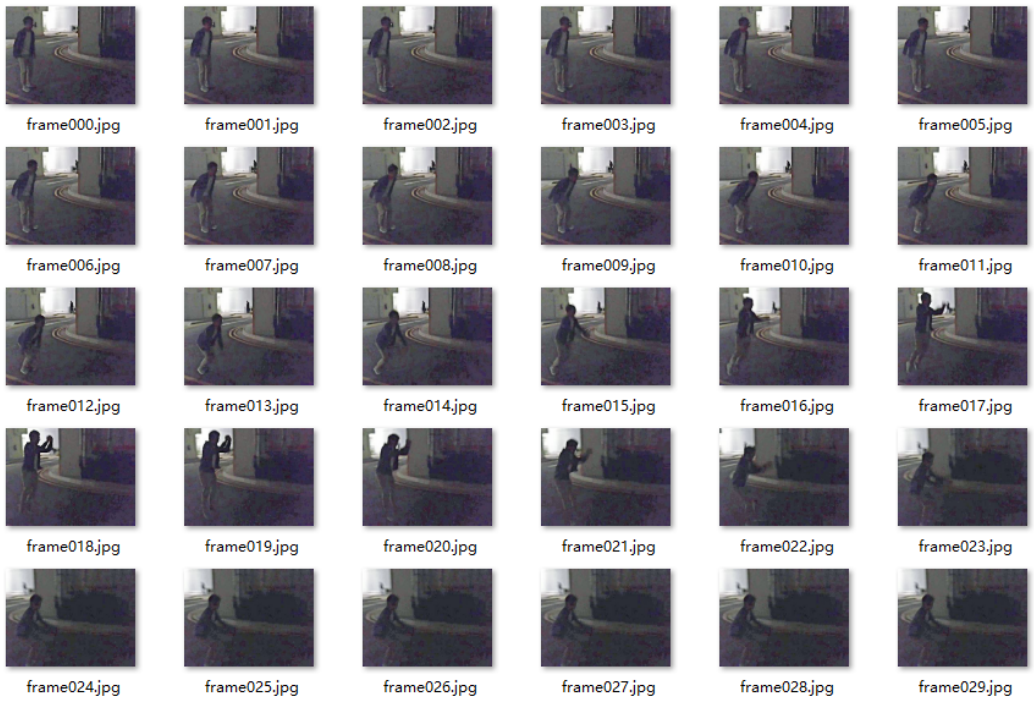


Figure 4: Min-max Sampling method sampled image frames from train video Jump_8_1.mp4.

3.2.1 CONTRAST LIMITED ADAPTIVE HISTOGRAM EQUALIZATION (CLAHE)

The histogram of an image can effectively represent its grayscale information, which is a comprehensive representation of brightness, contrast, color information, etc. Assuming that the input image is $f(x, y)$, the cumulative distribution function is as follows:

$$c(f) = \sum_{t=0}^{f} p_f(t) = \sum_{t=0}^{f} \frac{n_t}{n}, \quad f = 0, 1, \ldots, L \tag{3}$$

Then after applying histogram equalization (HE) to transform the input image in gray level, the output image is as follows:

$$g = T(f) = \text{round}\left(\frac{c(f) - c_{\text{min}}}{1 - c_{\text{min}}} L\right), \quad c(f) \geq c_{\text{min}} \tag{4}$$

When the process of intensity remapping is conducted across an image's various channels, leveraging its aggregate data, it can inadvertently amplify noise through excessive enhancement. Contrast Limited Adaptive Histogram Equalization (CLAHE) mitigates such noise amplification by partitioning the source image into discrete, non-overlapping contextual regions prior to applying HE. By computing individual histograms for each specified area, CLAHE imposes a cap on contrast enhancement, thereby curbing the prospective intensification of contrast to a permissible threshold. The image after enhancement using CLAHE is shown in Figure 5b.

3.2.2 ZERO-REFERENCE DEEP CURVE ESTIMATION (ZERO-DCE)

As a deep-structured image enhancement network paradigm, Zero-DCE can estimate the best-fit light enhancement curve based on the input image. It consists of three key components: the light enhancement curve, the DCE-Net, and the non-reference loss function.As a deep-structured image enhancement network paradigm, Zero-DCE can estimate the best-fit light enhancement curve based on the input image. It consists of three key components: the light enhancement curve, the DCE-Net, and the non-reference loss function.

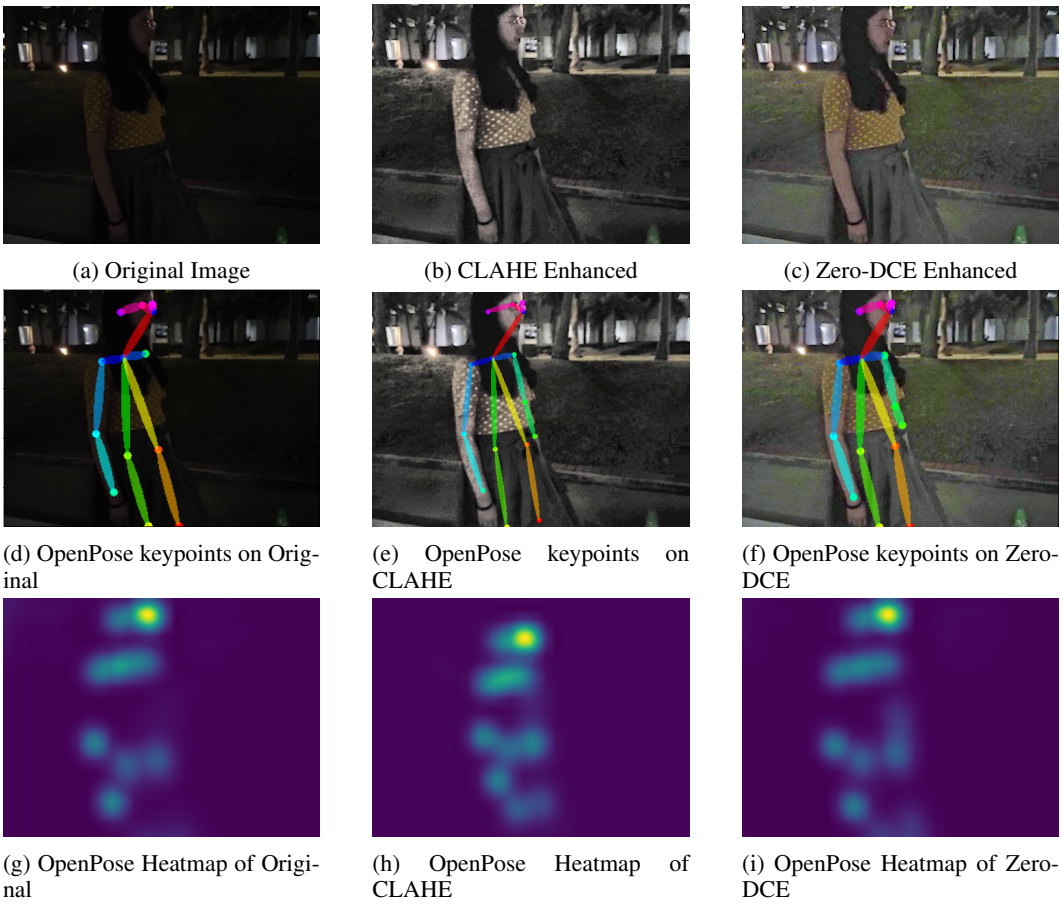
a) **Light-enhancement curve (LE-curve):** With the mapping technique, it automatically adjusts the curve parameters according to the input pixel values, while ensuring that the pixel values are regularized in the range of [0,1] to minimize information loss. What is more wonderful is that the monotonicity of the curve avoids the loss of details.

b) **DCE-Net:** Deep curve estimation networks are used to learn the mapping relationship from the input image to the optimal curve parameters, outputting a set of pixel-level parameter mappings representing higher-order curves. The network is known for its light weight.

c) **Non-reference loss functions:** The training of the model is performed by leveraging a linearly combined quintet of reference-based loss functions, specifically: spatial consistency loss (L_{spa}), exposure control loss (L_{exp}), color constancy loss (L_{col}), and illumination smoothness loss (L_{tvA}). This amalgamated loss function is crafted to maintain regional homogeneity and local luminance, while rectifying chromatic aberrations and ensuring the smooth transition of illumination across the image. The mathematical formulation is:

$$L_{\text{total}} = L_{\text{spa}} + L_{\text{exp}} + W_{\text{col}} L_{\text{col}} + W_{\text{tvA}} L_{\text{tvA}}, \tag{5}$$

where W_{col} and W_{tvA} are the weights of the losses.



(a) Original Image (b) CLAHE Enhanced (c) Zero-DCE Enhanced

(d) OpenPose keypoints on Original (e) OpenPose keypoints on CLAHE (f) OpenPose keypoints on Zero-DCE

(g) OpenPose Heatmap of Original (h) OpenPose Heatmap of CLAHE (i) OpenPose Heatmap of Zero-DCE

Figure 5: Comparison of Original, CLAHE, and Zero-DCE enhanced images from train video Stand_5_4's frame along with their corresponding Openpose keypoints and heatmaps.

In Figure 5c, the visual enhancement achieved by applying the Zero-DCE network to the original image is exhibited, **demonstrating a notable improvement in color saturation and brightness over the result augmented by CLAHE**. Furthermore, to assess the impact of varying image qualities on automated recognition systems, human skeletal keypoint maps and heatmaps are derived using the Openpose model[1], as depicted in Figure 5. **This comparative analysis highlights that image**

[1]https://github.com/Hzzone/pytorch-openpose

enhancement substantially influences the accuracy and clarity of the traced human keypoints and associated heatmaps. In summary, the image enhancement technology will significantly improve the contrast of the image while suppressing noise, and at the same time has an excellent correction effect on brightness and color, resulting in a good reproduction of the overall details.

3.3 FEATURE EXTRACTION NETWORK

The *ResNet* model, renowned for its superior residual learning framework, has recently become the preferred paradigm for 2D image classification. This paper employs the *ResNet-18 (2 + 1)D* architecture, a *ResNet*-style architecture that disentangles 3D convolutions into discrete 2D spatial and 1D temporal convolutions to extract clip-level features from videos. Given an input video with L frames of images with 3 channels and spatial dimensions $h \times w$, in a 3D CNN, the tensor dimensionality is four-dimensional, while in the *R (2 + 1)D* model, the original N_i 3D filters of dimensions $N_{(i-1)} \times t \times d \times d$ are substituted with M_i 2D spatial convolutional filters of dimensions $N_{(i-1)} \times 1 \times d \times d$ and N_i temporal convolutional filters of dimensions $M_i \times t \times 1 \times 1$. **Although this decomposition increases the volume of non-linear computations, it notably diminishes the training error.**

Due to the small size of the available dataset, this project directly adopts the use of pre-trained weights from Kinetics 400 for feature extraction from the Tran et al. (2018), where the output of the fully connected layer represents the size of the extracted features according to the network structure. The set of frames extracted from each video is fed into the model and the dimension of the extracted features is 400.

3.4 CLASSIFICATION NETWORK

In order to extensively verify the effect of image sampling method and image enhancement on the accuracy of extracted features, three classification methods are selected in this paper to observe whether the trends are consistent. The three classification methods are: **support vector machine**, **Naive Bayes** and adding a fully connected layer at the end of the R(2+1)D model to **output the classification results directly**.

3.4.1 CLASSIFIERS

In the realm of machine learning, various classifiers exhibit distinctive strengths contingent upon the nature and intricacy of the data. In this study, we employed *Support Vector Machine* (SVM) and *Naive Bayes* classifiers, grounded in their proven efficacy across a multitude of pattern recognition tasks.

