HUMAN ACTION RECOGNITION IN THE DARK

Zuo Jia
JZUO001@e.ntu.edu.sg
School of Electrical and Electrical Engineering
Nanyang Technological University

ABSTRACT

Human action recognition is a highly significant research area. Recognition under dark conditions often becomes a topic of considerable research value due to issues such as insufficient datasets and excessive noise.

This study conducted an investigation into each influencing factor separately. I compared two different video frame sampling methods: random sampling and uniform sampling. Feature extraction was performed using a ResNet50 pretrained model. Then I evaluated three types of classifiers: SVC (Support Vector Classifier), RandomForest, and a fully connected neural network, and compared different learning rate schedulers. Image enhancement was applied through gamma correction to explore the influence of these factors on the effectiveness of video classification for human action recognition (HAR) in dark environments.

Moreover, an end-to-end model was utilized for categorization. Given the small dataset size and the challenges presented by dark environments, a model with a **CSN** (Channel-Separated Convolutional Networks) backbone, pretrained on IG-65M (Ultra Large Scale Network Video), was selected. Enhancements to the dataset included gamma correction, random flipping, and random cropping, among others. The study also compared the effects of different learning rate schedulers and model architectures on the outcomes. Ultimately, the model achieved a classification accuracy on the test set with a top-1 accuracy of **67.71%** and a top-5 accuracy of **98.96%**.

1 INTRODUCTION

Human activity recognition (HAR) in dark environments is an important research direction with broad application prospects. It has significant research value for security systems, autonomous driving, smart homes, and more - for example, detecting anomalous behaviors at night. HAR is a challenging problem due to factors like high noise in dark environment data and existing models being trained on datasets with good lighting conditions. Tackling the unique obstacles posed by poor illumination will enable HAR systems to work reliably in real-world settings. Advancing this field can lead to innovations that make technology safer, more perceptive, and more useful across many domains.

To video classification, we must first sample the videos. Since most pretrained models are trained on datasets with good lighting conditions, it is necessary to enhance the images before using such models. After enhancing the images and extracting features, classifiers are then employed for categorization.

Since the approach of first extracting features and then employing classifiers for categorization tends to discard many features, resulting in suboptimal classification performance, I subsequently adopted a direct end-to-end model for video classification.

2 RELATED WORKS

2.1 3D SPATIOTEMPORAL CONVOLUTIONS(3D-CNN)

3D CNNs extend traditional 2D CNN architectures to handle the temporal dimension in video data, enabling the capture of spatial features and motion information over time. 3D CNNs analyze changes along the time axis by applying 3D convolutions across consecutive frames, thereby achieving action feature extraction.

Convolutional 3D (C3D) Tran et al. (2015) is a foundational model that applies 3D convolutions over spatial and temporal dimensions for video feature extraction. Inflated 3D CNNs (I3D) Carreira & Zisserman (2017) expand the filters and pooling kernels of a CNN to 3D, allowing the use of successful ImageNet architectures and their pretrained models.

ResNet-3D Hara et al. (2018) adapted the ResNet architecture with 3D convolutions, designed for depth and residual learning. Channel-Separated Convolutional Networks (CSN) Tran et al. (2019) separate channel interactions from temporal interactions to reduce complexity. X3D Feichtenhofer (2020) can efficiently scale different dimensions of the model.

2.2 VIDEO TRANSFORMERS(VIT)

Vision Transformers (ViTs) have revolutionized the field of computer vision with their unique approach to processing images and videos.

Introduced by Dosovitskiy et al. (2020), ViT firstly applies the transformer architecture, traditionally used in NLP, to image classification tasks. Developed to improve the data efficiency of ViTs, DeiT (Data-efficient Image Transformer) Touvron et al. (2021) incorporates a teacher-student strategy known as distillation to train the transformer on smaller datasets. CrossViT (Cross-Attention Multi-Scale Vision Transformer) Chen et al. (2021) integrates dual-branch transformers to process images at multiple scales, utilizing cross-attention for information exchange between branches.

Timesformer Bertasius et al. (2021) adapts the Vision Transformer to video by applying a spatio-temporal self-attention mechanism, enabling the model to consider both spatial and temporal features within the video data. It stands out for its capacity to handle long-range interactions across both space and time, making it particularly effective for video classification tasks.

3 METHODOLOGY

In this section, I will elucidate the theoretical underpinnings of the methods that will be utilized in this research. This includes two video sampling techniques, gamma correction, SVC and random forest classifiers, as well as the end-to-end CSN model used later.

3.1 VIDEO FRAMES SAMPLING

3.1.1 RANDOM SAMPLING

Random sampling, on the other hand, is a method that selects video frames at random, without following a fixed temporal interval. This method is particularly useful when dealing with data that is not uniformly distributed over time or when it is desirable for the model to be insensitive to the exact timing of events.

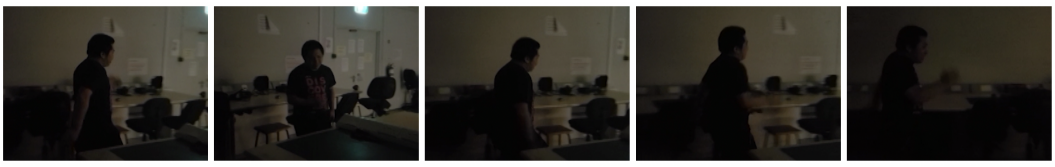


Figure 1: Example of Uniform Sampling

3.1.2 UNIFORM SAMPLING

Uniform sampling is a method that extracts frames from a video at fixed temporal intervals. Given a video sequence, uniform sampling selects frames from the start to the end of the video at equal intervals. This approach ensures that information is extracted uniformly across the entire video and is typically used to ensure coverage of the full temporal span of the video.

