HUMAN ACTION RECOGNITION IN THE DARK: A RESNET-BASED MODEL WITH LATE FUSION AND IMAGE ENHANCEMENT

Ke Xueyi
School of Electrical and Electronics Engineering
Nanyang Techonological University
xke001@e.ntu.edu.sg

ABSTRACT

This study addresses the challenge of Human Action Recognition (HAR) in low-light environments, where existing models typically underperform. We developed a HAR system using uniform sampling for preprocessing and a 2D+1D ResNet18-based architecture for feature extraction. Notably, we implemented late fusion and a custom classifier, composed of two linear layers, a ReLU activation function, and a dropout layer to mitigate overfitting, which turned out to have better performance than classic classifiers like supported vector machine (SVM). This end-to-end neural network demonstrates better performance over traditional 2D pretrained ResNet50 model in dark videos. We also investigate the effect of image enhancement on HAR accuracy in low-light settings. Our study suggests that image enhancement, although beneficial for human observers, does not uniformly translate to improved machine recognition in HAR, especially in dark conditions.

1 INTRODUCTION

1.1 BACKGROUND

The surge in video content, particularly on platforms like YouTube, underscores the need for advanced video analysis techniques. Human Action Recognition (HAR) plays a pivotal role in this landscape, automating the classification of human actions in videos. Driven by large-scale datasets and advancements in neural networks, HAR is becoming increasingly vital in areas such as security and autonomous systems.

Yet, a significant challenge in HAR research is its reliance on datasets that mainly feature well-lit environments. This creates a notable performance disparity in low-light conditions, a critical issue for applications like nighttime security surveillance where detecting activities in the dark is essential.

This study aims to bridge this gap by investigating HAR in dimly lit settings. We employ techniques like late fusion Karpathy et al. (2014) and image enhancement to evaluate their impact on HAR efficiency in such environments. We proposed an end-to-end network based on pre-trained ResNet He et al. (2016).

This study utilizes a specialized dataset comprising videos from six distinct action classes, with a focus on improving action recognition accuracy under reduced lighting conditions.

1.2 BRIEF OVERVIEW OF PREVIOUS STUDY

The ARID dataset Xu et al. (2021) is a significant stride forward, designed specifically for dark conditions, and includes more than 3,780 video clips spanning 11 action categories. This pioneering dataset enriches the scope of HAR research by facilitating the study of human actions in poorly lit scenarios.

One of the notable findings from the ARID research is the efficacy of 3D ResNet models in dark environments. When benchmarked against other models, 3D ResNet stands out for its superior performance in recognizing actions under low-light conditions.

However, our study diverges in approach due to practical constraints. Despite the proven effectiveness of 3D ResNet models, we have chosen to implement simpler models, which is a 2D + 1D ResNet, considering the requirement of uniform sampling and the dataset constraints.

1.3 DATASET

The dataset under consideration comprises a total of 246 short videos captured in low-light conditions, categorized into six distinct classes: jump, run, sit, stand, turn, and walk. These are further divided into 150 items for the training set and 96 for the validation set. As illustrated in Figure 1, the class distribution across these sets is relatively uniform, indicating a lack of significant imbalance.

However, the size of the training set presents a limitation, being relatively small not only for the training of deep learning networks but also for traditional machine learning models. This constraint is compounded by the absence of a dedicated test set, which limits our ability to effectively evaluate the model's performance. Consequently, the model tuning process relies heavily on the validation dataset, which may not provide a comprehensive assessment of the model's generalization capabilities.

Figure 1: The distribution of each class for each set

2 METHODOLOGY: STEP BY STEP

Followed by the step by step requirements in the doc of the assignment, we will introduce our method with very intuitive instruction to build such a net work for HAR using uniform sampling and late fusion.

2.1 FRAME SAMPLING: UNIFORM SAMPLING

In video analysis, two primary methods for frame sampling are typically considered: uniform sampling and random sampling. Uniform sampling involves selecting frames at regular intervals throughout a video. This can be achieved either by taking every nth frame or by selecting a fixed number of frames evenly distributed across the video. On the other hand, random sampling involves selecting frames without any specific interval, based purely on randomness.

To illustrate these concepts, let's consider a human jumping video from our training dataset, specifically file Jump_11_1.mp4, after performing both random and uniform sampling (see Figure 2). Without any sampling, the sheer volume of frames from the video is substantial. For instance, in a set of 15 frames, the subject is only just beginning to jump. With random sampling, the sequence of the subject's movements appears disjointed and chaotic, almost as if the jump occurs twice. In contrast, uniform sampling effectively reduces the data size while preserving accurate and essential information about the movement.