Support Vector Machine (SVM): SVM is renowned for its robustness and effectiveness in high-dimensional spaces, which is quintessential for handling the complex feature sets derived from video frames in human action recognition tasks. It operates on the principle of structural risk minimization, striving to find a hyperplane that categorically separates classes while maximizing the margin. This aspect makes it particularly adept at managing datasets with a pronounced distinction between classes, as is often the case in action recognition where specific movements are indicative of particular actions.

Naive Bayes: On the contrary, Naive Bayes classifiers offer a probabilistic perspective, leveraging the Bayes theorem with the "naive" assumption of independence between features. Its allure lies in its simplicity and speed, especially when dealing with large datasets. Moreover, it has demonstrated considerable resilience against irrelevant features and is capable of handling missing data, making it a suitable candidate for preliminary analyses or when computational resources are limited. Despite its simplicity, Naive Bayes can perform remarkably well and is often used as a baseline for classification problems.

3.4.2 END-TO-END CLASSIFICATION

In addressing the constraints posed by the dataset at hand, the conventional methodology of feature extraction followed by independent classification has demonstrated suboptimal performance. In pursuit of a more efficacious approach, this study introduces an alternative strategy. We augment the

Resnet-18 (2+1)D architecture with an appended linear fully-connected layer corresponding to the number of action categories, which is six in this case. This addition facilitates end-to-end training directly utilizing the training dataset while simultaneously outputting the classification results. Concurrently, the validation dataset is assimilated into the model, serving as a benchmark to ascertain the efficacy of each iteration of training. The culmination of this model's architecture is illustratively presented in Figure 6, encapsulating the integrated approach adopted in this research.

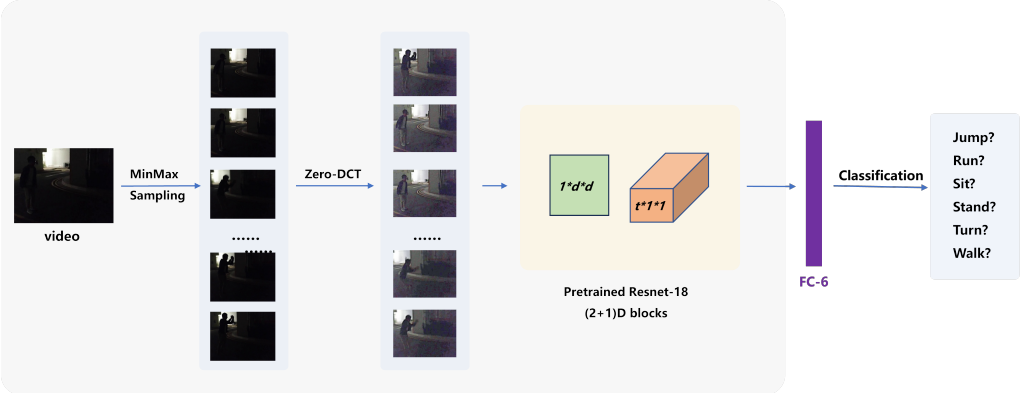


Figure 6: End-to-End Training Classification Architecture.

4 Experimental Result

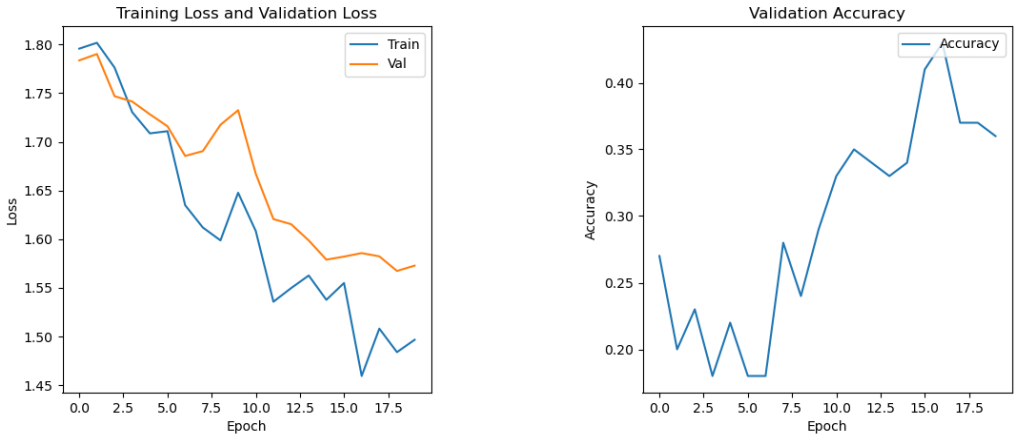
4.1 Experimental Details

All the proposed frameworks are implemented on the the Lenovo Legion y9000p with Intel Core i9-12900k and NVIDIA GEFORCE RTX3060. The training dataset used consists of six action categories, each containing 25 video data, and the corresponding validation dataset consists of 96 videos distributed within the corresponding categories. Experiments were completed using PyTorch. In this paper, we first compare the effects of different image enhancement methods on the classifier accuracy, and the results obtained are shown in Table 1.

Table 1: Comparison of Action Recognition Accuracy

Method	**Accuracy**	
	SVM	**Naïve Bayes**
Max-min Sampling + R(2+1)D	0.218	0.177
Max-min sampling + CLAHE + R(2+1)D	0.234	0.202
Max-min sampling + Zero-DCE + R(2+1)D	0.243	0.229
Max-min sampling + CLAHE + heatmap + R(2+1)D	0.217	0.199
Max-min sampling + Zero-DCE + heatmap + R(2+1)D	0.245	0.219

The input to the training experiments for the end-to-end model consisted of a series of frames of size 6 x 3 x 30 x 224 x 224, 30 frames of RGB three-channel images, with a batch size of 6. The cross-entropy loss function and the AdamW optimizer were used. The training experiments were conducted for 20 epochs and 40 epochs respectively.For the learning rate setting, a dynamic learning rate was adopted, where the learning rate is slowly increased at the beginning of the training to ensure that the error rate is reduced at the beginning of the training, and slowly reduced at about 7 epochs to ensure that the model accuracy is maintained. The training loss and verification loss during the 20 epochs training process are shown in Figure 7a, and the test accuracy is shown in Figure 7b. Meanwhile, Table 2 lists the accuracy of training 20 epochs and 40 epochs.



(a) Training Loss&Validation Loss

(b) Verification Accuracy

Figure 7: Training loss and verification accuracy for the model trained with 20 epochs. (a) Training Loss&Validation Loss. (b) Verification accuracy.

Table 2: End-to-End Training Accuracy Results

Method	**Top-1 (Max)**	**Top-3**
End-to-End Training with 20 epochs	44.38%	89.30%
End-to-End Training with 40 epochs	37.93%	87.52%

4.2 Result Discussion

4.2.1 Non-end-to-end model

The experimental summary reveals several critical insights into the model's performance. In the non-end-to-end models, the overall classification accuracy was notably low, with individual class accuracies struggling to reach 30%. The potential impacts of image enhancement, dataset quality, and the number of image frames on classification accuracy are discussed as follows:

1. Firstly, the improvement in feature extraction was marginal. Although most dark images were effectively enhanced visually, high-frequency noise persisting within the images, imperceptible to the human eye, may further impact feature extraction and, consequently, classifier performance.
2. Secondly, the dataset size, compared to public datasets with over 700 video materials (such as ARID), was insufficient in both quantity and diversity, leading to a propensity for overfitting and instability during training or testing.
3. Lastly, the number of image frames, as indicated in the appendix, one of the categories with lower accuracy scores, the "Walk" category, is often confused with the "Run" category due to their similarity. The spatiotemporal features of these two categories are very alike, making them challenging to distinguish in models lacking attention mechanisms. One solution is to increase the number of sampled frames per video to ensure the capture of every key action, which, while demanding more computational power, also imposes higher requirements on GPU memory.

4.2.2 End-to-end model

As depicted in Figure 7, the end-to-end training model exhibits a gradual decrease in training and validation losses, with a concurrent rise in validation accuracy, indicating that the model has not yet entered the overfitting phase. Table 2 further illustrates that both the validation and test sets

achieve a top-1 accuracy exceeding 35%, and the top-3 accuracy is approximately 90%. The superior performance of the end-to-end model over non-end-to-end models can be attributed to the enhanced accuracy of feature extraction achieved through the training of the feature extractor.

5 CONCLUSION

The field of Human Action Recognition (HAR) within video content is rapidly advancing, pushing the boundaries of machine learning and computer vision technologies into new territories such as surveillance and autonomous systems. This paper has taken a deep dive into the realm of HAR, specifically under the challenging conditions of low-light and nocturnal environments where conventional State of the Art (SOTA) methods tend to falter.

Our exploration began with an analytical comparison of various frame sampling techniques, including the traditional uniform sampling, the innovative MinMax-Sampling, and the stochastic random sampling. The objective was to discern the most effective strategy for capturing the quintessence of motion within the constrained luminance of dark video scenes.

In pursuit of refining the visual quality of these frames, we probed into the realms of image enhancement. Techniques such as Zero-Reference Deep Curve Estimation (Zero-DCE) and Contrast Limited Adaptive Histogram Equalization (CLAHE) were employed, each contributing significantly to the amplification of contrast and luminosity in the extracted frames. The result was a set of enriched images that served as a more robust foundation for feature extraction.

The crux of our feature extraction process was the ResNet-18 (2+1)D neural network. The network's adeptness at distilling pertinent features from the enhanced frames was paramount to the success of the subsequent classification phase. For the classification, we utilized a Support Vector Machine (SVM), a decision that was guided by the machine's proven proficiency in handling high-dimensional data. The accuracy of the SVM classifier was meticulously assessed, revealing a marked improvement over traditional methods when applied to low-light conditions.

Crucially, the integration of each discrete state-of-the-art methodology into a cohesive end-to-end model epitomized the innovative spirit of this research. By orchestrating the entire process from sampling to classification within a single streamlined model, we were able to achieve a synergy that not only optimized performance but also unveiled new avenues for HAR in less-than-ideal lighting scenarios.

The outcomes of this study underscore the potent potential of a holistic approach to HAR in dark videos. The top-1 and top-3 accuracies achieved in our experiments affirm the viability and superiority of the proposed methods. This research serves as a testament to the adaptability and resilience of machine learning techniques, even when confronted with the daunting task of deciphering human actions veiled in shadows.

ACKNOWLEDGMENTS

Thanks very much to Professor Xu Yuecong and Professor Jiang Xudong for giving us the treasure of knowledge. Through the exploration of experiments, I have grown a lot, and I hope to do more meaningful and contributing things in the future.

REFERENCES

Irwan Bello, William Fedus, Xianzhi Du, Ekin Dogus Cubuk, Aravind Srinivas, Tsung-Yi Lin, Jonathon Shlens, and Barret Zoph. Revisiting resnets: Improved training and scaling strategies. *Advances in Neural Information Processing Systems*, 34:22614–22627, 2021.

Aaron F. Bobick and James W. Davis. The recognition of human movement using temporal templates. *IEEE Transactions on pattern analysis and machine intelligence*, 23(3):257–267, 2001.

Joao Carreira and Andrew Zisserman. Quo vadis, action recognition? a new model and the kinetics dataset. In *proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 6299–6308, 2017.

Chen Chen, Qifeng Chen, Minh N Do, and Vladlen Koltun. Seeing motion in the dark. In *Proceedings of the IEEE/CVF International conference on computer vision*, pp. 3185–3194, 2019.

Deepti Ghadiyaram, Du Tran, and Dhruv Mahajan. Large-scale weakly-supervised pre-training for video action recognition. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 12046–12055, 2019.