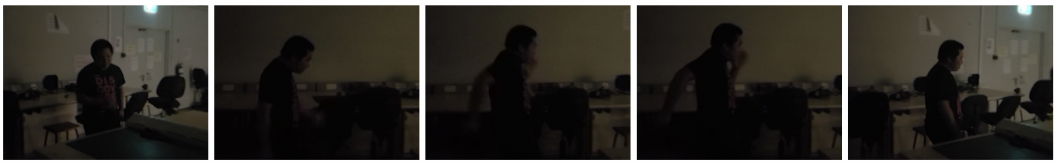


Figure 2: Example of Uniform Sampling

Uniform sampling offers consistent temporal coverage, while random sampling provides greater diversity and potential robustness. In this study, **uniform sampling** was chosen due to the dataset's sensitivity to temporal sequences. For instance, "sit" and "stand" represent two opposing processes, while "run" and "walk" are more sensitive to the frequency of the motion over time.

I have employed the method of uniform sampling on all videos, storing the sampled frames as images. This ensures that it is not necessary to resample every time the code is executed.

3.2 GAMMA CORRECTION

Gamma Correction is a non-linear encoding method that enhances human perception of brightness and optimizes encoding space. The gamma function is expressed as:

$$C = I^{\gamma} \tag{1}$$

where γ is a constant. When $\gamma < 1$, highlights are compressed, and revealing more shadow details. The overall image appears darker. When $\gamma > 1$, highlights are expanded, and shadows are compressed, resulting in a brighter overall image.

Figure 3: Example of Gamma Correction with $\gamma = 3$

In this study, since the pre-trained models used were trained on datasets with good lighting conditions, it is necessary to perform gamma correction on our dataset to match the brightness level to that of the pre-trained model's dataset for effective feature extraction.

3.3 CLASSIFIERS

3.3.1 SUPPORT VECTOR CLASSIFIER (SVC)

The Support Vector Machine (SVM) Cortes & Vapnik (1995) is a supervised learning model extensively utilized for classification and regression tasks. The fundamental concept of SVM is to find an optimal decision boundary that maximizes the distance to the nearest points of the categories.

The objective of SVM is to minimize the following objective function:

$$\min_{w,b} \frac{1}{2}||w||^2 + C\sum_{i=1}^{n}\xi_i \tag{2}$$

$$y_i(w \cdot x_i + b) \geq 1 - \xi_i, \xi_i \geq 0, i = 1, ..., n \tag{3}$$

where C is the regularization parameter, which controls the trade-off between the margin width and classification error, ξ_i is the slack variable, and y_i is the class label for the i-th data point.

3.3.2 RANDOM FOREST CLASSIFIER

Random Forest (RF) Breiman (2001) is an ensemble learning method applicable for classification tasks. It operates by constructing multiple decision trees and aggregating their predictions to enhance the predictive accuracy and stability of the overall model.

Initially, N samples are selected from the original training dataset using bootstrapped random sampling with replacement. Subsequently, decision trees are built by seeking the best split points on random subsets of features, thus forming a forest.

In the end, each tree delivers its prediction result, and the final classification outcome is determined by majority voting.

3.4 CHANNEL-SEPARATED CONVOLUTIONAL NETWORKS (CSN)

CSN Tran et al. (2019) is a kind of video classification model based on 3D-CNN. The main idea of this network is to utilize separated convolution operations to reduce computational complexity while capturing temporal and spatial information.

Firstly, spatial Convolutions are applied independently on each frame, processing spatial information to capture intra-frame features. Temporal Convolutions span across multiple time frames to capture temporal dynamics and motion information. Fusion Layers integrate spatial and temporal features so that the network can consider both spatial positions and temporal dynamics simultaneously.

CSN employs group convolution, where input and output channels are divided into multiple groups, with each group performing convolution operations independently. This reduces the number of model parameters and computational cost.

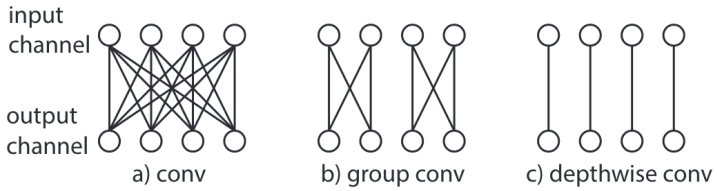


Figure 4: Enter Caption

Finally, a fully connected layers or other classifiers are used to make the ultimate classification decision based on the extracted features.

CSN is particularly suitable for small-scale datasets in dark environments. It use channel-separated convolutions which can efficiently manage the model's capacity with fewer parameters compared to standard 3D convolutions. This is beneficial for small-scale datasets where there's a limited amount of training data to learn from, as it reduces the risk of overfitting.

4 EXPERIMENT WITH SEPARATED MODULES

In this section, I conducted comparative experiments on each module to investigate the impact of different modules on classification performance.

4.1 EXPERIMENTAL SETUP

The dataset employed consists of 150 training videos and 96 test videos, spanning a total of 6 classes. Experiments were conducted using PyTorch 2.1.1 on a Xeon® E5-2620 v4 CPU with a single GPU NVIDIA RTX A4000.

The total processor is shown as follows:

(1) Firstly, I performed uniform sampling and random sampling respectively on the dataset. Given the relatively small size of the dataset, I chose a sampling interval of 2 to ensure that more information could be included.

(2) I performed gamma correction on the images after sampling. Apart from the experimental section comparing the effects of gamma correction, $\gamma = 3$ was used in all other experiments.

(3) After that, I used a pre-trained model ResNet50 in torchvision to extract the features of videos. To save time and avoid the need to re-extract features with each code execution, I saved the extracted features for subsequent use.

(4) Subsequently, I fed the extracted features from the training set into the classifier for training. Then, the trained classifier model was used to predict the classes of the test set.

(5) Finally, output the classification accuracy on the test set, along with a visualization of the loss reduction curve during the training process.

Generally, I choose uniform sampling, $\gamma = 3$ to enhance the images to match the pre-trained model ResNet50, with a simple neural network classifier.

4.2 Performance with Two Sampling Methods

With all other experimental modules being constant, a comparison was conducted between uniform sampling and random sampling to determine their effects on classification accuracy, as shown in the table.