Given that our videos capture transient human actions such as jumping, running, and sitting, which are brief in duration, uniform sampling emerges as a more suitable approach. In this study, we have

Figure 2: Different sampling techniques. The first row shows the original frames, the second row demonstrates uniform sampling, and the third exhibits random sampling.

opted to select 15 frames per video. Details on the implementation of this method can be found in the appendix.

2.2 Feature Extraction: 2D+1D ResNet

2.2.1 Model Implementation

For feature extraction from video data, we utilize the 2D+1D ResNet model Tran et al. (2018). This model is uniquely suited for handling video inputs as it preserves temporal information in the data sequence, which is crucial for accurately interpreting human activities. Unlike a traditional ResNet, which treats video frames as independent images, the 2D+1D ResNet combines spatial feature extraction (using 2D convolutions) with temporal feature extraction (using 1D convolutions). This hybrid approach allows the model to effectively capture the dynamics of human movements over time, making it particularly advantageous for Human Activity Recognition (HAR) in video data.

The chosen pre-trained 2D+1D ResNet model excels in extracting intricate features from each frame while maintaining temporal relationships between them. This is pivotal in understanding the sequence of actions within the video, which is often lost in models that do not explicitly account for temporal data. The pre-trained nature of the model also provides a robust foundation of learned features, enhancing its performance and efficiency in our specific application.

As shown in Figure 3, the architecture of the 2D+1D model differs significantly from traditional 2D networks. The figure illustrates various ResNet structures, highlighting how the (2+1)D convolution in our model integrates spatial and temporal dimensions, an essential aspect for effective video analysis.

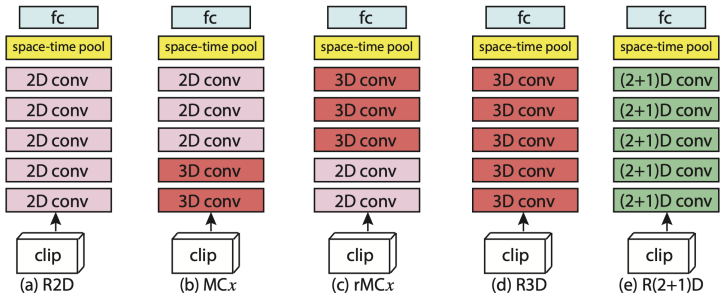

Figure 3: (a) R2D are 2D ResNets; (b) MCx are ResNets with mixed convolutions (MC3 is presented in this figure); (c) rMCx use reversed mixed convolutions (CMC3 is shown here); (d) R3D are 3D ResNets; and (e) R(2+1)D are ResNets with (2+1)D convolutions. For interpretability, residual connections are omitted.

2.2.2 LATE FUSION STRATEGY

To effectively synthesize the extracted features, we employ a late fusion strategy. This approach involves first obtaining features from each uniformly sampled frame and then integrating them just before the classification stage. We implement this by applying an average pooling operation across all sampled frames, which facilitates the fusion of temporal information.

To accommodate this late fusion process, the model underwent specific modifications. We removed the final fully connected layer, which is typically used for classification in standard architectures. In its place, we introduced a late fusion mechanism. This alteration is crucial for preserving the integrity of the feature structure unique to each video frame, while also allowing for an effective amalgamation of these features for final classification. This strategy ensures that the temporal dynamics captured by our 2D+1D ResNet are fully utilized in the decision-making process.

2.2.3 FEATURES' STURCTURE

The feature extraction process for video frames involves the following steps,see appendix for more code implementation:

1. Individual video frames are processed and converted into tensors.
2. These frames are stacked to form a high-dimensional tensor representing the video.
3. An additional dimension for batch size is added at the beginning of the tensor, changing its shape to $1 \times T \times C \times H \times W$, where T is the number of frames.
4. The dimensions are permuted to $1 \times C \times T \times H \times W$ to fit the input format of the feature extractor.
5. The feature extractor processes this tensor to extract relevant features from the video frames.

Our dataset consists of two parts: training data and validation data. After feature extraction process, the following table1 summarizes the dimensions of our dataset:

Table 1: Dimensions of the Training and Validation Datasets

Data Type	**Batch Size**	**Channels $\times$ Height $\times$ Width**
Training Data	150	$512 \times 1 \times 1$
Validation Data	96	$512 \times 1 \times 1$

2.3 CLASSIFIER TRAINING AND EVALUATION

2.3.1 FULLY CONNECTED (FC) LAYERS

In our endeavor to classify video-based human actions, we adopted a simplistic yet effective approach by incorporating a Fully Connected (FC) layer as our classifier. This layer, trained on features extracted through a 2D+1D ResNet model and corresponding labels, was chosen for its ease of implementation and proficiency in learning non-linear feature combinations.

Despite its potential, we encountered persistent overfitting, a formidable challenge akin to a recurring nightmare, throughout our experiments. This was particularly pronounced post-epoch 50, as evidenced by the performance metrics of our trained classifier (Figure 4).