Chunle Guo, Chongyi Li, Jichang Guo, Chen Change Loy, Junhui Hou, Sam Kwong, and Runmin Cong. Zero-reference deep curve estimation for low-light image enhancement. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 1780–1789, 2020.

Kensho Hara, Hirokatsu Kataoka, and Yutaka Satoh. Learning spatio-temporal features with 3d residual networks for action recognition. In *Proceedings of the IEEE international conference on computer vision workshops*, pp. 3154–3160, 2017.

Haiyang Jiang and Yinqiang Zheng. Learning to see moving objects in the dark. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 7324–7333, 2019.

Yu Kong and Yun Fu. Human action recognition and prediction: A survey. *International Journal of Computer Vision*, 130(5):1366–1401, 2022. doi: 10.1007/s11263-022-01594-9.

Ivan Laptev. On space-time interest points. *International journal of computer vision*, 64:107–123, 2005.

Hongying Meng, Nick Pears, and Chris Bailey. A human action recognition system for embedded computer vision application. In *2007 IEEE conference on computer vision and pattern recognition*, pp. 1–6. IEEE, 2007.

Komal Mourya, Sharda Patil, Tabasum Nadaf, Divya Voccaligara, Harsha Chari, and Shailendra Aswale. Techniques for learning to see in the dark: A survey. In *2020 International Conference on Emerging Trends in Information Technology and Engineering (ic-ETITE)*, pp. 1–7. IEEE, 2020.

Karen Simonyan and Andrew Zisserman. Two-stream convolutional networks for action recognition in videos. *Advances in neural information processing systems*, 27, 2014.

Himanshu Singh, Saurabh Suman, Badri Narayan Subudhi, Vinit Jakhetiya, and Ashish Ghosh. Action recognition in dark videos using spatio-temporal features and bidirectional encoder representations from transformers. *IEEE Transactions on Artificial Intelligence*, pp. 1–11, 2022. doi: 10.1109/tai.2022.3221912.

Du Tran, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. Learning spatiotemporal features with 3d convolutional networks. In *Proceedings of the IEEE international conference on computer vision*, pp. 4489–4497, 2015.

Du Tran, Heng Wang, Lorenzo Torresani, Jamie Ray, Yann LeCun, and Manohar Paluri. A closer look at spatiotemporal convolutions for action recognition. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 6450–6459, 2018.

Lun Zhang, Stan Z Li, Xiaotong Yuan, and Shiming Xiang. Real-time object classification in video surveillance based on appearance learning. In *2007 IEEE conference on computer vision and pattern recognition*, pp. 1–8. IEEE, 2007.

A Appendix

A.1 Python Code

```
import os
import cv2

def min_max_sampling(video_path, output_folder, required_frames):
    video = cv2.VideoCapture(video_path)
    total_frames = int(video.get(cv2.CAP_PROP_FRAME_COUNT))
    S_max = total_frames // required_frames