Table 1: Comparison of Performance with Uniform Sampling and Radom Sampling

Sampling Method	Accuracy of Top 1 on Test Sets	Accuracy of Top 1 on Train Sets
Uniform Sampling	**25.00%**	94.67%
Radom Sampling	23.95%	94.00%

It was observed that the accuracy of uniform sampling is marginally higher than that of random sampling. This is because random sampling disrupts the temporal information of the original videos, potentially leading to confusion between videos that have similar static features.

4.3 Performance with Gamma Correction

With all other experimental modules being constant, a comparison was conducted between $\gamma = 1$ and $\gamma = 3$ to determine their effects on classification accuracy, as shown in the table.

Table 2: Comparison of Performance with $\gamma = 1$ and $\gamma = 3$

γ	Accuracy of Top 1 on Test Sets	Accuracy of Top 1 on Train Sets
$\gamma = 3$	**25.00%**	94.67%
$\gamma = 1$	19.79%	94.67%

It can be observed that gamma correction has a substantial impact on the video classification results. This is explainable given that the pre-trained model, ResNet50, used for feature extraction, was predominantly trained on datasets with good lighting conditions. It requires that our video mean and standard deviation be similar to the reference dataset's mean and standard deviation. When gamma is set to 3, the video more closely aligns with this reference data.

4.4 Performance with Different Classifiers

With all other experimental modules being constant, I used SVM, Random Forest and NN classifiers respectively to determine their effects on classification accuracy, as shown in the table.

Data above indicates that classifier based on Neural Networks outperform Random Forest classifier, which in turn are superior to SVM classifier.

Table 3: Comparison of Performance with Three Classifiers

Classifier	Accuracy of Top 1 on Test Sets	Accuracy of Top 1 on Train Sets
SVM	16.67%	100.00%
Random Forest	17.71%	100.00%
Neural Network	**25.00%**	94.67%

5 EXPERIMENT WITH END-TO-END MODEL

This chapter employs an end-to-end model based on **CSN** Tran et al. (2019) , modifies the model, conducts comparative experiments, adjusts parameters, and ultimately visualizes the results.

5.1 MODEL SELECTION

I used the open-source models from **MMAction2** Contributors (2020) , which include action recognition models such as C2D, I3D, CSN, and TimeSformer etc.

Table 4: Performance of Different Models in Kinetics-400 dataset

Model	Backbone	Params	Pretrain	Top-1 Acc
C2D	ResNet50	24.3M	ImageNet	73.44%
I3D	ResNet50	35.4M	ImageNet	74.80%
CSN	ResNet50	13.13M	IG65M	79.44%
CSN	ResNet152	29.70M	IG65M	**82.84%**
TimeSformer	TimeSformer	122M	ImageNet-21K	77.69%

Datas above comes from results in MMAction2.

The performance of these models on video classification in the Kinetics-400 dataset is shown in the table. Considering that the videos in the dataset are relatively short, and TimeSformer is mainly used for long temporal sequence video processing, it was not chosen for this study.

Since CSN employs channel-separated convolutions, it has fewer parameters, enabling faster training with less computational resources. Additionally, given that the regularization methods in CSN are beneficial for handling small-scale datasets. Therefore, I selected a model with **CSN** based on **ResNet152** as its backbone with **IG-65M** (65 million Instagram videos) pre-trained weights and structure Ghadiyaram et al. (2019) .

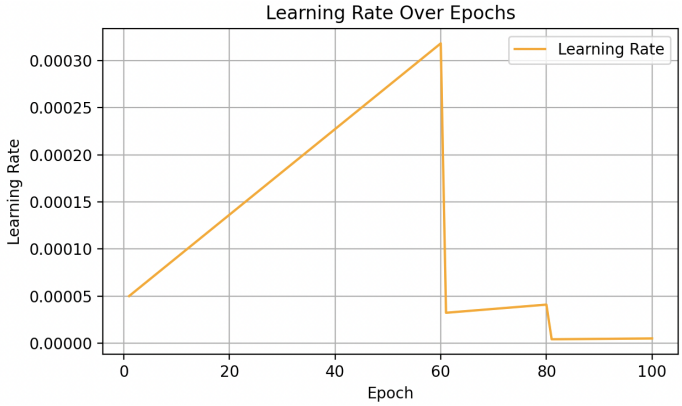
5.2 EXPERIMENTAL SETUP

The dataset employed consists of 150 training videos and 96 test videos, spanning a total of 6 classes. Experiments were conducted using PyTorch 2.1.1 on a Xeon® E5-2620 v4 CPU with a single GPU NVIDIA RTX A4000.

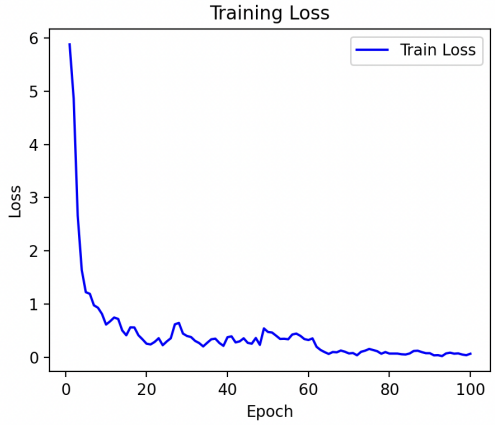
I trained the model on GPU with cuda 12.1, with batch size of 12. Depth of the model backbone is 152, based on the ResNet50 and ResNet152 pre-trained weights respectively. To prevent overfitting in the model, I employed a regularization technique known as dropout, with a rate of 0.5, to enhance the model's generalization ability.

Total number of epochs is 100. Every 5 epochs, the current model is evaluated for accuracy using the validation set, and the model with the highest accuracy at that point is saved.