The FC classifier's performance is detailed in Table 2.

Moreover, a confusion matrix analysis was conducted to further scrutinize the classifier's performance (Figure 5).

Pros:

- *Simplicity:* The implementation and integration of FC layers into existing architectures are straightforward, facilitating rapid experimentation and refinement.

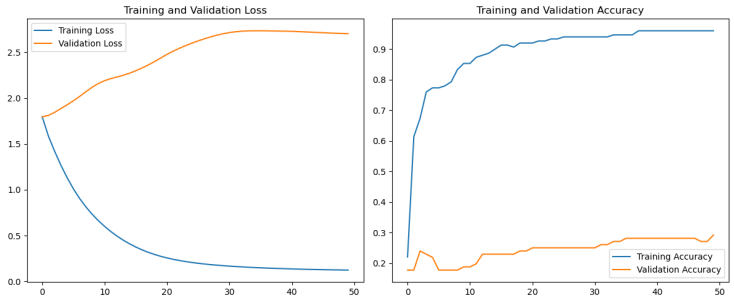

Figure 4: The left graph depicts the loss functions for both training and validation sets, while the right graph illustrates accuracy. A notable increase in the validation set's loss function signifies clear overfitting.

Table 2: Evaluation of the FC Classifier

Class	Precision	Recall	F1-Score	Support
Jump	0.38	0.29	0.33	17
Run	0.33	0.40	0.36	15
Sit	0.38	0.20	0.26	15
Stand	0.39	0.56	0.46	16
Turn	0.00	0.00	0.00	17
Walk	0.17	0.31	0.22	16
Accuracy			0.29	96
Macro Avg	0.28	0.29	0.27	96
Weighted Avg	0.27	0.29	0.27	96

- *Non-linear Mapping:* They adeptly learn non-linear mappings from extracted features to output space, essential for complex pattern recognition in Human Action Recognition (HAR).
- *End-to-End Training:* In conjunction with CNNs as feature extractors, FC layers allow for seamless end-to-end training, optimizing both features and classifier simultaneously.

Cons:

- *Overfitting:* The large number of parameters in FC layers makes them prone to overfitting, a challenge exacerbated by our limited dataset size.
- *Computational Intensity:* Due to the high dimensionality of video data, FC layers can be computationally demanding in terms of both memory and processing power.
- *Limited Interpretability:* The opaque nature of deep networks often leads to reduced interpretability of the learned features and decision-making process.

2.3.2 Supported Vector Machine (SVM)

In contrast to the FC layer, we also implemented a Support Vector Machine (SVM) classifier as a comparative study. Despite its robustness and effectiveness in high-dimensional spaces, the SVM did not outperform the FC layer in our experimental setup, the outcome is shown on the figure 3 below.

The SVM classifier exhibited modest proficiency in recognizing 'Stand' and 'Run' actions, as indicated by their relatively higher F1-Scores. However, it struggled notably with 'Sit' and 'Turn' actions, as reflected by the zero scores. The overall accuracy of 21.9% suggests a need for significant enhancements.

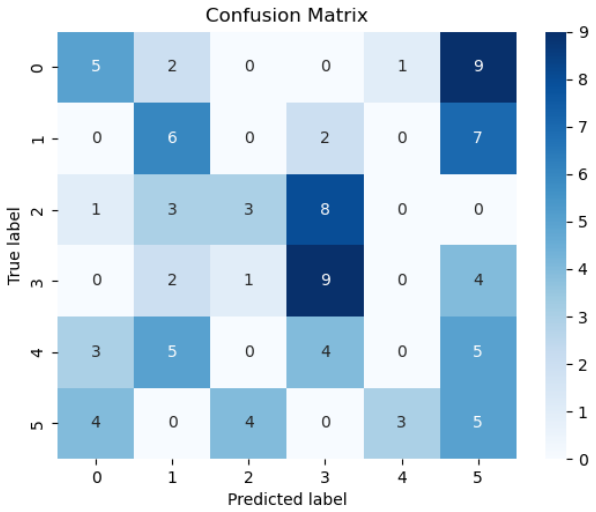

Figure 5: Confusion matrix of the FC classifier.

Table 3: The Evaluation of SVM

Class	Precision	Recall	F1-Score	Support
Jump	0.13	0.12	0.13	17
Run	0.26	0.33	0.29	15
Sit	0.00	0.00	0.00	15
Stand	0.273	0.563	0.37	16
Turn	0.00	0.00	0.00	17
Walk	0.20	0.31	0.24	16
Accuracy			0.22	96
Macro Avg	0.15	0.22	0.17	96
Weighted Avg	0.14	0.22	0.17	96

2.3.3 Results Evaluation

The underperformance of both the Fully Connected (FC) and Support Vector Machine (SVM) classifiers can be primarily attributed to the limited size and diversity of our dataset, coupled with the inherent challenges in distinguishing subtle actions, especially in low-light video conditions. Expanding the dataset both in size and variety, alongside implementing data augmentation techniques, could potentially mitigate these limitations.