```

```
    if not os.path.exists(output_folder):
        os.makedirs(output_folder)

    for i in range(required_frames):
        frame_id = i * S_max
        video.set(cv2.CAP_PROP_POS_FRAMES, frame_id)
        success, frame = video.read()
        if success:
            cv2.imwrite(os.path.join(output_folder, f'frame_{i:03d}.jpg'), frame)
        else:
            print(f"Failed to read frame {frame_id} from {video_path}")

    video.release()

def process_videos(src_folder, dst_folder, required_frames):
    for root, dirs, files in os.walk(src_folder):
        for file in files:
            if file.endswith('.mp4'):
                video_path = os.path.join(root, file)
                relative_path = os.path.relpath(root, src_folder)
                output_path = os.path.join(dst_folder, relative_path, os.path.splitext(file)
                    [0])
                min_max_sampling(video_path, output_path, required_frames)

# Paths
train_video_folder = 'D:/Human-Action-Recognition-In-The-Dark-master/Human-Action-Recognition-
    In-The-Dark-master/EE6222_data/train'
val_video_folder = 'D:/Human-Action-Recognition-In-The-Dark-master/Human-Action-Recognition-In
    -The-Dark-master/EE6222_data/validate'
output_train_folder = 'D:/Human-Action-Recognition-In-The-Dark-master/Human-Action-Recognition
    -In-The-Dark-master/EE6222_data/train_img_2'
output_val_folder = 'D:/Human-Action-Recognition-In-The-Dark-master/Human-Action-Recognition-
    In-The-Dark-master/EE6222_data/validate_img_2'
required_frames = 30

# Process training and validation videos
process_videos(train_video_folder, output_train_folder, required_frames)
process_videos(val_video_folder, output_val_folder, required_frames)
```

```
import cv2
import numpy as np
import math
from scipy.ndimage import gaussian_filter
import matplotlib.pyplot as plt
import torch
import sys
sys.path.append('..')  #
from openpose_pytorch.src import util
from openpose_pytorch.src.model import bodypose_model

class Body(object):
    #
    def __init__(self, model_path):
        self.model = bodypose_model()
        if torch.cuda.is_available():
            self.model = self.model.cuda()
        model_dict = util.transfer(self.model, torch.load(model_path))
        self.model.load_state_dict(model_dict)
        self.model.eval()

    #
    def __call__(self, oriImg):
        # scale_search = [0.5, 1.0, 1.5, 2.0]
        scale_search = [0.5]
        boxsize = 368
        stride = 8
        padValue = 128
        thre1 = 0.1
        thre2 = 0.05
        multiplier = [x * boxsize / oriImg.shape[0] for x in scale_search]
        heatmap_avg = np.zeros((oriImg.shape[0], oriImg.shape[1], 19))
        paf_avg = np.zeros((oriImg.shape[0], oriImg.shape[1], 38))

        for m in range(len(multiplier)):
            scale = multiplier[m]
            imageToTest = cv2.resize(oriImg, (0, 0), fx=scale, fy=scale, interpolation=cv2.
                INTER_CUBIC)
            imageToTest_padded, pad = util.padRightDownCorner(imageToTest, stride, padValue)
            im = np.transpose(np.float32(imageToTest_padded[:, :, :, np.newaxis]), (3, 2, 0,
                1)) / 256 - 0.5
```

```
            im = np.ascontiguousarray(im)

            data = torch.from_numpy(im).float()
            if torch.cuda.is_available():
                data = data.cuda()
            # data = data.permute([2, 0, 1]).unsqueeze(0).float()
            with torch.no_grad():
                Mconv7_stage6_L1, Mconv7_stage6_L2 = self.model(data)
            Mconv7_stage6_L1 = Mconv7_stage6_L1.cpu().numpy()
            Mconv7_stage6_L2 = Mconv7_stage6_L2.cpu().numpy()

            # extract outputs, resize, and remove padding
            # heatmap = np.transpose(np.squeeze(net.blobs[output_blobs.keys()[1]].data), (1, 2, 0))  # output 1 is heatmaps
            heatmap = np.transpose(np.squeeze(Mconv7_stage6_L2), (1, 2, 0))  # output 1 is heatmaps
            heatmap = cv2.resize(heatmap, (0, 0), fx=stride, fy=stride, interpolation=cv2.INTER_CUBIC)
            heatmap = heatmap[:imageToTest_padded.shape[0] - pad[2], :imageToTest_padded.shape[1] - pad[3], :]
            heatmap = cv2.resize(heatmap, (oriImg.shape[1], oriImg.shape[0]), interpolation=cv2.INTER_CUBIC)

            # paf = np.transpose(np.squeeze(net.blobs[output_blobs.keys()[0]].data), (1, 2, 0))  # output 0 is PAFs
            paf = np.transpose(np.squeeze(Mconv7_stage6_L1), (1, 2, 0))  # output 0 is PAFs
            paf = cv2.resize(paf, (0, 0), fx=stride, fy=stride, interpolation=cv2.INTER_CUBIC)
            paf = paf[:imageToTest_padded.shape[0] - pad[2], :imageToTest_padded.shape[1] - pad[3], :]
            paf = cv2.resize(paf, (oriImg.shape[1], oriImg.shape[0]), interpolation=cv2.INTER_CUBIC)

            heatmap_avg += heatmap_avg + heatmap / len(multiplier)
            paf_avg += + paf / len(multiplier)

        all_peaks = []
        peak_counter = 0

        for part in range(18):
            map_ori = heatmap_avg[:, :, part]
            one_heatmap = gaussian_filter(map_ori, sigma=3)

            map_left = np.zeros(one_heatmap.shape)
            map_left[1:, :] = one_heatmap[:-1, :]
            map_right = np.zeros(one_heatmap.shape)
            map_right[:-1, :] = one_heatmap[1:, :]
            map_up = np.zeros(one_heatmap.shape)
            map_up[:, 1:] = one_heatmap[:, :-1]
            map_down = np.zeros(one_heatmap.shape)
            map_down[:, :-1] = one_heatmap[:, 1:]

            peaks_binary = np.logical_and.reduce(
                (one_heatmap >= map_left, one_heatmap >= map_right, one_heatmap >= map_up, one_heatmap >= map_down, one_heatmap > thre1))
            peaks = list(zip(np.nonzero(peaks_binary)[1], np.nonzero(peaks_binary)[0]))  # note reverse
            peaks_with_score = [x + (map_ori[x[1], x[0]],) for x in peaks]
            peak_id = range(peak_counter, peak_counter + len(peaks))
            peaks_with_score_and_id = [peaks_with_score[i] + (peak_id[i],) for i in range(len(peak_id))]

            all_peaks.append(peaks_with_score_and_id)
            peak_counter += len(peaks)

        # find connection in the specified sequence, center 29 is in the position 15
        limbSeq = [[2, 3], [2, 6], [3, 4], [4, 5], [6, 7], [7, 8], [2, 9], [9, 10], \
                   [10, 11], [2, 12], [12, 13], [13, 14], [2, 1], [1, 15], [15, 17], \
                   [1, 16], [16, 18], [3, 17], [6, 18]]
        # the middle joints heatmap correpondence
        mapIdx = [[31, 32], [39, 40], [33, 34], [35, 36], [41, 42], [43, 44], [19, 20], [21, 22], \
                  [23, 24], [25, 26], [27, 28], [29, 30], [47, 48], [49, 50], [53, 54], [51, 52], \
                  [55, 56], [37, 38], [45, 46]]

        connection_all = []
        special_k = []
        mid_num = 10

        for k in range(len(mapIdx)):
            score_mid = paf_avg[:, :, [x - 19 for x in mapIdx[k]]]
```

```
            candA = all_peaks[limbSeq[k][0] - 1]
            candB = all_peaks[limbSeq[k][1] - 1]
            nA = len(candA)
            nB = len(candB)
            indexA, indexB = limbSeq[k]
            if (nA != 0 and nB != 0):
                connection_candidate = []
                for i in range(nA):
                    for j in range(nB):
                        vec = np.subtract(candB[j][:2], candA[i][:2])
                        norm = math.sqrt(vec[0] * vec[0] + vec[1] * vec[1])
                        norm = max(0.001, norm)
                        vec = np.divide(vec, norm)

                        startend = list(zip(np.linspace(candA[i][0], candB[j][0], num=mid_num)
                            , \
                                            np.linspace(candA[i][1], candB[j][1], num=mid_num)
                                                ))

                        vec_x = np.array([score_mid[int(round(startend[I][1])), int(round(
                            startend[I][0])), 0] \
                                          for I in range(len(startend))])
                        vec_y = np.array([score_mid[int(round(startend[I][1])), int(round(
                            startend[I][0])), 1] \
                                          for I in range(len(startend))])

                        score_midpts = np.multiply(vec_x, vec[0]) + np.multiply(vec_y, vec[1])
                        score_with_dist_prior = sum(score_midpts) / len(score_midpts) + min(
                            0.5 * oriImg.shape[0] / norm - 1, 0)
                        criterion1 = len(np.nonzero(score_midpts > thre2)[0]) > 0.8 * len(
                            score_midpts)
                        criterion2 = score_with_dist_prior > 0
                        if criterion1 and criterion2:
                            connection_candidate.append(
                                [i, j, score_with_dist_prior, score_with_dist_prior + candA[i
                                    ][2] + candB[j][2]])

                connection_candidate = sorted(connection_candidate, key=lambda x: x[2],
                    reverse=True)
                connection = np.zeros((0, 5))
                for c in range(len(connection_candidate)):
                    i, j, s = connection_candidate[c][0:3]
                    if (i not in connection[:, 3] and j not in connection[:, 4]):
                        connection = np.vstack([connection, [candA[i][3], candB[j][3], s, i, j
                            ]])
                        if (len(connection) >= min(nA, nB)):
                            break

                connection_all.append(connection)
            else:
                special_k.append(k)
                connection_all.append([])

        # last number in each row is the total parts number of that person
        # the second last number in each row is the score of the overall configuration
        subset = -1 * np.ones((0, 20))
        candidate = np.array([item for sublist in all_peaks for item in sublist])

        for k in range(len(mapIdx)):
            if k not in special_k:
                partAs = connection_all[k][:, 0]
                partBs = connection_all[k][:, 1]
                indexA, indexB = np.array(limbSeq[k]) - 1

                for i in range(len(connection_all[k])):  # = 1:size(temp,1)
                    found = 0
                    subset_idx = [-1, -1]
                    for j in range(len(subset)):  # 1:size(subset,1):
                        if subset[j][indexA] == partAs[i] or subset[j][indexB] == partBs[i]:
                            subset_idx[found] = j
                            found += 1

                    if found == 1:
                        j = subset_idx[0]
                        if subset[j][indexB] != partBs[i]:
                            subset[j][indexB] = partBs[i]
                            subset[j][-1] += 1
                            subset[j][-2] += candidate[partBs[i].astype(int), 2] +
                                connection_all[k][i][2]
                    elif found == 2:  # if found 2 and disjoint, merge them
                        j1, j2 = subset_idx
```

```
                        membership = ((subset[j1] >= 0).astype(int) + (subset[j2] >= 0).astype(int))[:-2]
                        if len(np.nonzero(membership == 2)[0]) == 0:  # merge
                            subset[j1][:-2] += (subset[j2][:-2] + 1)
                            subset[j1][-2:] += subset[j2][-2:]
                            subset[j1][-2] += connection_all[k][i][2]
                            subset = np.delete(subset, j2, 0)
                        else:  # as like found == 1
                            subset[j1][indexB] = partBs[i]
                            subset[j1][-1] += 1
                            subset[j1][-2] += candidate[partBs[i].astype(int), 2] +
                                connection_all[k][i][2]

                    # if find no partA in the subset, create a new subset
                    elif not found and k < 17:
                        row = -1 * np.ones(20)
                        row[indexA] = partAs[i]
                        row[indexB] = partBs[i]
                        row[-1] = 2
                        row[-2] = sum(candidate[connection_all[k][i, :2].astype(int), 2]) +
                            connection_all[k][i][2]
                        subset = np.vstack([subset, row])
        # delete some rows of subset which has few parts occur
        deleteIdx = []
        for i in range(len(subset)):
            if subset[i][-1] < 4 or subset[i][-2] / subset[i][-1] < 0.4:
                deleteIdx.append(i)
        subset = np.delete(subset, deleteIdx, axis=0)

        # subset: n*20 array, 0-17 is the index in candidate, 18 is the total score, 19 is the total parts
        # candidate: x, y, score, id
        return candidate, subset, heatmap_avg

if __name__ == "__main__":
    body_estimation = Body('../model/body_pose_model.pth')
    test_image = 'D:\\Human-Action-Recognition-In-The-Dark-master\\Human-Action-Recognition-In-The-Dark-master\\Zero-DCE_code\data\\result\\LIME\\Zero.jpg'
    # test_image = 'D:\\Human-Action-Recognition-In-The-Dark-master\\Human-Action-Recognition-In-The-Dark-master\\EE6222_data\\train_img\\Stand\\Stand_5_4\\frame076.jpg'  #

    oriImg = cv2.imread(test_image)  # B,G,R order
    candidate, subset, heatmap_avg = body_estimation(oriImg)

    #
    canvas = util.draw_bodypose(oriImg, candidate, subset)
    plt.imshow(canvas[:, :, [2, 1, 0]])
    plt.title("Keypoints")
    plt.show()

    #
    plt.imshow(heatmap_avg[:, :, :-1].sum(axis=2))
    plt.title("Heatmaps")
    plt.show()
```

```
from PIL import Image
import matplotlib.pyplot as plt
import cv2
import numpy as np

#
image_path = 'D:\\Human-Action-Recognition-In-The-Dark-master\\Human-Action-Recognition-In-The-Dark-master\\EE6222_data\\enhanced_train_img_4\\Jump\\Jump_8_2\\frame000.jpg'
# "D:\Human-Action-Recognition-In-The-Dark-master\Human-Action-Recognition-In-The-Dark-master\EE6222_data\enhanced_train_img_4\Jump\Jump_8_2\frame000.jpg"
# image_path = 'E:\OneDrive - Nanyang Technological University\Desktop\Figure_11.png'
original_image = cv2.imread(image_path)

#              BGR           LAB
lab_image = cv2.cvtColor(original_image, cv2.COLOR_BGR2LAB)

#     LAB                 L, A,     B
l_channel, a_channel, b_channel = cv2.split(lab_image)

#                CLAHE
clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8, 8))
l_channel = clahe.apply(l_channel)

#
updated_lab_image = cv2.merge((l_channel, a_channel, b_channel))
```

```

#              LAB           BGR
clahe_image = cv2.cvtColor(updated_lab_image, cv2.COLOR_LAB2BGR)
plt.imshow(cv2.cvtColor(clahe_image, cv2.COLOR_BGR2RGB))

#
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.imshow(cv2.cvtColor(original_image, cv2.COLOR_BGR2RGB))
plt.title('Original Image')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(cv2.cvtColor(clahe_image, cv2.COLOR_BGR2RGB))
plt.title('CLAHE Image')
plt.axis('off')

plt.show()
```

```
import os
import pandas as pd
import pickle
import numpy as np
import joblib
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import classification_report, accuracy_score
from sklearn.metrics import top_k_accuracy_score
#
project_root = "D:/Human-Action-Recognition-In-The-Dark-master/Human-Action-Recognition-In-The
    -Dark-master"

def load_features_and_labels(features_file, labels_file):
    #
    features = joblib.load(features_file)
    #
    labels_data = pd.read_csv(labels_file, sep='\t', header=None)
    labels = labels_data[0].values
    #
    features_array = []
    labels_array = []
    for path, data in features.items():
        frame_feature = data['features']
        frame_label = data['label']
        features_array.append(frame_feature)
        labels_array.append(frame_label)
    return np.array(features_array), np.array(labels_array)

#
train_features, train_labels = load_features_and_labels(
    os.path.join(project_root, 'train_video_features.pkl'),
    os.path.join(project_root, 'EE6222_data/train.txt')
)
test_features, test_labels = load_features_and_labels(
    os.path.join(project_root, 'val_video_features.pkl'),
    os.path.join(project_root, 'EE6222_data/validate.txt')
)

def reshape_features(features_array):
    #
    if features_array.ndim == 3:
        n_samples = features_array.shape[0]
        n_features = features_array.shape[2]
        return features_array.reshape(n_samples, n_features)
    return features_array

#
train_features = reshape_features(train_features)
test_features = reshape_features(test_features)

#         SVM
# Ensure your SVM classifier can return probabilities
svm_classifier = SVC(probability=True)
svm_classifier.fit(train_features, train_labels)
#
svm_probs = svm_classifier.predict_proba(test_features)
# The top-1 predictions are simply the class with the highest probability
svm_top1_predictions = np.argmax(svm_probs, axis=1)
# Calculate Top-5 accuracy using the probabilities
```

```
svm_top5_accuracy = top_k_accuracy_score(test_labels, svm_probs, k=3)

print("SVM Classifier Report:")
print(classification_report(test_labels, svm_top1_predictions))
print(f"SVM Top-1 Accuracy: {accuracy_score(test_labels, svm_top1_predictions)}")
print(f"SVM Top-3 Accuracy: {svm_top5_accuracy}\n")

# Naive Bayes naturally gives probabilities, so no need to change the classifier
    initialization
nb_classifier = GaussianNB()
nb_classifier.fit(train_features, train_labels)

# Getting the probabilities for each class
nb_probs = nb_classifier.predict_proba(test_features)
# The top-1 predictions are simply the class with the highest probability
nb_top1_predictions = np.argmax(nb_probs, axis=1)
# Calculate Top-5 accuracy using the probabilities
nb_top5_accuracy = top_k_accuracy_score(test_labels, nb_probs, k=3)

print("Naive Bayes Classifier Report:")
print(classification_report(test_labels, nb_top1_predictions))
print(f"Naive Bayes Top-1 Accuracy: {accuracy_score(test_labels, nb_top1_predictions)}")
print(f"Naive Bayes Top-3 Accuracy: {nb_top5_accuracy}")
# svm_predictions = svm_classifier.predict(test_features)
# print("SVM Classifier Report:")
# print(classification_report(test_labels, svm_predictions))
# print(f"SVM Accuracy: {accuracy_score(test_labels, svm_predictions)}\n")

# #
# nb_classifier = GaussianNB()
# nb_classifier.fit(train_features, train_labels)
# #
# nb_predictions = nb_classifier.predict(test_features)
# print("Naive Bayes Classifier Report:")
# print(classification_report(test_labels, nb_predictions))
# print(f"Naive Bayes Accuracy: {accuracy_score(test_labels, nb_predictions)}")
```

```
from pathlib import Path
import os


DATA_DIR  = Path('EE6222_data')

MODEL_NAME = 'resnet18' # For CNN-RNN only
MODEL_TYPE = 'r2plus1d_18' # 'cnn-rnn', 'r3d_18', 'r2plus1d_18'
IMG_SHAPE = 'BCTHW' #'BCTHW', 'BTCHW'
#SAVED_WEIGHTS_DIR = '/kaggle/input/openpose-r21d-model-state-dict-221027/best_val_loss (1).pt
    '
PRETRAINED = Path('model/state_dict')
CHECKPOINT_DIR = PRETRAINED / 'checkpoint'
MODEL_STATE_DIR = PRETRAINED / 'bestloss'
OPENPOSE_STATE_DIR = 'model/body_pose_model.pth'

SUBMISSION_DIR = Path('model/Submission')

TRAIN_VID_FOLDER = 'train'
TRAIN_IMG_FOLDER = 'train_img'
TRAIN_VID_DIR = DATA_DIR / TRAIN_VID_FOLDER
TRAIN_IMG_DIR = DATA_DIR / TRAIN_IMG_FOLDER
TRAIN_CSV = DATA_DIR / 'train.txt'

VAL_VID_FOLDER = 'validate'
VAL_IMG_FOLDER = 'validate_img'
VAL_VID_DIR = DATA_DIR / VAL_VID_FOLDER
VAL_IMG_DIR = DATA_DIR / VAL_IMG_FOLDER
VAL_CSV = DATA_DIR / 'validate.txt'

INF_VID_FOLDER = 'validate'
INF_IMG_FOLDER = 'validate_img'
INF_VID_DIR = DATA_DIR / INF_VID_FOLDER
INF_IMG_DIR = DATA_DIR / INF_IMG_FOLDER
INF_CSV = DATA_DIR / 'validate.txt'

MAP_TABLE_DIR = DATA_DIR / 'mapping_table_23.txt'
NUM_CLASSES = 6

EXTENSION = ".mp4"
EXTENSION1 = ".jpg"
N_FRAMES = 12
INTERVAL = 5
```

```

SPLITS = 5

IMG_DIM = 224

IMAGENET_MEAN = [0.485, 0.456, 0.406]  # RGB
IMAGENET_STD = [0.229, 0.224, 0.225]  # RGB

DROPOUT = 0.3
RNN_HIDDEN_SIZE = 128
RNN_NUM_LAYERS = 1
ACCUM_ITER = 8

TRAIN_BATCH_SIZE = 6
VAL_BATCH_SIZE = 1 * TRAIN_BATCH_SIZE
TEST_BATCH_SIZE = VAL_BATCH_SIZE

LR = 1e-4 # 1e-3
NUM_EPOCHS = 40
NUM_WARMUP_EPOCHS = 5
NUM_COS_CYCLE = 0.5 # 0.5

TOP_K = 5
```

```
import os
import utils
from config.constants import (TRAIN_VID_FOLDER, TRAIN_IMG_FOLDER, TRAIN_VID_DIR,
                              VAL_VID_FOLDER, VAL_IMG_FOLDER, VAL_VID_DIR,
                              EXTENSION, N_FRAMES)

listOfCategories = os.listdir(TRAIN_VID_DIR)

def list_cat(train_path):
    for cat in listOfCategories:
        print("category:", cat)
        path2acat = os.path.join(train_path, cat)
        listOfSubs = os.listdir(path2acat)
        print("number of sub-folders:", len(listOfSubs))
        print("-"*50)
    return

def sv_frame(path, subfolder, subfolder_jpg):
    for root, dirs, files in os.walk(path, topdown=False):
        #print(f'root: {root}')
        for name in files:
            if EXTENSION not in name:
                continue
            path2vid = os.path.join(root, name)
            #print(path2vid)
            frames, vlen = utils.get_frames(path2vid, n_frames= N_FRAMES)
            path2store = path2vid.replace(subfolder, subfolder_jpg)
            path2store = path2store.replace(EXTENSION, "")
            print(path2store)
            os.makedirs(path2store, exist_ok= True)
            utils.store_frames(frames, path2store)
        print("-"*50)

#sv_frame(TRAIN_VID_DIR, TRAIN_VID_FOLDER, TRAIN_IMG_FOLDER)
#sv_frame(VAL_VID_DIR, VAL_VID_FOLDER, VAL_IMG_FOLDER)
```

```
import cv2
import numpy as np
import torch
import glob
import os
import albumentations as a
from pathlib import Path
from config.constants import N_FRAMES, INTERVAL, IMG_DIM, IMG_SHAPE, OPENPOSE_STATE_DIR
from torch.utils.data import Dataset
from .openpose_pytorch.src.body import Body_HM

# Dataset class with shape (T, C, H, W)

class VideoDataset(Dataset):
    def __init__(self, df, transforms, img_path_col="path", labelAvailable=True):
        self.df = df
        self.transforms = transforms #
        self.img_path_col = img_path_col #
        self.labelAvailable = labelAvailable #
```

```
20
21      def __len__(self):
22          print("here 1") #
23          return len(self.df) #
24
25      def __getitem__(self, idx): #obtain single sample
26
27          # Video path
28          img_folder = self.df[self.img_path_col].iloc[idx] + "/*.jpg"
29          #print(img_folder)
30          # Append path of all frames in a video
31          all_img_path = glob.glob(img_folder)
32          all_img_path = sorted(all_img_path)
33          v_len = len(all_img_path)
34          #print(v_len)
35
36          # Uniformly samples N_FRAMES number of frames
37          if v_len > N_FRAMES*INTERVAL:
38              frame_list = np.arange(np.int64((v_len - (N_FRAMES*INTERVAL))*0.5), np.int64((
                    v_len - (N_FRAMES*INTERVAL))*0.5) + N_FRAMES*INTERVAL, INTERVAL)
39          else:
40              frame_list = np.arange(0, N_FRAMES*INTERVAL, INTERVAL)
41          #print("here 2")   p  r  o  c  e  s  s  12
42          img_path = []
43          for fn in range(v_len):
44              if (fn in frame_list):
45                  img_path.append(all_img_path[fn])
46          #print(img_path)
47          #print(img_path)  o u t p u t 1 2 path
48          images = []
49          for p2i in img_path:
50              p2i = Path(p2i)
51              img = cv2.imread(p2i.as_posix())
52              img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
53              images.append(img)
54          #print("here 3") #after              RGB
55          while (len(images) < N_FRAMES):
56              #if not enough ,the black screen is added for the rest of the frames
57              images.append(np.zeros((IMG_DIM,IMG_DIM,3), np.uint8))
58          images_tr = []
59
60          body_estimation_hm = Body_HM(OPENPOSE_STATE_DIR)
61
62          for image in images:
63              #print("here 4") #       12
64              if len(images_tr) == 0:
65                  augmented = self.transforms(image=image)
66                  image = augmented['image']
67                  body_heatmap = body_estimation_hm(image)
68                  hm = torch.Tensor((1 - body_heatmap[:,:,-1]))
69                  hm = torch.stack((hm,hm,hm), dim = 0)
70                  images_tr.append(torch.Tensor(hm))
71                  data_replay = augmented['replay']
72
73              else:
74                  image = a.ReplayCompose.replay(data_replay, image=image)
75                  image = image['image']
76                  body_heatmap = body_estimation_hm(image)
77                  hm = torch.Tensor((1 - body_heatmap[:,:,-1]))
78                  hm = torch.stack((hm,hm,hm), dim = 0)
79                  images_tr.append(torch.Tensor(hm))
80
81          if len(images_tr)>0:
82              images_tr = torch.stack(images_tr)
83          #print("here 5") #      12
84          if self.labelAvailable == True:
85              # Label
86              label = self.df["label"].iloc[idx]
87              return img_folder, images_tr, label
88          else:
89              return img_folder, images_tr
90
91  # Train image transformation function
92  #
93
94  def get_train_transforms(img_dim):
95      trans = a.ReplayCompose([
96          a.CLAHE(clip_limit=(10, 10), tile_grid_size=(8, 8), always_apply=True),
97          a.RandomGamma((150, 150), always_apply=True),
          a.PadIfNeeded(img_dim, img_dim),
```

```
        a.CenterCrop(img_dim, img_dim, always_apply=True),
        a.HorizontalFlip(p=0.4),
        a.VerticalFlip(p=0.5),
        a.Rotate(limit=120, p=0.8),

    ])
    #print("transfered")
    return trans

# Validation image transformation function
def get_val_transforms(img_dim):
    trans = a.ReplayCompose([
        a.CLAHE(clip_limit=(10, 10), tile_grid_size=(8, 8), always_apply=True),
        a.RandomGamma((150, 150), always_apply=True),
        a.PadIfNeeded(img_dim, img_dim),
        a.CenterCrop(img_dim, img_dim, always_apply=True),
    ])
    #print("transformed-val")
    return trans

#diy COLLATE function
def collate_fn(batch):
    if len(batch[0]) == 3:
        #print("collate-1")
                                                    3
        img_folder_batch, imgs_batch, label_batch = list(zip(*batch))
        label_batch = [torch.tensor(l) for l, imgs in zip(label_batch, imgs_batch) if len(imgs
            )>0]
        labels_tensor = torch.stack(label_batch)
    else:
        #print("collate-2")
        img_folder_batch, imgs_batch = list(zip(*batch))
    print("collate-3")
    img_folder_batch = [folders for folders in img_folder_batch if len(folders)>0]
    imgs_batch = [imgs for imgs in imgs_batch if len(imgs)>0]

    #imgs_tensor = torch.stack(torch.Tensor(imgs_batch))
    if IMG_SHAPE == 'BCTHW':
        #print("if")       BCTHW
        imgs_batch = [torch.transpose(imgs, 1, 0) for imgs in imgs_batch if len(imgs)>0]
    elif IMG_SHAPE == 'BTCHW':
        imgs_batch = [imgs for imgs in imgs_batch if len(imgs)>0]
    #print("collect-5")
    imgs_tensor = torch.stack(imgs_batch)

    if len(batch[0]) == 3:
        #print("collate-6")

        return img_folder_batch, imgs_tensor,labels_tensor
    else:
        return img_folder_batch, imgs_tensor
```

```
import os
import cv2
import numpy as np
from tqdm import tqdm
from scipy.ndimage import gaussian_filter
import matplotlib.pyplot as plt
import torch
import sys
import math
sys.path.append('..')  #
from openpose_pytorch.src import util
from openpose_pytorch.src.model import bodypose_model

#          CLAHE
def apply_clahe(image):
    #          BGRLAB
    lab = cv2.cvtColor(image, cv2.COLOR_BGR2LAB)

    #              LAB
    lab_planes = list(cv2.split(lab))  #

    #         CLAHE
    clahe = cv2.createCLAHE(clipLimit=10.0, tileGridSize=(8, 8))
    lab_planes[0] = clahe.apply(lab_planes[0])  #         LCLAHE

    #
    lab = cv2.merge(lab_planes)