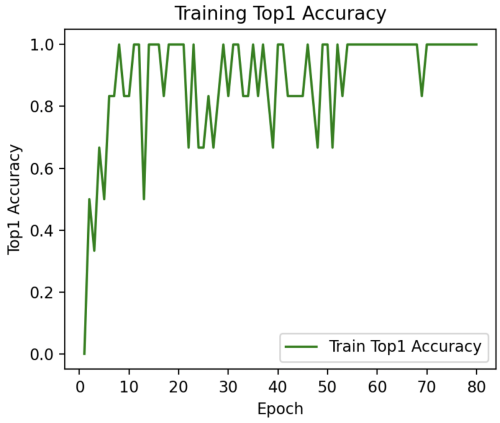
To optimize the training of the model, the Stochastic Gradient Descent (SGD) optimizer is used with an initial learning rate of 5e-4. The learning rate scheduler starts with a LinearLR to increase the learning rate linearly from a smaller initial value, allowing the model to converge quickly in the early stages. Subsequently, the learning rate scheduler switches to MultiStepLR, which reduces the learning rate at epochs 60 and 80, respectively, to allow for fine-tuning of the model in the later stages. The variation curve of learning rate and corresponding training loss and accuracy are shown in Fig.5.



(a) Learning Rate Variation Curve



(b) Training Loss Variation Curve



(c) Training Accuracy Variation Curve

Figure 5: Variation Curve of Learning Rate, Training Loss and Accuracy

5.3 Experiment Result

5.3.1 Comparison of ResNet50 and ResNet152

Firstly, with other parts fixed, I compared model with the two different ResNet50 and ResNet152 pre-trained weights. Results can be shown as follows:

Table 5: Results with ResNet50 and ResNet152 on Test Sets

Pre-trained Weight	Top-1 Acc on Test	Top-5 Acc on Test
ResNet50	56.25%	98.96%
ResNet152	**67.71%**	98.96%

We can easily find that model with ResNet152 pre-trained weight performs better. Because more parameters are included with more training data and more complete features can be obtained.

5.3.2 Impact of Gamma Correction

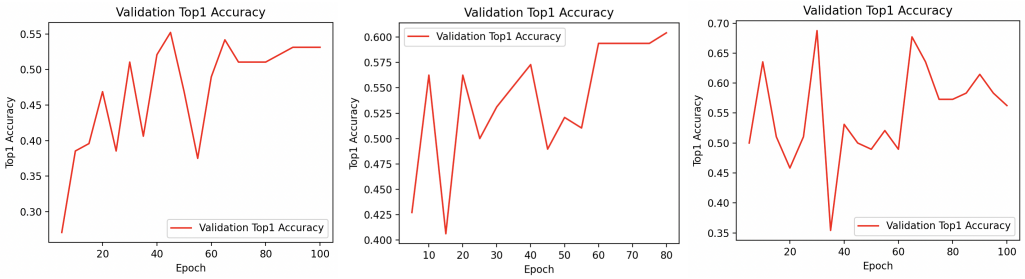
I modified the structure of the model to include gamma correction code in the dataset pipeline section immediately following the sampling process.

To investigate the impact of gamma correction on the model's classification accuracy, I selected gamma values of 1 (no gamma correction), 3, and 5, keeping all other parameters constant. The comparative experimental results are as follows in the table:

Table 6: Results with Different Gamma on Test Sets

Gamma	Top-1 Acc on Test	Top-5 Acc on Test
1	58.33%	100.00%
3	61.46%	97.92%
5	**67.71%**	98.96%

The experimental results reveal that the model achieves the highest Top-1 accuracy of **67.71%** when gamma is set to 5. With gamma set to 3 or without gamma correction, the accuracies are 61.46% and 58.33%, respectively. This demonstrates the importance of gamma correction for video classification under dark conditions. Since the dataset used to pre-train the selected model was mostly under very good lighting conditions, it is necessary to process our dataset with gamma correction to bring it to a level similar to the pre-trained dataset.



(a) Val Top-1 Acc with $\gamma = 1$ (b) Val Top-1 Acc with $\gamma = 3$ (c) Val Top-1 Acc with $\gamma = 5$

Figure 6: Validation Top-1 Acc with $\gamma = 1, 3, 5$

6 Conclusion

The first part of this project examines the impact of different sampling methods, image enhancement techniques, pre-trained feature extraction models, and classifiers on the classification effectiveness of human action videos under dark conditions. Based on the experimental data, the optimal module configuration for this dataset includes: uniform sampling, image enhancement with gamma set to 3, feature extraction using the pre-trained ResNet50 model, and a neural network classifier. This setup can achieve a classification Top-1 accuracy of **25%** on test datasets.

The second part of this project employs an end-to-end model for video classification. Based on a CSN backbone and pre-trained weights from IG65M, combined with gamma correction set to 5, as well as essential regularization techniques and learning rate curve optimization methods, it ultimately achieves a classification **Top-1** accuracy of **67.71%** on **test** datasets.

REFERENCES

Gedas Bertasius, Heng Wang, and Lorenzo Torresani. Is space-time attention all you need for video understanding? In *ICML*, volume 2, pp. 4, 2021.

Leo Breiman. Random forests. *Machine learning*, 45:5–32, 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.

Chun-Fu Richard Chen, Quanfu Fan, and Rameswar Panda. Crossvit: Cross-attention multi-scale vision transformer for image classification. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 357–366, 2021.

MMAction2 Contributors. Openmmlab's next generation video understanding toolbox and benchmark. `https://github.com/open-mmlab/mmaction2`, 2020.

Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20:273–297, 1995.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

Christoph Feichtenhofer. X3d: Expanding architectures for efficient video recognition. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 203–213, 2020.

Deepti Ghadiyaram, Du Tran, and Dhruv Mahajan. Large-scale weakly-supervised pre-training for video action recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 12046–12055, 2019.

Kensho Hara, Hirokatsu Kataoka, and Yutaka Satoh. Can spatiotemporal 3d cnns retrace the history of 2d cnns and imagenet? In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 6546–6555, 2018.

Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. In *International conference on machine learning*, pp. 10347–10357. PMLR, 2021.

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, and Matt Feiszli. Video classification with channel-separated convolutional networks. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 5552–5561, 2019.

A APPENDIX

```
# main of first part
import os
import cv2
import numpy as np
import random
import torch
import torch.nn
import torch.optim as optim
from torch.utils.data import DataLoader, Dataset
from torchvision.models import resnet50, ResNet50_Weights
from torchvision.transforms import transforms
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from tqdm import tqdm
from nn_classifier import classifier_train, clf_with_model_inference
from sampling import uniform_frame_sampling

train_video_paths = '/tmp/pycharm_project_zzj/dataset/'
test_video_paths = '/tmp/pycharm_project_zzj/dataset/'

# after sampling
train_sample_paths = '/tmp/pycharm_project_zzj/sampled/'
test_sample_paths = '/tmp/pycharm_project_zzj/sampled/'

train_labels = read_labels('/tmp/pycharm_project_zzj/sampled/
    updated_train_ann.txt')
test_labels = read_labels('/tmp/pycharm_project_zzj/sampled/
    updated_val_ann.txt')

uniform_frame_sampling(train_video_paths,train_sample_paths,2)
uniform_frame_sampling(train_video_paths,test_sample_paths,2)

# gamma correction
def adjust_gamma(image, gamma=1.0):
    invGamma = 1.0 / gamma
    table = np.array([((i / 255.0) ** invGamma) * 255
                      for i in np.arange(0, 256)]).astype("uint8")

    return cv2.LUT(image, table)

# SVC classifier
def train_classifier_SVC(X_train, y_train):
    clf = SVC(kernel='linear')
    clf.fit(X_train, y_train)
    return clf

# random forest classifier
def train_classifier_RandomForest(X_train, y_train):
    clf = RandomForestClassifier(n_estimators=100, random_state=42)
    clf.fit(X_train, y_train)
    return clf

# evaluate classifier
def evaluate_classifier(clf, X_test, y_test):
    y_pred = clf.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    return accuracy

def read_labels(label_path):
    labels = {}
    with open(label_path, 'r') as f:
        for line in f:
```

```
            video_id, frame_num, label = line.strip().split()
            labels[video_id] = int(label)
    return labels

# save features
def save_features(features, filename):
    os.makedirs(os.path.dirname(filename), exist_ok=True)
    np.save(filename, features)

# load features
def load_features(filename):
    return np.load(filename)

# features extract
def extract_features(frame_paths, device):
    model = resnet50(weights=ResNet50_Weights.IMAGENET1K_V1).to(device)
    model.eval()
    transform = transforms.Compose([
        transforms.ToPILImage(),
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229,
            0.224, 0.225]),
    ])
    features = []
    with torch.no_grad():
        for frame_path in frame_paths:
            image = cv2.imread(frame_path)
            if image is None:
                print(f"Could not read image from path: {frame_path}")
                continue
            image = adjust_gamma(image, gamma=3.0)
            image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
            image = transform(image).unsqueeze(0).to(device)
            feature = model(image).squeeze(0)
            features.append(feature)
    features = torch.stack(features)
    return features.mean(dim=0)

# load and extract features
def load_and_extract_features(sample_paths, labels, device):
    X, y = [], []
    for video_id in tqdm(sorted(labels.keys()), desc='Processing videos',
         unit='video'):
        label = labels[video_id]
        feature_path = f'/tmp/pycharm_project_zzj/features_gamma/{
            video_id}.npy'
        if os.path.exists(feature_path):
            video_feature = load_features(feature_path)
        else:
            frame_dir = os.path.join(sample_paths, video_id)
            frame_paths = [os.path.join(frame_dir, frame) for frame in
                sorted(os.listdir(frame_dir))]
            video_feature = extract_features(frame_paths, device).cpu().
                numpy()
            save_features(video_feature, feature_path)
        X.append(video_feature)
        y.append(label)
    return np.array(X), np.array(y)

# main
def zzj_run_main():
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    print(f"Using device: {device}")
```

```
    print('Start load and extract features of train')
    X_train, y_train = load_and_extract_features(train_sample_paths,
        train_labels, device)

    print('Train ...')
    # model_path = f'/tmp/pycharm_project_zzj/classifier/first_model.pth'
    # clf = classifier_train(X_train, y_train, model_path, device)
    clf = train_classifier_SVC(X_train, y_train)

    print('Start load and extract features of test')
    X_test, y_test = load_and_extract_features(test_sample_paths,
        test_labels, device)

    print('Predict ...')
    # y_pred_train = clf_with_model_inference(X_train, y_train, clf,
        device)
    # y_pred = clf_with_model_inference(X_test, y_test, clf, device)
    y_pred_train = clf.predict(X_train)
    y_pred = clf.predict(X_test)
    train_accuracy = accuracy_score(y_train, y_pred_train)
    test_accuracy = accuracy_score(y_test, y_pred)
    print(f'Train Accuracy: {train_accuracy}')
    print(f'Test Accuracy: {test_accuracy}')

# ------------------------------------------------------------------ #
# sampling.py
import cv2
import numpy as np
import matplotlib.pyplot as plt
import random
import os
from PIL import Image

def random_frame_sampling(video_path, sample_path, num_frames=5):
    video_files = [f for f in os.listdir(video_path) if f.endswith('.mp4'
        )]

    for video_file in video_files:
        video_file_path = os.path.join(video_path, video_file)
        cap = cv2.VideoCapture(video_file_path)
        total_frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
        frames_to_sample = sorted(random.sample(range(total_frames),
            num_frames))

        video_sample_path = os.path.join(sample_path, os.path.splitext(
            video_file)[0])
        if not os.path.exists(video_sample_path):
            os.makedirs(video_sample_path)

        for count, i in enumerate(frames_to_sample):
            cap.set(cv2.CAP_PROP_POS_FRAMES, i)
            ret, frame = cap.read()
            if ret:
                frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
                frame_file = os.path.join(video_sample_path, f"frame_{
                    count}.png")
                # save frames
                cv2.imwrite(frame_file, cv2.cvtColor(frame, cv2.
                    COLOR_RGB2BGR))

        cap.release()

def uniform_frame_sampling(video_path, sample_path, sample_interval=5):
```

```
    video_files = [f for f in os.listdir(video_path) if f.endswith('.mp4')]

    for video_file in video_files:
        video_file_path = os.path.join(video_path, video_file)
        cap = cv2.VideoCapture(video_file_path)
        total_frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))

        video_sample_path = os.path.join(sample_path, os.path.splitext(video_file)[0])
        if not os.path.exists(video_sample_path):
            os.makedirs(video_sample_path)

        frame_count = 0
        while frame_count < total_frames:
            cap.set(cv2.CAP_PROP_POS_FRAMES, frame_count)
            ret, frame = cap.read()
            if ret:
                frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
                frame_file = os.path.join(video_sample_path, f"frame_{frame_count}.png")
                # save frames
                cv2.imwrite(frame_file, cv2.cvtColor(frame, cv2.COLOR_RGB2BGR))
            frame_count += sample_interval

        cap.release()

def adjust_gamma(image, gamma=1.0):
    invGamma = 1.0 / gamma
    table = np.array([((i / 255.0) ** invGamma) * 255
                      for i in np.arange(0, 256)]).astype("uint8")

    return cv2.LUT(image, table)

def visualize_frames(path):
    frame_files = [os.path.join(path, f) for f in os.listdir(path) if f.endswith(('.png', '.jpg', '.jpeg'))]

    if not frame_files:
        print("No frames to display.")
        return

    frames = [cv2.imread(frame_file) for frame_file in frame_files]
    frames = [adjust_gamma(frame, gamma=3) for frame in frames if frame is not None]

    num_frames = len(frames)
    cols = 5
    rows = np.ceil(num_frames / cols).astype(int)

    fig, axs = plt.subplots(rows, cols, figsize=(5 * cols, 5 * rows))
    if not isinstance(axs, np.ndarray):
        axs = np.array([axs])
    axs = axs.flatten()

    for i in range(rows * cols):
        ax = axs[i]
        if i < num_frames:
            ax.imshow(frames[i])
            ax.axis('off')
        else:
            ax.set_visible(False)

    plt.tight_layout()
```

```
    plt.show()

# ------------------------------------------------------------------ #
# nn_classifier.py
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, Dataset
import os

class MyDataset(Dataset):
    def __init__(self, features, labels):
        self.features = features
        self.labels = labels

    def __len__(self):
        return len(self.features)

    def __getitem__(self, idx):
        feature = torch.tensor(self.features[idx], dtype=torch.float32)
        label = torch.tensor(self.labels[idx], dtype=torch.long)
        return feature, label

class SimpleNN(nn.Module):
    def __init__(self, input_size, hidden_size, num_classes):
        super(SimpleNN, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(hidden_size, num_classes)
        self.dropout = nn.Dropout(0.5)

    def forward(self, x):
        out = self.fc1(x)
        out = self.relu(out)
        out = self.dropout(out)
        out = self.fc2(out)
        return out

def clf_with_model_inference(X_test, y_test, model, device):
    test_dataset = MyDataset(X_test, y_test)
    test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)

    model.eval()
    y_predict = []
    with torch.no_grad():
        for images, _ in test_loader:
            images = images.to(device)
            outputs = model(images)
            _, predicted = torch.max(outputs.data, 1)
            y_predict.extend(predicted.cpu().numpy())
    return y_predict

def clf_without_model_inference(X_test, y_test, model_path, device):
    test_dataset = MyDataset(X_test, y_test)
    test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)

    model = SimpleNN(1000, 500, 6).to(device)
    model.load_state_dict(torch.load(model_path))
    model = model.to(device)
    model.eval()

    y_out = []
    with torch.no_grad():
        for images, _ in test_loader:
            images = images.to(device)
```

```
            outputs = model(images)
            _, predicted = torch.max(outputs.data, 1)
            y_out.extend(predicted.cpu().numpy())
    return y_out

base_lr = 0.001
num_epochs = 20
d_1 = 30
d_2 = 40
def lr_lambda(epoch):
    if epoch < d_1:
        return 1 + 9 * (epoch / d_1)
    elif epoch < d_2:
        return 0.8
    else:
        return 0.5

def classifier_train(X_train, y_train, model_path, device):
    train_dataset = MyDataset(X_train, y_train)
    train_loader = DataLoader(train_dataset, batch_size=4, shuffle=True)

    num_classes = 6
    model = SimpleNN(1000, 500, num_classes).to(device)

    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(model.parameters(), lr=base_lr)
    scheduler = optim.lr_scheduler.LambdaLR(optimizer, lr_lambda)

    for epoch in range(num_epochs):
        for i, (images, labels) in enumerate(train_loader):