Another contributing factor is the nature of the dataset itself. As evidenced by the performance metrics, both classifiers struggled significantly with movements such as 'Sit' and 'Turn.' These actions typically induce minimal changes in the global pixel values of video frames, leading the network to potentially misinterpret them as inconsequential background noise rather than significant movements.

Next, we will explore the impact of image enhancement techniques on the classifiers' performance. By improving the quality of the video frames, we aim to determine whether such enhancements can aid the machine learning models in more accurately classifying human actions.

3 IMAGE ENHANCEMENT FOR HAR

In this section, we explore the impact of image enhancement techniques on our Human Action Recognition (HAR) model. Recognizing actions in poorly lit videos is a challenge, and image enhancements could be key in making video frames clearer and more discernible. We will apply selected image enhancement methods and assess how they affect our trained classifiers' performance, focusing on changes in the reference mean and standard deviation values of video frames. Additionally, we will provide samples of enhanced video frames to visually demonstrate the effects of these enhancements. This study aims to determine if image enhancements can significantly improve the accuracy of action recognition in HAR models.

3.1 HISTOGRAM EQUALIZATION

In our approach, we employ histogram equalization (HE), a proven technique for enhancing image contrast. This method begins by converting the original image to grayscale, simplifying the data to a single channel. We then apply histogram equalization to this grayscale image, effectively spreading out its most frequent intensity values. This process enhances the overall contrast, particularly improving visibility in regions that were initially too bright or too dark. Finally, we convert the enhanced image back to the BGR color space. We take a sit action as an example to show how it works6.

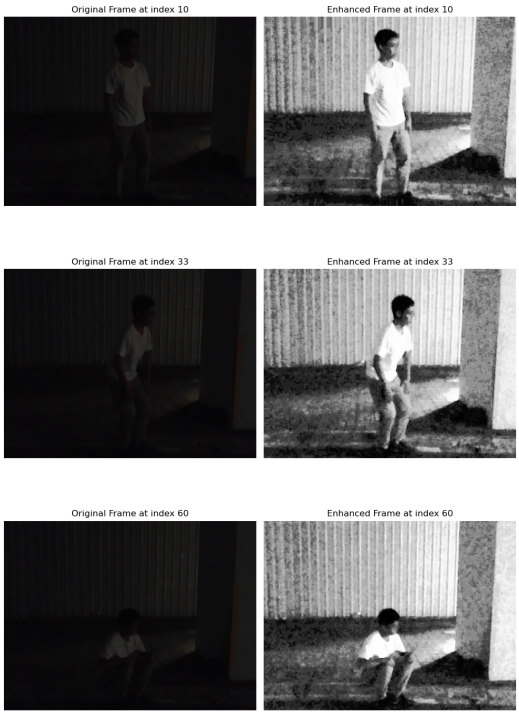

Figure 6: Enhanced Frames using HE from file sit_2_4.png

We can observe that for human perception, frames enhanced through processing techniques are significantly clearer, aiding in the recognition of human activities within the video. This clarity raises the question: does it similarly benefit machine analysis?

3.2 Results After Enhancement

3.2.1 Results and Analysis

For the preprocessing phase, we apply Histogram Equalization (HE) to enhance the frames. Subsequently, we retrain the classifier using these enhanced frames. The outcomes are displayed at the table 4 and the figure 7 below.

Table 4: The Evaluation of FC after Enhancement

Class	Precision	Recall	F1-Score	Support
Jump	0.22	0.29	0.25	17
Run	0.11	0.20	0.14	15
Sit	0.00	0.00	0.00	15
Stand	0.12	0.12	0.12	16
Turn	0.00	0.00	0.00	17
Walk	0.17	0.25	0.21	16
Accuracy			**0.15**	96
Macro Avg	0.10	0.14	0.12	96
Weighted Avg	0.10	0.15	0.12	96

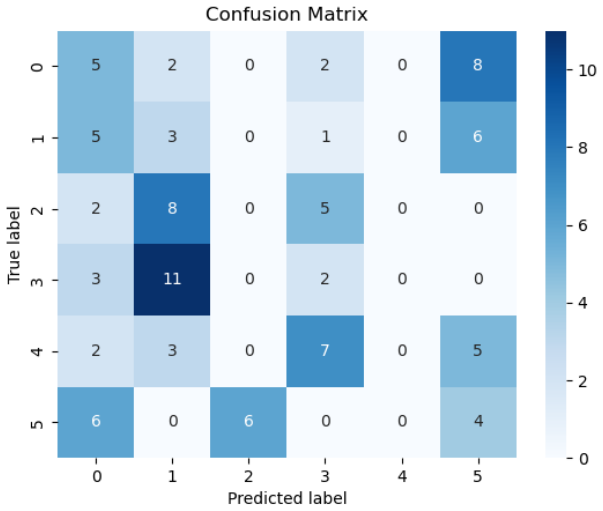

Figure 7: Confusion matrix of the FC classifier after HE

Contrary to expectations, the application of Histogram Equalization (HE) for image enhancement in the dataset has resulted in a paradoxical decrease in model performance. This counterintuitive outcome is particularly pronounced in the 'sit' and 'turn' action class, where despite the anticipated clarification of image features conducive to classification, the models exhibited a decline in accuracy.