```

```
    #          LABBGR
    enhanced_img = cv2.cvtColor(lab, cv2.COLOR_LAB2BGR)
    return enhanced_img

#        OpenPose
class BodyPoseEstimator(object):
     def __init__(self, model_path):
        self.model = bodypose_model()
        if torch.cuda.is_available():
            self.model = self.model.cuda()
        model_dict = util.transfer(self.model, torch.load(model_path))
        self.model.load_state_dict(model_dict)
        self.model.eval()

     def __call__(self, oriImg):
        # scale_search = [0.5, 1.0, 1.5, 2.0]
        scale_search = [0.5]
        boxsize = 368
        stride = 8
        padValue = 128
        thre1 = 0.1
        thre2 = 0.05
        multiplier = [x * boxsize / oriImg.shape[0] for x in scale_search]
        heatmap_avg = np.zeros((oriImg.shape[0], oriImg.shape[1], 19))
        paf_avg = np.zeros((oriImg.shape[0], oriImg.shape[1], 38))

        for m in range(len(multiplier)):
            scale = multiplier[m]
            imageToTest = cv2.resize(oriImg, (0, 0), fx=scale, fy=scale, interpolation=cv2.
                INTER_CUBIC)
            imageToTest_padded, pad = util.padRightDownCorner(imageToTest, stride, padValue)
            im = np.transpose(np.float32(imageToTest_padded[:, :, :, np.newaxis]), (3, 2, 0,
                1)) / 256 - 0.5
            im = np.ascontiguousarray(im)

            data = torch.from_numpy(im).float()
            if torch.cuda.is_available():
                data = data.cuda()
            # data = data.permute([2, 0, 1]).unsqueeze(0).float()
            with torch.no_grad():
                Mconv7_stage6_L1, Mconv7_stage6_L2 = self.model(data)
            Mconv7_stage6_L1 = Mconv7_stage6_L1.cpu().numpy()
            Mconv7_stage6_L2 = Mconv7_stage6_L2.cpu().numpy()

            # extract outputs, resize, and remove padding
            # heatmap = np.transpose(np.squeeze(net.blobs[output_blobs.keys()[1]].data), (1,
                2, 0))  # output 1 is heatmaps
            heatmap = np.transpose(np.squeeze(Mconv7_stage6_L2), (1, 2, 0))  # output 1 is
                heatmaps
            heatmap = cv2.resize(heatmap, (0, 0), fx=stride, fy=stride, interpolation=cv2.
                INTER_CUBIC)
            heatmap = heatmap[:imageToTest_padded.shape[0] - pad[2], :imageToTest_padded.shape
                [1] - pad[3], :]
            heatmap = cv2.resize(heatmap, (oriImg.shape[1], oriImg.shape[0]), interpolation=
                cv2.INTER_CUBIC)

            # paf = np.transpose(np.squeeze(net.blobs[output_blobs.keys()[0]].data), (1, 2, 0)
                )  # output 0 is PAFs
            paf = np.transpose(np.squeeze(Mconv7_stage6_L1), (1, 2, 0))  # output 0 is PAFs
            paf = cv2.resize(paf, (0, 0), fx=stride, fy=stride, interpolation=cv2.INTER_CUBIC)
            paf = paf[:imageToTest_padded.shape[0] - pad[2], :imageToTest_padded.shape[1] -
                pad[3], :]
            paf = cv2.resize(paf, (oriImg.shape[1], oriImg.shape[0]), interpolation=cv2.
                INTER_CUBIC)

            heatmap_avg += heatmap_avg + heatmap / len(multiplier)
            paf_avg += + paf / len(multiplier)

        all_peaks = []
        peak_counter = 0

        for part in range(18):
            map_ori = heatmap_avg[:, :, part]
            one_heatmap = gaussian_filter(map_ori, sigma=3)

            map_left = np.zeros(one_heatmap.shape)
            map_left[1:, :] = one_heatmap[:-1, :]
            map_right = np.zeros(one_heatmap.shape)
            map_right[:-1, :] = one_heatmap[1:, :]
            map_up = np.zeros(one_heatmap.shape)
            map_up[:, 1:] = one_heatmap[:, :-1]
```

```
            map_down = np.zeros(one_heatmap.shape)
            map_down[:, :-1] = one_heatmap[:, 1:]

            peaks_binary = np.logical_and.reduce(
                (one_heatmap >= map_left, one_heatmap >= map_right, one_heatmap >= map_up,
                     one_heatmap >= map_down, one_heatmap > thre1))
            peaks = list(zip(np.nonzero(peaks_binary)[1], np.nonzero(peaks_binary)[0]))  #
                 note reverse
            peaks_with_score = [x + (map_ori[x[1], x[0]],) for x in peaks]
            peak_id = range(peak_counter, peak_counter + len(peaks))
            peaks_with_score_and_id = [peaks_with_score[i] + (peak_id[i],) for i in range(len(
                 peak_id))]

            all_peaks.append(peaks_with_score_and_id)
            peak_counter += len(peaks)

        # find connection in the specified sequence, center 29 is in the position 15
        limbSeq = [[2, 3], [2, 6], [3, 4], [4, 5], [6, 7], [7, 8], [2, 9], [9, 10], \
                   [10, 11], [2, 12], [12, 13], [13, 14], [2, 1], [1, 15], [15, 17], \
                   [1, 16], [16, 18], [3, 17], [6, 18]]
        # the middle joints heatmap correpondence
        mapIdx = [[31, 32], [39, 40], [33, 34], [35, 36], [41, 42], [43, 44], [19, 20], [21,
             22], \
                  [23, 24], [25, 26], [27, 28], [29, 30], [47, 48], [49, 50], [53, 54], [51,
                       52], \
                  [55, 56], [37, 38], [45, 46]]

        connection_all = []
        special_k = []
        mid_num = 10

        for k in range(len(mapIdx)):
            score_mid = paf_avg[:, :, [x - 19 for x in mapIdx[k]]]
            candA = all_peaks[limbSeq[k][0] - 1]
            candB = all_peaks[limbSeq[k][1] - 1]
            nA = len(candA)
            nB = len(candB)
            indexA, indexB = limbSeq[k]
            if (nA != 0 and nB != 0):
                connection_candidate = []
                for i in range(nA):
                    for j in range(nB):
                        vec = np.subtract(candB[j][:2], candA[i][:2])
                        norm = math.sqrt(vec[0] * vec[0] + vec[1] * vec[1])
                        norm = max(0.001, norm)
                        vec = np.divide(vec, norm)

                        startend = list(zip(np.linspace(candA[i][0], candB[j][0], num=mid_num)
                             , \
                                            np.linspace(candA[i][1], candB[j][1], num=mid_num)
                                                 ))

                        vec_x = np.array([score_mid[int(round(startend[I][1])), int(round(
                             startend[I][0])), 0] \
                                          for I in range(len(startend))])
                        vec_y = np.array([score_mid[int(round(startend[I][1])), int(round(
                             startend[I][0])), 1] \
                                          for I in range(len(startend))])

                        score_midpts = np.multiply(vec_x, vec[0]) + np.multiply(vec_y, vec[1])
                        score_with_dist_prior = sum(score_midpts) / len(score_midpts) + min(
                            0.5 * oriImg.shape[0] / norm - 1, 0)
                        criterion1 = len(np.nonzero(score_midpts > thre2)[0]) > 0.8 * len(
                             score_midpts)
                        criterion2 = score_with_dist_prior > 0
                        if criterion1 and criterion2:
                            connection_candidate.append(
                                [i, j, score_with_dist_prior, score_with_dist_prior + candA[i
                                     ][2] + candB[j][2]])

                connection_candidate = sorted(connection_candidate, key=lambda x: x[2],
                     reverse=True)
                connection = np.zeros((0, 5))
                for c in range(len(connection_candidate)):
                    i, j, s = connection_candidate[c][0:3]
                    if (i not in connection[:, 3] and j not in connection[:, 4]):
                        connection = np.vstack([connection, [candA[i][3], candB[j][3], s, i, j
                             ]])
                        if (len(connection) >= min(nA, nB)):
                            break
```

```
                connection_all.append(connection)
            else:
                special_k.append(k)
                connection_all.append([])

        # last number in each row is the total parts number of that person
        # the second last number in each row is the score of the overall configuration
        subset = -1 * np.ones((0, 20))
        candidate = np.array([item for sublist in all_peaks for item in sublist])

        for k in range(len(mapIdx)):
            if k not in special_k:
                partAs = connection_all[k][:, 0]
                partBs = connection_all[k][:, 1]
                indexA, indexB = np.array(limbSeq[k]) - 1

                for i in range(len(connection_all[k])):  # = 1:size(temp,1)
                    found = 0
                    subset_idx = [-1, -1]
                    for j in range(len(subset)):  # 1:size(subset,1):
                        if subset[j][indexA] == partAs[i] or subset[j][indexB] == partBs[i]:
                            subset_idx[found] = j
                            found += 1

                    if found == 1:
                        j = subset_idx[0]
                        if subset[j][indexB] != partBs[i]:
                            subset[j][indexB] = partBs[i]
                            subset[j][-1] += 1
                            subset[j][-2] += candidate[partBs[i].astype(int), 2] +
                                 connection_all[k][i][2]
                    elif found == 2:  # if found 2 and disjoint, merge them
                        j1, j2 = subset_idx
                        membership = ((subset[j1] >= 0).astype(int) + (subset[j2] >= 0).astype
                             (int))[:-2]
                        if len(np.nonzero(membership == 2)[0]) == 0:  # merge
                            subset[j1][:-2] += (subset[j2][:-2] + 1)
                            subset[j1][-2:] += subset[j2][-2:]
                            subset[j1][-2] += connection_all[k][i][2]
                            subset = np.delete(subset, j2, 0)
                        else:  # as like found == 1
                            subset[j1][indexB] = partBs[i]
                            subset[j1][-1] += 1
                            subset[j1][-2] += candidate[partBs[i].astype(int), 2] +
                                 connection_all[k][i][2]

                    # if find no partA in the subset, create a new subset
                    elif not found and k < 17:
                        row = -1 * np.ones(20)
                        row[indexA] = partAs[i]
                        row[indexB] = partBs[i]
                        row[-1] = 2
                        row[-2] = sum(candidate[connection_all[k][i, :2].astype(int), 2]) +
                             connection_all[k][i][2]
                        subset = np.vstack([subset, row])
        # delete some rows of subset which has few parts occur
        deleteIdx = []
        for i in range(len(subset)):
            if subset[i][-1] < 4 or subset[i][-2] / subset[i][-1] < 0.4:
                deleteIdx.append(i)
        subset = np.delete(subset, deleteIdx, axis=0)

        # subset: n*20 array, 0-17 is the index in candidate, 18 is the total score, 19 is the
             total parts
        # candidate: x, y, score, id
        return candidate, subset, heatmap_avg

#         OpenPose
body_pose_estimator = BodyPoseEstimator('../model/body_pose_model.pth')

#
src_train_folder = 'D:/Human-Action-Recognition-In-The-Dark-master/Human-Action-Recognition-In
     -The-Dark-master/EE6222_data/enhanced_train_img_4'
src_val_folder = 'D:/Human-Action-Recognition-In-The-Dark-master/Human-Action-Recognition-In-
     The-Dark-master/EE6222_data/enhanced_validate_img_4'
dst_train_folder = 'D:/Human-Action-Recognition-In-The-Dark-master/Human-Action-Recognition-In
     -The-Dark-master/EE6222_data/train_img_6'
dst_val_folder = 'D:/Human-Action-Recognition-In-The-Dark-master/Human-Action-Recognition-In-
     The-Dark-master/EE6222_data/validate_img_6'