            images, labels = images.to(device), labels.to(device)

            outputs = model(images)
            loss = criterion(outputs, labels)

            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
            scheduler.step()

            current_lr = optimizer.param_groups[0]['lr']
            print(f'Epoch [{epoch + 1}/{num_epochs}], LR: {current_lr:.5f
                }, Step [{i + 1}/{len(train_loader)}], Loss: {loss.item()
                :.4f}')

    print('Finished Training')
    os.makedirs(os.path.dirname(model_path), exist_ok=True)
    torch.save(model.state_dict(), model_path)

    return model

# ---------------------------------------------------------------- #
# main of second part
import torch
from mmengine.config import Config, DictAction
from mmengine.runner import Runner
from mmaction.registry import RUNNERS
from zzj_main import zzj_run_main

def run_inference():
    # load config
    cfg = Config.fromfile('/tmp/pycharm_project_zzj/zzj_csn_r152.py')
    cfg.work_dir = '/tmp/pycharm_project_zzj/zzj_csn_200116_g5_r50'
    cfg.load_from = '/tmp/pycharm_project_zzj/zzj_csn_200116_g5_r50/
        best_acc_top1_epoch_85.pth'
```

```
    # build the runner from config
    if 'runner_type' not in cfg:
        # build the default runner
        runner = Runner.from_cfg(cfg)
    else:
        # build customized runner from the registry
        # if 'runner_type' is set in the cfg
        runner = RUNNERS.build(cfg)
    # start testing
    runner.test()

def run_train():
    torch.cuda.empty_cache()
    cfg = Config.fromfile('/tmp/pycharm_project_zzj/zzj_csn_r152.py')
    # merge cli arguments to config
    cfg.work_dir = '/tmp/pycharm_project_zzj/zzj_csn_200116_g5_r50'
    # build the runner from config
    if 'runner_type' not in cfg:
        # build the default runner
        runner = Runner.from_cfg(cfg)
    else:
        # build customized runner from the registry
        # if 'runner_type' is set in the cfg
        runner = RUNNERS.build(cfg)
    # start training
    runner.train()
    torch.cuda.empty_cache()

if __name__ == '__main__':
    run_inference()

# ---------------------------------------------------------------- #
# config of end to end model
# zzj_csn_r152.py

_base_ = [
'/tmp/pycharm_project_zzj/mmaction2/configs/_base_/models/ircsn_r152.py',
    '/tmp/pycharm_project_zzj/mmaction2/configs/_base_/default_runtime.
    py'
]

# dataset settings
dataset_type = 'VideoDataset'
data_root = '/tmp/pycharm_project_zzj/dataset/train/'
data_root_val = '/tmp/pycharm_project_zzj/dataset/validate/'
data_root_test = data_root_val
ann_file_train = '/tmp/pycharm_project_zzj/dataset/processed_train.txt'
ann_file_val = '/tmp/pycharm_project_zzj/dataset/processed_validate.txt'
ann_file_test = ann_file_val

# model settings
model = dict(
    backbone=dict(
        # depth=50,
        norm_eval=True,
        bn_frozen=True,
        norm_eval=True,
        type='ResNet3dCSN',
        # pretrained='https://download.openmmlab.com/mmaction/recognition
            /csn/'
        # 'ircsn_from_scratch_r50_ig65m_20210617-ce545a37.pth'),
        pretrained='https://download.openmmlab.com/mmaction/recognition/
            csn/'
        'ircsn_from_scratch_r152_ig65m_20200807-771c4135.pth'),
    data_preprocessor=dict(
```

```
        format_shape='NCTHW',
        mean=[0.07, 0.07, 0.07],
        std=[0.1, 0.09, 0.08],
        type='ActionDataPreprocessor')
        )

file_client_args = dict(io_backend='disk')

train_pipeline = [
    dict(type='DecordInit', **file_client_args),
    # dict(type='SampleFrames', clip_len=40, frame_interval=1, num_clips
        =1),
    dict(type='UniformSampleFrames', clip_len=40, gamma=5), # add gamma
    dict(type='DecordDecode'),
    dict(type='Resize', scale=(-1, 64)),
    dict(type='RandomResizedCrop'),
    dict(type='Resize', scale=(56, 56), keep_ratio=False),
    dict(type='Flip', flip_ratio=0),
    dict(type='FormatShape', input_format='NCTHW'),
    dict(type='PackActionInputs')
]
val_pipeline = [
    dict(type='DecordInit', **file_client_args),
    # dict(type='SampleFrames', clip_len=40, frame_interval=1, num_clips
        =1, test_mode=True),
    dict(type='UniformSampleFrames', clip_len=40, num_clips=1, test_mode=
        True, gamma=5),
    dict(type='DecordDecode'),
    dict(type='Resize', scale=(-1, 64)),
    dict(type='CenterCrop', crop_size=56),
    dict(type='FormatShape', input_format='NCTHW'),
    dict(type='PackActionInputs')
]
test_pipeline = [
    dict(type='DecordInit', **file_client_args),
    # dict(type='SampleFrames', clip_len=40, frame_interval=1, num_clips
        =1, test_mode=True),
    dict(type='UniformSampleFrames', clip_len=40, num_clips=1, test_mode=
        True, gamma=5),
    dict(type='DecordDecode'),
    dict(type='Resize', scale=(-1, 64)),
    dict(type='ThreeCrop', crop_size=64),
    dict(type='FormatShape', input_format='NCTHW'),
    dict(type='PackActionInputs')
]

train_dataloader = dict(
    batch_size=12,
    num_workers=8,
    persistent_workers=True,
    sampler=dict(type='DefaultSampler', shuffle=True),
    dataset=dict(
        type=dataset_type,
        ann_file=ann_file_train,
        data_prefix=dict(video=data_root),
        num_classes=6,
        pipeline=train_pipeline))

val_dataloader = dict(
    batch_size=1,
    num_workers=8,
    persistent_workers=True,
    sampler=dict(type='DefaultSampler', shuffle=False),
    dataset=dict(
        type=dataset_type,
```

```
        ann_file=ann_file_val,
        data_prefix=dict(video=data_root_val),
        pipeline=val_pipeline,
        num_classes=6,
        test_mode=True))

test_dataloader = dict(
    batch_size=1,
    num_workers=8,
    persistent_workers=True,
    sampler=dict(type='DefaultSampler', shuffle=False),
    dataset=dict(
        type=dataset_type,
        ann_file=ann_file_test,