3.2.2 The Reasons of Poor Performance After Enhancement

This could due to many reasons,

- **Insufficient Contrast Enhancement:** Although HE is capable of producing images with enhanced contrast, the action recognition accuracy actually diminished with the dataset. This suggests that merely increasing contrast is inadequate for enhancing performance, especially under complex action scenarios and low-light conditions.

- **Distributional Changes and Noise Introduction:** The visual clarity of video frames may improve post-HE application; however, such enhancements might be perceived as artificial effects or adversarial attacks on the video content. While enhanced frames appear clearer, they potentially disrupt the original distribution and introduce noise, which could degrade the performance of action recognition models.
- **Subjectivity of Image Quality:** While HE method does improve certain visual characteristics, these improvements do not necessarily translate into increased accuracy for action recognition. This indicates that current frame enhancement methods, though visually improving dark video frames, do not necessarily augment the accuracy of action recognition in dark videos.

3.2.3 Previous Work Support

In Xu et al. (2021)'s previous work, they analysis of frame enhancement techniques, such as Histogram Equalization (HE), LIME, BIMEF, KinD, and Guided Image Contrast Enhancement (GIC). Notably, the GIC method demonstrated a consistent increase in recognition accuracy across all evaluated models, attributed to its capacity to elevate the luminance of video frames. Conversely, other techniques, including KinD and HE, were less effective.

These findings not only prove the practice of enhancement in this study, but also suggest the potential for selective frame enhancement methods in HAR, warranting further investigation.

4 The End-to-end Network

End-to-end training is pivotal in the modern landscape of machine learning models for its direct approach and applicability in real-world scenarios. This section details the design and implementation of an end-to-end HAR model, moving away from the traditional method of explicitly storing video features.

4.1 Model Design

Our end-to-end network, which integrates features extraction and classification, is intuitively constructed by combining the previously developed components. The backbone of this model is a 2D+1D ResNet, wherein we have replaced the last fully connected (FC) layer to better suit our needs.

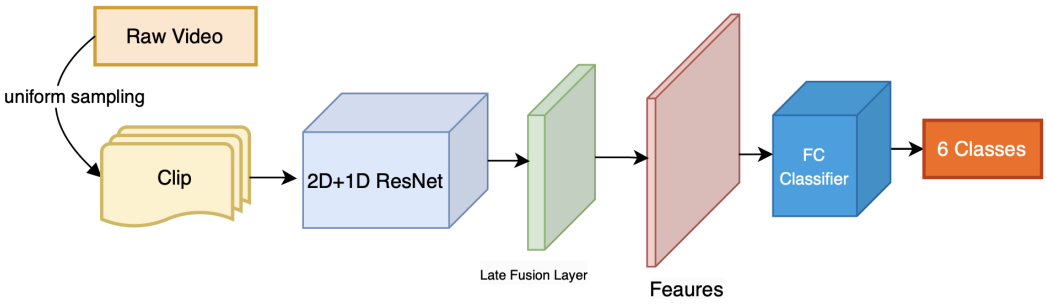

Figure 8: The end-to-end network working flow

The feature extraction process within this model is enhanced by incorporating uniform sampling prior to data input. This step ensures a more consistent and representative selection of video data for processing. Following the 2D+1D ResNet, we employ a late fusion technique, implemented as a max pooling layer. This layer effectively consolidates features extracted from different temporal segments of the video, thereby enhancing the model's ability to recognize actions over time.

For the classification segment, our network utilizes a structure composed of two linear layers. These are interspersed with a ReLU (Rectified Linear Unit) activation function, which introduces nonlinearity to the model, enabling it to learn more complex patterns in the data. Additionally, a dropout

layer is included to mitigate overfitting, a crucial aspect given the model's complexity and the potential variability in the video data.