```

```
#
action_classes = os.listdir(src_train_folder)
for action_class in action_classes:
    action_class_path = os.path.join(src_train_folder, action_class)
    video_folders = os.listdir(action_class_path)
    for video_folder in video_folders:
        video_folder_path = os.path.join(action_class_path, video_folder)
        frame_files = os.listdir(video_folder_path)
        for frame_file in frame_files:
            frame_path = os.path.join(video_folder_path, frame_file)
            oriImg = cv2.imread(frame_path)
            #apply CLAHE
            #enhanced_img = apply_clahe(oriImg)
            # assecc heatmap
            _, _, heatmap_avg = body_pose_estimator(oriImg)

            #
            dst_video_folder = os.path.join(dst_train_folder, action_class, video_folder)
            if not os.path.exists(dst_video_folder):
                os.makedirs(dst_video_folder)
            #
            # heatmap_img = np.uint8(255 * heatmap_avg / np.max(heatmap_avg))  #

            heatmap_path = os.path.join(dst_video_folder, frame_file)
            if len(heatmap_avg.shape) == 3 and heatmap_avg.shape[2] != 1:
                #
                heatmap_img = np.mean(heatmap_avg, axis=2)
            else:
                heatmap_img = heatmap_avg
            #
            heatmap_img_normalized = cv2.normalize(heatmap_img, None, 0, 255, cv2.NORM_MINMAX)
            #               8
            heatmap_img_normalized = np.uint8(heatmap_img_normalized)
            #
            heatmap_img_color = cv2.applyColorMap(heatmap_img_normalized, cv2.COLORMAP_JET)
            #
            cv2.imwrite(heatmap_path, heatmap_img_color)
print("train data available")

video_folders = os.listdir(src_val_folder)
for video_folder in video_folders:
    video_folder_path = os.path.join(src_val_folder, video_folder)
    if os.path.isdir(video_folder_path):
        frame_files = os.listdir(video_folder_path)
        for frame_file in frame_files:
            frame_path = os.path.join(video_folder_path, frame_file)
            oriImg = cv2.imread(frame_path)

            #             CLAHEOpenPose
            # enhanced_img = apply_clahe(oriImg)
            _, _, heatmap_avg = body_pose_estimator(oriImg)

            #
            dst_video_folder = os.path.join(dst_val_folder, video_folder)
            if not os.path.exists(dst_video_folder):
                os.makedirs(dst_video_folder)

            #
            # heatmap_img = np.uint8(255 * heatmap_avg / np.max(heatmap_avg))  #

            heatmap_path = os.path.join(dst_video_folder, frame_file)
            if len(heatmap_avg.shape) == 3 and heatmap_avg.shape[2] != 1:
                #
                heatmap_img = np.mean(heatmap_avg, axis=2)
            else:
                heatmap_img = heatmap_avg
            #
            heatmap_img_normalized = cv2.normalize(heatmap_img, None, 0, 255, cv2.NORM_MINMAX)
            #               8
            heatmap_img_normalized = np.uint8(heatmap_img_normalized)
            #
            heatmap_img_color = cv2.applyColorMap(heatmap_img_normalized, cv2.COLORMAP_JET)
            #
            cv2.imwrite(heatmap_path, heatmap_img_color)

print("Congratulations All Done!!")
```

```
import torch
import torchvision.transforms as transforms
from torchvision.models.video import r3d_18, r2plus1d_18, R3D_18_Weights, R2Plus1D_18_Weights
```

```
import torch.nn as nn
import cv2
import os
import numpy as np
import pandas as pd
import pickle
from pathlib import Path
import os

# from config.constants import (TRAIN_VID_DIR, TRAIN_VID_FOLDER, TRAIN_IMG_FOLDER, VAL_VID_DIR, VAL_VID_FOLDER, VAL_IMG_FOLDER,
#                               MAP_TABLE_DIR, TRAIN_CSV, TRAIN_IMG_DIR, EXTENSION, EXTENSION1, VAL_CSV, VAL_IMG_DIR,
#                               IMG_DIM, TRAIN_BATCH_SIZE, VAL_BATCH_SIZE, MODEL_TYPE,
#                               MODEL_NAME, DROPOUT, RNN_HIDDEN_SIZE, RNN_NUM_LAYERS, LR, ACCUM_ITER,
#                               NUM_WARMUP_EPOCHS, NUM_EPOCHS, NUM_COS_CYCLE, MODEL_STATE_DIR, TOP_K, NUM_CLASSES, CHECKPOINT_DIR, SUBMISSION_DIR)

#
model = r2plus1d_18(weights=R2Plus1D_18_Weights.DEFAULT)
model.eval()  #

DATA_DIR  = Path('EE6222_data')
MAP_TABLE_DIR = DATA_DIR / 'mapping_table_23.txt'

labels = pd.read_csv(MAP_TABLE_DIR, sep="\t", header=None, index_col=0)
label_dict = {}
for i in range(len(labels)):
    label_dict[i] = labels.loc[i, 1]

label_list = labels.index.tolist()

transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.298, 0.276, 0.257], std=[0.209, 0.198, 0.180])
])

def extract_features_from_frames(frames,size=(240, 320)):
    #                                    4D
    frames_resized = [cv2.resize(frame, size, interpolation=cv2.INTER_AREA) for frame in frames]
    frames_tensor = torch.stack([transform(cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)) for frame in frames_resized])
    #                                 (batch_size, channels, depth, height, width)
    #        frames_tensor              [depth, channels, height, width]
    frames_tensor = frames_tensor.permute(1, 0, 2, 3).unsqueeze(0)  #           [batch_size, channels, depth, height, width]

    with torch.no_grad():
        features = model(frames_tensor)  #                            unsqueeze(0)
    if features.dim() > 2:
        pooled_features = torch.mean(features, dim=2)
    else:
        pooled_features = features
    return pooled_features

def process_video_frames(video_frames_folder):
    frames = []
    for frame_name in sorted(os.listdir(video_frames_folder)):
        frame_path = os.path.join(video_frames_folder, frame_name)
        frame = cv2.imread(frame_path)
        if frame is not None:
            frames.append(frame)
    return extract_features_from_frames(frames)

project_root = 'D:/Human-Action-Recognition-In-The-Dark-master/Human-Action-Recognition-In-The-Dark-master/'
TRAIN_CSV = os.path.join(project_root, 'EE6222_data/train.txt')
VAL_CSV = os.path.join(project_root, 'EE6222_data/validate.txt')
TRAIN_IMG_DIR = os.path.join(project_root, 'EE6222_data/train_img_6')
VAL_IMG_DIR = os.path.join(project_root, 'EE6222_data/validate_img_6')
EXTENSION = '.mp4'


# #
df_train = pd.read_csv(TRAIN_CSV, sep="\t", header=None, index_col=0)
df_train.columns = ['label', 'path']
df_train['path'] = str(TRAIN_IMG_DIR) + '/' + df_train['path'].str.replace(EXTENSION, "",regex=True)