        data_prefix=dict(video=data_root_val),
        pipeline=test_pipeline,
        num_classes=6,
        test_mode=True))

val_evaluator = dict(type='AccMetric')
test_evaluator = val_evaluator

train_cfg = dict(
    type='EpochBasedTrainLoop', max_epochs=100, val_begin=1, val_interval
        =5)
val_cfg = dict(type='ValLoop')
test_cfg = dict(type='TestLoop')

param_scheduler = [
    dict(type='LinearLR', start_factor=0.1, by_epoch=True, begin=0, end
        =100),
    dict(
        type='MultiStepLR',
        begin=0,
        end=100,
        by_epoch=True,
        milestones=[60, 80],
        gamma=0.1)
]

optim_wrapper = dict(
    optimizer=dict(type='SGD', lr=5e-4, momentum=0.9, weight_decay=1e-4),
    clip_grad=dict(max_norm=40, norm_type=2))

default_hooks = dict(checkpoint=dict(interval=5, max_keep_ckpts=5))
find_unused_parameters = True

auto_scale_lr = dict(enable=False, base_batch_size=96)

# ------------------------------------------------------------------- #
# plot the Variation Curves from log
# info_plt.py
import matplotlib.pyplot as plt
import re

# The log content
with open('./20231116_201432.log', 'r') as file:
    log_content = file.read()

def info_plt_func():

    # Extract training and validation losses and accuracies
    train_loss = re.findall(r"Epoch\(train\)\s+\[(\d+)\].*loss: (\d+\.\d
        +)", log_content)
```

```
train_top1_acc = re.findall(r"Epoch\(train\)\s+\[(\d+)\].*top1_acc: (\d+\.\d+)", log_content)
val_top1_acc = re.findall(r"Epoch\(val\)\s+\[(\d+)\].*acc/top1: (\d+\.\d+)", log_content)

# Convert strings to floats and match epochs
train_epochs, train_loss_values = zip(*[(int(epoch), float(loss)) for epoch, loss in train_loss])
train_top1_epochs, train_top1_acc_values = zip(*[(int(epoch), float(acc)) for epoch, acc in train_top1_acc])
val_epochs, val_top1_acc_values = zip(*[(int(epoch), float(acc)) for epoch, acc in val_top1_acc])

lr_pattern = r"Epoch\(train\)\s+\[\d+\].*lr: (\d+\.\d+e?-?\d*)"
learning_rates = re.findall(lr_pattern, log_content)

# Extract epochs for learning rates
epochs_pattern = r"Epoch\(train\)\s+\[(\d+)\]"
epochs = re.findall(epochs_pattern, log_content)

# Convert strings to floats for learning rates and integers for epochs
lr_values = [float(lr) for lr in learning_rates]
epochs = [int(epoch) for epoch in epochs]

# Plot the learning rates
plt.figure(figsize=(7, 4))
plt.plot(epochs, lr_values, label='Learning Rate', color='orange')
plt.title('Learning Rate Over Epochs')
plt.xlabel('Epoch')
plt.ylabel('Learning Rate')
plt.legend()
plt.grid(True)
plt.show()

# Plot the training loss
plt.figure(figsize=(14, 4))
plt.subplot(1, 3, 1)
plt.plot(train_epochs, train_loss_values, label='Train Loss', color='blue')
plt.title('Training Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

# Plot the training top1 accuracy
plt.subplot(1, 3, 2)
plt.plot(train_top1_epochs, train_top1_acc_values, label='Train Top1 Accuracy', color='green')
plt.title('Training Top1 Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Top1 Accuracy')
plt.legend()

# Plot the validation top1 accuracy
plt.subplot(1, 3, 3)
plt.plot(val_epochs, val_top1_acc_values, label='Validation Top1 Accuracy', color='red')
plt.title('Validation Top1 Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Top1 Accuracy')
plt.legend()

# Display the plots
plt.tight_layout()
```

```
    plt.show()

# ------------------------------------------------------------------ #
# Preprocess the label document
# txt_pre.py
test_txt_path = './../dataset/validate.txt'
train_txt_path = './../dataset/train.txt'

sampled_test_path = './../ass2/sampled/test/'
sampled_train_path = './../ass2/sampled/train/'

def update_paths_in_ann_file(input_file, output_file, old_str, new_str):
    with open(input_file, 'r') as file:
        lines = file.readlines()

    with open(output_file, 'w') as file:
        for line in lines:
            updated_line = line.replace(old_str, new_str)
            file.write(updated_line)

old_str = './../ass2/sampled/'
new_str = ''

update_paths_in_ann_file('./val_ann.txt', 'updated_val_ann.txt', old_str,
    new_str)
update_paths_in_ann_file('train_ann.txt', 'updated_train_ann.txt',
    old_str, new_str)

def create_mmaction2_ann_file(original_ann_path, sampled_data_path,
    output_file):
    with open(original_ann_path, 'r') as orig_ann_file, open(output_file,
        'w') as out_ann_file:
        for line in orig_ann_file:
            parts = line.strip().split()
            label = parts[1]
            video_name = os.path.splitext(parts[2])[0]

            frame_dir = os.path.join(sampled_data_path, video_name)
            num_frames = len([name for name in os.listdir(frame_dir) if
                os.path.isfile(os.path.join(frame_dir, name))])

            path_online_ = './sampled/test'
            out_ann_file.write(f'{frame_dir} {num_frames} {label}\n')

create_mmaction2_ann_file(train_txt_path, sampled_train_path, 'train_ann.
    txt')
create_mmaction2_ann_file(test_txt_path, sampled_test_path, 'val_ann.txt'
    )

# ------------------------------------------------------------------ #
# result of end-to-end model on test dataset
# 20231119_184157.json
{"acc/top1": 0.6770833333333334, "acc/top5": 0.9895833333333334, "acc/
    mean1": 0.6764297385620915, "data_time": 0.013988114893436432, "time"
    : 0.321664238969485}

# ------------------------------------------------------------------ #
# output of first part
/root/miniconda3/bin/python /tmp/pycharm_project_zzj/main.py
Using device: cuda
Start load and extract features of train
Processing videos: 100%|| 150/150 [00:00<00:00, 6638.38video/s]
Train ...
Finished Training
Start load and extract features of test
```

```
Processing videos: 100%|| 96/96 [00:00<00:00, 7160.32video/s]
Predict ...
Train Accuracy: 0.9466666666666667
Test Accuracy: 0.25

Process finished with exit code 0
```