4.2 Training Process

The end-to-end HAR model utilizes a modified 2D+1D ResNet architecture as its core. Key aspects of the training process include:

- **Data Preprocessing:** Frames are converted to RGB, resized, and normalized to ensure consistency across different video inputs.
- **Uniform Sampling:** A fixed number of frames are uniformly sampled from each video to provide a representative overview while managing computational resources.
- **Feature Extraction:** Features extracted from each frame are processed through late fusion, averaging them to effectively capture temporal information.
- **Classifier Configuration:** Comprises two linear layers, a ReLU activation function, and a 50% dropout layer to mitigate overfitting.
- **Optimization and Loss:** The Adam optimizer with a learning rate of 0.001 is used, alongside the CrossEntropyLoss function for loss computation.
- **Batch Processing and Epochs:** Training data is processed in batches of 8, and validation data in batches of 32, over 10 epochs.

4.3 Hyperparameter Tuning

The validation and hyperparameter tuning process is as follows:

- **Performance Metrics:** Loss and accuracy are computed during validation using the CrossEntropyLoss function and accuracy measurement.
- **Hyperparameter Tuning:** Learning rate, batch sizes and epoch sizes are adjusted based on the model's performance during validation.
- **Iterative Improvement:** The process allows for adjustments in model architecture, hyperparameters, or preprocessing steps based on observed performance.

4.4 Results and Evaluation

After undergoing a meticulous fine-tuning process, our model demonstrated a marked improvement in performance, effectively mitigating the risk of overfitting. This enhancement is evident in the performance metrics of our end-to-end model, as illustrated in the table below:

Table 5: The Evaluation of the End-to-End Model on Original Dark Frames

Class	**Precision**	**Recall**	**F1-Score**	**Support**
Jump	0.30	0.37	0.33	17
Run	0.25	0.34	0.29	15
Sit	0.00	0.00	0.00	15
Stand	0.22	0.22	0.22	16
Turn	0.00	0.00	0.00	17
Walk	0.27	0.35	0.31	16
Accuracy			**0.32**	96
Macro Avg	0.17	0.21	0.19	96
Weighted Avg	0.17	0.21	0.19	96

Our results, obtained at an epoch of 8 and a batch size of 8, indicate that the model outperforms previous iterations where feature extraction and classification were conducted separately. This is primarily attributed to the fine-tuning capabilities of our approach.

However, it's notable that the performance in the 'Sit' and 'Turn' classes remains suboptimal. This underperformance can be primarily ascribed to two factors. Firstly, the limitations of the Fully Connected (FC) and Support Vector Machine (SVM) classifiers in our dataset, which is both limited in size and diversity. Such constraints are particularly challenging when distinguishing subtle actions, especially in low-light video conditions. Enhancing the dataset by increasing its size and diversity, and incorporating data augmentation techniques, are potential strategies to address these challenges.

Secondly, the inherent nature of the dataset poses significant challenges. The minimal global pixel value changes in video frames induced by actions like 'Sit' and 'Turn' lead to difficulties in classification. These subtle movements are often misinterpreted as inconsequential, akin to background noise, rather than significant actions. Future improvements could focus on refining the model's sensitivity to subtle movements, possibly through advanced feature extraction techniques or more sophisticated classification algorithms.

4.5 Pros and Cons of the end-to-end Model

In evaluating the HAR model, it's important to consider both its advantages and limitations. Below is a summary of its key pros and cons:

Pros:

- **Improved Accuracy:** The model shows a significant improvement in accuracy, particularly in certain classes, after fine-tuning.
- **Advanced Feature Learning:** It is capable of extracting complex features from video data, making it highly effective for detailed activity recognition.
- **Scalability:** The model benefits from larger datasets and shows potential for improvement over time with more data.
- **Adaptability:** Can be fine-tuned and adjusted to specific requirements, offering flexibility in application.

Cons:

- **Computational Demand:** The model requires significant computational resources for training and fine-tuning, which might be a limiting factor in some scenarios.
- **Suboptimal in Subtle Movement Recognition:** It struggles with recognizing actions that induce minimal changes in video frames, such as 'Sit' and 'Turn'.
- **Data Dependency:** The performance of the model heavily relies on the quality and diversity of the training data.
- **Overfitting Risk:** While measures have been taken to mitigate this, deep learning models are inherently prone to overfitting without careful tuning.

5 Conclusion

In this study, we delve into Human Action Recognition (HAR) by methodically applying a deep learning-based approach. Our process begins with uniform sampling to extract video clips for feature analysis. For this purpose, we utilize a 2D+1D ResNet network as the foundational architecture. Following this, we implement late fusion techniques, strategically positioned before the classifiers, which comprise both Fully Connected (FC) classifiers and Support Vector Machines (SVM).

However, our initial results revealed suboptimal performance, which we attribute to two primary factors: the limited size and diversity of our dataset, and the inherent challenges of recognizing certain actions in low-light video frames.

Additionally, we explored various image enhancement techniques, using Histogram Stretching (HS) as a case study. Contrary to expectations, we found that these enhancement procedures did not significantly aid the network in improving its learning efficacy.