```

```
df_val = pd.read_csv(VAL_CSV, sep="\t", header=None, index_col=0)
df_val.columns = ['label', 'path']
df_val['path'] = str(VAL_IMG_DIR) + '/' + df_val['path'].str.replace(EXTENSION, "",regex=True)


#
def extract_and_save_features(df, filename):
    features_dict = {}
    for _, row in df.iterrows():
        video_folder_name = row['path'].replace(".mp4", "")  #       . m p 4
        video_folder_path = os.path.join(TRAIN_IMG_DIR if 'train' in filename else VAL_IMG_DIR
            , video_folder_name)
        if not os.path.exists(video_folder_path):
            print(f"               : {video_folder_path}")
            continue
        label = row['label']
        features = process_video_frames(video_folder_path)
        features_dict[video_folder_path] = {'features': features.cpu().numpy(), 'label': label
            }
        print(f"Processed {video_folder_path}, Features Shape: {features.shape}")

    with open(filename, 'wb') as f:
        pickle.dump(features_dict, f)
    print(f"                  : {filename}")

#
extract_and_save_features(df_train, 'train_video_features2.pkl')

#
extract_and_save_features(df_val, 'val_video_features2.pkl')

print("                              ")
```

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
import random

def extract_frames(video_path, sample_interval=5, num_frames=16, random_sampling=False):
    cap = cv2.VideoCapture(video_path)
    if not cap.isOpened():
        print(f"Error opening video file: {video_path}")
        return []

    total_frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
    if total_frames == 0:
        print(f"No frames found in video: {video_path}")
        return []

    if random_sampling:
        frame_indices = random.sample(range(total_frames), min(num_frames, total_frames))
    else:
        frame_indices = [i * sample_interval for i in range(num_frames)]

    frames = []
    for i in range(max(frame_indices) + 1):
        ret, frame = cap.read()
        if i in frame_indices and ret:
            frames.append(cv2.cvtColor(frame, cv2.COLOR_BGR2RGB))
        elif i in frame_indices:
            frame_height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
            frame_width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
            frames.append(np.zeros((frame_height, frame_width, 3), dtype=np.uint8))

    cap.release()
    return frames

def plot_frames(frames, title):
    fig, axs = plt.subplots(4, 4, figsize=(10, 10))
    axs = axs.flatten()
    for i in range(16):
        if i < len(frames) and frames[i].size != 0:
            axs[i].imshow(frames[i])
        else:
            axs[i].imshow(np.zeros((100, 100, 3), dtype=np.uint8))  #

        axs[i].axis('off')
        axs[i].set_title(f'frame_{i+1}')  #
    plt.suptitle(title)
    plt.tight_layout()
```

```
    plt.show()

#
video_path = 'D:\\Human-Action-Recognition-In-The-Dark-master\\Human-Action-Recognition-In-The-Dark-master\\EE6222_data\\train\\Jump\\Jump_8_5.mp4'  #


#
uniform_frames = extract_frames(video_path, sample_interval=5, num_frames=16, random_sampling=False)
plot_frames(uniform_frames, "Uniform Sampling")

#
random_frames = extract_frames(video_path, num_frames=16, random_sampling=True)
plot_frames(random_frames, "Random Sampling")
```

```
import torch
import torchvision
from torchvision import transforms
from PIL import Image
import os
import numpy as np
import model
import glob
import time

def lowlight(image_path, model_path, result_path):
    data_lowlight = Image.open(image_path)
    data_lowlight = (np.asarray(data_lowlight)/255.0)
    data_lowlight = torch.from_numpy(data_lowlight).float()
    data_lowlight = data_lowlight.permute(2, 0, 1)
    data_lowlight = data_lowlight.cuda().unsqueeze(0)

    DCE_net = model.enhance_net_nopool().cuda()
    DCE_net.load_state_dict(torch.load(model_path))
    _, enhanced_image, _ = DCE_net(data_lowlight)

    if not os.path.exists(os.path.dirname(result_path)):
        os.makedirs(os.path.dirname(result_path))
    torchvision.utils.save_image(enhanced_image, result_path)

def process_dataset(dataset_path, result_root, model_path):
    os.environ['CUDA_VISIBLE_DEVICES']='0'
    for root, dirs, files in os.walk(dataset_path):
        for file in files:
            if file.endswith('.jpg') or file.endswith('.png'):
                image_path = os.path.join(root, file)
                result_path = image_path.replace(dataset_path, result_root).replace('\\', '/')
                #print(f'Processing {image_path}')
                lowlight(image_path, model_path, result_path)

if __name__ == '__main__':
    model_path = 'snapshots/Epoch99.pth'
    train_data_path = 'D:/Human-Action-Recognition-In-The-Dark-master/Human-Action-Recognition-In-The-Dark-master/EE6222_data/train_img_5'
    val_data_path = 'D:/Human-Action-Recognition-In-The-Dark-master/Human-Action-Recognition-In-The-Dark-master/EE6222_data/validate_img_5'
    result_train_root = 'D:/Human-Action-Recognition-In-The-Dark-master/Human-Action-Recognition-In-The-Dark-master/EE6222_data/enhanced_train_img_7'
    result_val_root = 'D:/Human-Action-Recognition-In-The-Dark-master/Human-Action-Recognition-In-The-Dark-master/EE6222_data/enhanced_validate_img_7'

    with torch.no_grad():
        process_dataset(train_data_path, result_train_root, model_path)
        process_dataset(val_data_path, result_val_root, model_path)
```

```
import pandas as pd
import torch
from config.constants import (INF_CSV, INF_IMG_DIR, EXTENSION, IMG_DIM, TEST_BATCH_SIZE, NUM_CLASSES,
                              MODEL_TYPE, MODEL_NAME, DROPOUT, RNN_HIDDEN_SIZE, RNN_NUM_LAYERS, MODEL_STATE_DIR, SUBMISSION_DIR,
                              VAL_CSV, VAL_IMG_DIR, TOP_K, INF_VID_DIR, INF_VID_FOLDER, INF_IMG_FOLDER)
from model.data import get_val_transforms, collate_fn, VideoDataset
from model.model import HARModel
from model.inference_gndtruth import inference_loop
from torch.utils.data import DataLoader
from torchvision.models.video import r3d_18, r2plus1d_18, R3D_18_Weights, R2Plus1D_18_Weights
```

```
from sklearn.metrics import accuracy_score, top_k_accuracy_score, confusion_matrix
from data_prep import sv_frame
import torch.nn as nn

# sv_frame(INF_VID_DIR, INF_VID_FOLDER, INF_IMG_FOLDER)

df_test = pd.read_csv(VAL_CSV, sep="\t", header=None, index_col=0)
df_test.columns = ['label', 'path']
df_test['path'] = str(INF_IMG_DIR) + '/' + df_test['path'].str.replace(EXTENSION, "",regex=True)
print(df_test.head())