In response to these challenges, we proposed and evaluated an end-to-end network. Through meticulous fine-tuning, we observed a noticeable improvement in performance. This outcome highlights the potential of deep learning methodologies in advancing the field of HAR, particularly when optimized for the complexities of low-light environments.

REFERENCES

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. Large-scale video classification with convolutional neural networks. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1725–1732, 2014. doi: 10.1109/CVPR.2014.223.

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 (CVPR)*, June 2018.

Yuecong Xu, Jianfei Yang, Haozhi Cao, Kezhi Mao, Jianxiong Yin, and Simon See. Arid: A new dataset for recognizing action in the dark. In Xiaoli Li, Min Wu, Zhenghua Chen, and Le Zhang (eds.), *Deep Learning for Human Activity Recognition*. Springer Singapore, 2021.

A APPENDIX

A.1 FEATURE EXTRACTION

```
import torch
import torch.nn as nn
from torchvision.models.video import r2plus1d_18
import torchvision.transforms as transforms
import cv2
import os
from tqdm import tqdm
from PIL import Image

class R2Plus1DFeatureExtractor(nn.Module):
    def __init__(self):
        super(R2Plus1DFeatureExtractor, self).__init__()
        original_model = r2plus1d_18(pretrained=False)
        # remove the last classification layer in order to extract
        ↪  features
        self.feature_extractor =
        ↪  nn.Sequential(*list(original_model.children())[:-1])

    def forward(self, x):
        print(x.shape)
        x = self.feature_extractor(x)
        # late fusion
        x = torch.mean(x, dim=2)  # average features across time
        return x

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
feature_extractor = R2Plus1DFeatureExtractor().to(device)
feature_extractor.eval()

normalize = transforms.Normalize(mean=[0.07, 0.07, 0.07], std=[0.1,
↪  0.09, 0.08])
preprocess = transforms.Compose([
```

```
    transforms.Resize((112, 112)),
    transforms.ToTensor(),
    normalize
])

from PIL import Image

def extract_features(frames):
    processed_frames = [preprocess(Image.fromarray(cv2.cvtColor(frame,
    ↪  cv2.COLOR_BGR2RGB))) for frame in frames]
    video_tensor =
    ↪  torch.stack(processed_frames).unsqueeze(0).to(device)  #  (1,
    ↪  T, C, H, W)
    video_tensor = video_tensor.permute(0, 2, 1, 3, 4)  #  (1, C, T, H,
    ↪  W)
    with torch.no_grad():
        features = feature_extractor(video_tensor)
    return features

def late_fusion_average(features):
    return torch.mean(features, dim=0).squeeze(0)

def uniform_sampling(video_path, num_frames):
    cap = cv2.VideoCapture(video_path)
    total_frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
    sampling_interval = max(1, total_frames // num_frames)

    frames = []
    for i in range(0, total_frames, sampling_interval):
        cap.set(cv2.CAP_PROP_POS_FRAMES, i)
        ret, frame = cap.read()
        if ret:
            frames.append(frame)
            if len(frames) == num_frames:
                break

    cap.release()
    return frames

def process_videos(file_path, base_dir, desc):
    features = []
    labels = []

    with open(file_path, 'r') as file:
        for line in tqdm(file, desc=desc):
            index, label, video_rel_path = line.strip().split()
            video_path = os.path.join(base_dir, video_rel_path)

            sampled_frames = uniform_sampling(video_path,
            ↪  num_frames_to_sample)
            video_features = extract_features(sampled_frames)
            averaged_features = late_fusion_average(video_features)

            features.append(averaged_features)
            labels.append(int(label))

    return torch.stack(features), torch.tensor(labels)

num_frames_to_sample = 15  # number of frames to sample from each video
num_classes = 6

# Process videos
train_features, train_labels = process_videos('data/train.txt',
↪  'data/train', "Processing Train Videos")
```

```
val_features, val_labels = process_videos('data/validate.txt',
↪  'data/validate', "Processing Validation Videos")

# Save features and labels
torch.save(train_features, 'train_features.pt')
torch.save(train_labels, 'train_labels.pt')
torch.save(val_features, 'val_features.pt')
torch.save(val_labels, 'val_labels.pt')
```

A.2 FC CLASSIFIER

```
from sklearn.metrics import confusion_matrix, classification_report
import torch
import torch.nn as nn
import torch.optim as optim
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
import random

seed = 0
torch.manual_seed(seed)
np.random.seed(seed)
random.seed(seed)
if torch.cuda.is_available():
    torch.cuda.manual_seed(seed)
    torch.cuda.manual_seed_all(seed)
    torch.backends.cudnn.deterministic = True
    torch.backends.cudnn.benchmark = False

class Classifier(nn.Module):
    def __init__(self, input_size, num_classes):
        super(Classifier, self).__init__()
        self.flatten = nn.Flatten()
        self.fc = nn.Linear(input_size, num_classes)

    def forward(self, x):
        x = self.flatten(x)
        out = self.fc(x)
        return out


# Load features and labels
train_features = torch.load('train_features.pt')
train_labels = torch.load('train_labels.pt')

# Load validation features and labels
val_features = torch.load('val_features.pt')
val_labels = torch.load('val_labels.pt')

# Create the network
input_size = train_features.size(1)
hidden_size = 256
num_classes = torch.max(train_labels) + 1
net = Classifier(input_size, num_classes)

torch.manual_seed(0)
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

net = net.to(device)
train_features = train_features.to(device)
```

```
train_labels = train_labels.to(device)

val_features = val_features.to(device)
val_labels = val_labels.to(device)

# Loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(net.parameters(), lr=0.001,
↪  weight_decay=0.01)

train_losses = []
val_losses = []
train_accuracies = []
val_accuracies = []

# Training loop
for epoch in range(100):
    net.train()
    # Forward pass
    outputs = net(train_features)
    loss = criterion(outputs, train_labels)
    train_losses.append(loss.item())

    # Calculate training accuracy
    _, train_predicted = torch.max(outputs, 1)
    train_accuracy = (train_predicted == train_labels).float().mean()
    train_accuracies.append(train_accuracy.item())

    # Backward and optimize
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    # Validation
    net.eval()  # Set the model to evaluation mode
    val_outputs = net(val_features)
    val_loss = criterion(val_outputs, val_labels)
    val_losses.append(val_loss.item())

    # Calculate validation accuracy
    _, val_predicted = torch.max(val_outputs, 1)
    val_accuracy = (val_predicted == val_labels).float().mean()
    val_accuracies.append(val_accuracy.item())

    print(f'Epoch {epoch+1}, Train Loss: {loss.item()}, Validation
    ↪  Loss: {val_loss.item()}')

# After training, plot the training and validation losses
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(train_losses, label='Training Loss')
plt.plot(val_losses, label='Validation Loss')
plt.title('Training and Validation Loss')
plt.legend()

# Plot the training and validation accuracies
plt.subplot(1, 2, 2)
plt.plot(train_accuracies, label='Training Accuracy')
plt.plot(val_accuracies, label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.legend()

plt.tight_layout()
plt.savefig('pics/nn_loss_acc.png')
```

```
plt.close()

# Evaluate the model using validation set
net.eval()
with torch.no_grad():
    val_outputs = net(val_features)
    _, predicted = torch.max(val_outputs.data, 1)
    total = val_labels.size(0)
    correct = (predicted == val_labels).sum().item()
    print(f'Validation Accuracy: {100 * correct / total}%')

val_labels_cpu = val_labels.cpu().numpy()
predicted_cpu = predicted.cpu().numpy()

# confusion matrix
conf_mat = confusion_matrix(val_labels_cpu, predicted_cpu)
plt.figure()
sns.heatmap(conf_mat, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix')
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.savefig('pics/nn_evalu.png')
plt.close()

# report
report = classification_report(val_labels_cpu, predicted_cpu)
with open('results/nn/report.txt', 'w') as file:
    file.write(report)
print(report)
```

A.3 SVM Classifier

```
import torch
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix,
↪  roc_curve, auc
import matplotlib.pyplot as plt
import seaborn as sns

def to_cpu(tensor):
    return tensor.to('cpu') if tensor.is_cuda else tensor

train_features = torch.load('train_features.pt')
train_labels = torch.load('train_labels.pt')
val_features = torch.load('val_features.pt')
val_labels = torch.load('val_labels.pt')

train_features = to_cpu(train_features).reshape(150, -1)
train_labels = to_cpu(train_labels)
val_features = to_cpu(val_features).reshape(96, -1)
val_labels = to_cpu(val_labels)

svm = SVC()

svm.fit(train_features.numpy(), train_labels.numpy())

train_preds = svm.predict(train_features.numpy())
val_preds = svm.predict(val_features.numpy())

# evaluation
conf_mat = confusion_matrix(val_labels.numpy(), val_preds)
```

```
sns.heatmap(conf_mat, annot=True, fmt='d')
plt.savefig('pics/svm_evalu.png')

print(classification_report(val_labels.numpy(), val_preds))
```

A.4 Image Enhancement Part

```
# Step 4 Normalization
normalize = transforms.Normalize(mean=[0.485, 0.456, 0.406],
↪  std=[0.229, 0.224, 0.225])

def enhance_image_histogram_equalization(image):
    # Convert to grayscale for histogram equalization
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # Apply histogram equalization
    enhanced = cv2.equalizeHist(gray)

    # Convert back to BGR
    enhanced_bgr = cv2.cvtColor(enhanced, cv2.COLOR_GRAY2BGR)
    return enhanced_bgr
```