#df_test = pd.read_csv(INF_CSV, sep="\t", header=None, index_col=0)
#df_test.columns = ['path']
#df_test['path'] = str(INF_IMG_DIR) + '/' + df_test['path'].str.replace(EXTENSION, "")

inf_transforms = get_val_transforms(IMG_DIM)
print("here-1,inf")
inf_data = VideoDataset(df=df_test,
                        transforms=inf_transforms,
                        labelAvailable=True)
print("here-2,inf")
inf_loader  = DataLoader(inf_data,
                         batch_size=TEST_BATCH_SIZE,
                         shuffle=False,
                         collate_fn=collate_fn)
print("here-3,inf")
device = 'cuda' if torch.cuda.is_available() else 'cpu'

if MODEL_TYPE == 'cnn-rnn':
    model = HARModel(
        model_name=MODEL_NAME,
        dropout=DROPOUT,
        rnn_hidden_size=RNN_HIDDEN_SIZE,
        rnn_num_layers=RNN_NUM_LAYERS,
        num_classes=NUM_CLASSES,
        pretrained=True)
elif MODEL_TYPE == 'r3d_18':
    model = r3d_18(weights=None, progress=False)
    num_features = model.fc.in_features
    model.fc = nn.Linear(num_features, NUM_CLASSES)
elif MODEL_TYPE == 'r2plus1d_18':
    model = r2plus1d_18(weights=None, progress=False)
    num_features = model.fc.in_features
    model.fc = nn.Linear(num_features, NUM_CLASSES)
print(MODEL_STATE_DIR,'MODEL_STATE_DIR')
model_paths = sorted(list(MODEL_STATE_DIR.glob('best*')))
print(model_paths,'model_paths')
overall_results = dict()
print(overall_results,'overall_results')
for m in range(len(model_paths)):

    model_path = model_paths[m]

    overall_results[m] = inference_loop(model, model_path, inf_loader, device)


for i, model_result in overall_results.items():
    model_folder, model_logits, model_pred, model_target, model_loss = model_result
    model_score = accuracy_score(model_target, model_pred)
    model_con_mat = confusion_matrix(model_target, model_pred, labels=[0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
    model_top_k_score = top_k_accuracy_score(model_target, model_logits, k=TOP_K, labels=[0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
    print(f'Model accuracy score: {model_score}')
    print(f'Model Top-{TOP_K} score: {model_top_k_score}')
    print(f'Model loss: {model_loss}')
    print(f'Confusion matrix: \n{model_con_mat}')
    predictions = pd.Series(model_pred, name='prediction')
    predictions.to_csv(SUBMISSION_DIR / 'vr.txt', sep='\t', header=False)
```

```
import pandas as pd
import numpy as np
from torch.utils.data import DataLoader
from config.constants import (TRAIN_VID_DIR, TRAIN_VID_FOLDER, TRAIN_IMG_FOLDER, VAL_VID_DIR, VAL_VID_FOLDER, VAL_IMG_FOLDER,
                              MAP_TABLE_DIR, TRAIN_CSV, TRAIN_IMG_DIR, EXTENSION, VAL_CSV, VAL_IMG_DIR,
                              IMG_DIM, TRAIN_BATCH_SIZE, VAL_BATCH_SIZE, MODEL_TYPE,
```

```
                            MODEL_NAME, DROPOUT, RNN_HIDDEN_SIZE, RNN_NUM_LAYERS, LR, ACCUM_ITER,
                            NUM_WARMUP_EPOCHS, NUM_EPOCHS, NUM_COS_CYCLE, MODEL_STATE_DIR, TOP_K, NUM_CLASSES, CHECKPOINT_DIR, SUBMISSION_DIR)
from model.data import get_train_transforms, get_val_transforms, collate_fn, VideoDataset
from transformers import get_cosine_schedule_with_warmup
from model.model import HARModel
from model.train import train_loop, val_loop
from data_prep import sv_frame
from torchvision.models.video import r3d_18, r2plus1d_18, R3D_18_Weights, R2Plus1D_18_Weights
import torch.nn as nn
import torch
import matplotlib.pyplot as plt

# sv_frame(TRAIN_VID_DIR, TRAIN_VID_FOLDER, TRAIN_IMG_FOLDER)
# sv_frame(VAL_VID_DIR, VAL_VID_FOLDER, VAL_IMG_FOLDER)

# Label dictionary
# {0: 'Drink', 1: 'Jump', 2: 'Pick', 3: 'Pour', 4: 'Push', 5: 'Run', 6: 'Sit', 7: 'Stand', 8: 'Turn', 9: 'Walk'}
labels = pd.read_csv(MAP_TABLE_DIR, sep="\t", header=None, index_col=0)
label_dict = {}
for i in range(len(labels)):
    label_dict[i] = labels.loc[i, 1]

label_list = labels.index.tolist()

# Load training data
df_train = pd.read_csv(TRAIN_CSV, sep="\t", header=None, index_col=0)
df_train.columns = ['label', 'path']
df_train['path'] = str(TRAIN_IMG_DIR) + '/' + df_train['path'].str.replace(EXTENSION, "",regex=True)

df_val = pd.read_csv(VAL_CSV, sep="\t", header=None, index_col=0)
df_val.columns = ['label', 'path']
df_val['path'] = str(VAL_IMG_DIR) + '/' + df_val['path'].str.replace(EXTENSION, "",regex=True)

# for testing,if training ,remove this two rows
# df_train = df_train.iloc[:10]
# df_val = df_val.iloc[:10]

train_transforms = get_train_transforms(IMG_DIM)
val_transforms = get_val_transforms(IMG_DIM)

train_data = VideoDataset(df=df_train,
                          transforms=train_transforms,
                          labelAvailable=True)

val_data = VideoDataset(df=df_val,
                        transforms=val_transforms,
                        labelAvailable=True)

train_loader = DataLoader(train_data, batch_size= TRAIN_BATCH_SIZE,
                    shuffle=True, collate_fn= collate_fn)
val_loader  = DataLoader(val_data, batch_size= VAL_BATCH_SIZE,
                    shuffle=False, collate_fn= collate_fn)

fold_results = dict()
device = 'cuda' if torch.cuda.is_available() else 'cpu'

if MODEL_TYPE == 'cnn-rnn':
    model = HARModel(
        model_name=MODEL_NAME,
        dropout=DROPOUT,
        rnn_hidden_size=RNN_HIDDEN_SIZE,
        rnn_num_layers=RNN_NUM_LAYERS,
        num_classes=NUM_CLASSES,
        pretrained=True)
elif MODEL_TYPE == 'r3d_18':
    model = r3d_18(weights=R3D_18_Weights.DEFAULT, progress=False)
    num_features = model.fc.in_features
    model.fc = nn.Linear(num_features, NUM_CLASSES)
elif MODEL_TYPE == 'r2plus1d_18':
    model = r2plus1d_18(weights=R2Plus1D_18_Weights.DEFAULT, progress=False)
    num_features = model.fc.in_features
    model.fc = nn.Linear(num_features, NUM_CLASSES) #

model = model.to(device)
print("CUDA")
optimizer = torch.optim.AdamW(model.parameters(), lr=LR)
# Configs for scheduler
```

```
#print(len(train_loader),'len(train_loader)')
#print(ACCUM_ITER,'ACCUM_ITER')          8
num_steps_per_epoch = np.round_(len(train_loader)/ACCUM_ITER)
#print(num_steps_per_epoch,'num_steps_per_epoch')                  5
num_warmup_steps = NUM_WARMUP_EPOCHS * num_steps_per_epoch
#print(num_warmup_steps,'num_warmup_steps')            5
num_training_steps = NUM_EPOCHS * num_steps_per_epoch
#print(num_training_steps,'num_training_steps')             10

scheduler = get_cosine_schedule_with_warmup(
    optimizer=optimizer,
    num_warmup_steps=num_warmup_steps,
    num_training_steps=num_training_steps,
    num_cycles=NUM_COS_CYCLE
)

best_accuracy = 0
best_val_loss = 999_999
fold_train_loss = []
fold_val_loss = []
fold_accuracy = []
fold_k_score = []

for epoch_num in range(NUM_EPOCHS):
    lr_list = []
    epoch_info = f'Epoch {epoch_num+1}/{NUM_EPOCHS}'
    print(epoch_info,'epoch_info')
    lr_list, train_losses = train_loop(
        model, train_loader, device, optimizer, ACCUM_ITER,
        scheduler=scheduler,epoch_info=epoch_info
    )

    val_losses, score, top_k_score = val_loop(
        model, val_loader, device, label_list=label_list, k=TOP_K, epoch_info=epoch_info
    )

    avg_train_loss = np.mean(train_losses)
    avg_val_loss = np.mean(val_losses)
    avg_val_accuracy_score = np.mean(score)
    avg_val_top_k_accuracy = np.mean(top_k_score)

    fold_train_loss.append(avg_train_loss)
    fold_val_loss.append(avg_val_loss)
    fold_accuracy.append(avg_val_accuracy_score)
    fold_k_score.append(avg_val_top_k_accuracy)

    # Save model if val loss improves
    if avg_val_loss < best_val_loss:
        model_state_dict = model.state_dict()

        model_name = f'best_val_loss.pt'
        torch.save(
            model.state_dict(),
            MODEL_STATE_DIR / model_name
        )

        best_val_loss = avg_val_loss

    # Save checkpoint at the last
    if epoch_num == NUM_EPOCHS - 1:
        model_name = f'model_checkpoint.pt'
        torch.save({'epoch':epoch_num,
                    'model_state_dict': model.state_dict(),
                    'optimizer_state_dict': optimizer.state_dict(),
        }, CHECKPOINT_DIR / model_name)

fold_results[0] = (fold_train_loss, fold_val_loss, fold_accuracy, fold_k_score)

(train_loss, val_loss, accuracy, k_score) = fold_results[0]

plt.figure(figsize=(5, 5))
plt.plot(train_loss, label='Train')
plt.plot(val_loss, label='Val')
plt.legend(loc='upper right')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title(f'Training Loss and Validation Loss')
plt.show()

plt.figure(figsize=(5, 5))
plt.plot(fold_accuracy, label='Accuracy')
```

```
plt.legend(loc='upper right')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title(f'Validation Accuracy')
plt.show()
```

A.2 RESULT

Table 3: Classification Performance

Classifier	Precision	Recall	F1-Score	Support	Accuracy
SVM Classifier					
0	0.21	0.35	0.27	17	
1	0.19	0.20	0.19	15	
2	0.12	0.07	0.09	15	
3	0.20	0.25	0.22	16	
4	0.44	0.24	0.31	17	
5	0.13	0.12	0.13	16	
Average	0.22	0.20	0.20	96	0.2083
Top-5 Accuracy					0.90625
Naive Bayes Classifier					
0	0.18	0.17	0.18	17	
1	0.07	0.07	0.07	15	
2	0.19	0.20	0.19	15	
3	0.20	0.44	0.27	16	
4	0.57	0.24	0.33	17	
5	0.10	0.06	0.08	16	
Average	0.22	0.20	0.19	96	0.1979
Top-5 Accuracy					0.875

Table 4: Classification Performance

Classifier	Precision	Recall	F1-Score	Support	Accuracy
SVM Classifier					
0	0.17	0.12	0.14	17	
1	0.14	0.27	0.19	15	
2	0.7	0.07	0.07	15	
3	0.14	0.38	0.21	16	
4	0.38	0.29	0.33	17	
5	0.03	0.02	0.01	16	
Average	0.14	0.18	0.15	96	0.17708
Top-3 Accuracy					0.6046
Naive Bayes Classifier					
0	0.07	0.07	0.07	17	
1	0.17	0.17	0.18	15	
3	0.14	0.44	0.23	16	
2	0.25	0.33	0.29	15	
4	0.47	0.41	0.44	16	
5	0.01	0.03	0.02	17	
Average	0.18	0.22	0.19	96	0.2179
Top-3 Accuracy					0.4479