HUMAN ACTION RECOGNITION IN THE DARK: A SIMPLE EXPLORATION WITH LATE FUSION AND IMAGE ENHANCEMENT

Anonymous authors
Paper under double-blind review

ABSTRACT

In this study, we develop an enhanced Human Action Recognition (HAR) model, starting with uniform frame sampling and feature extraction using ResNet-50. The model employs a Support Vector Machine (SVM) for classification, achieving notable accuracy, further improved by image enhancements like histogram equalization. The culmination of our research is an end-to-end HAR model using Vision Transformer (ViT) with self-supervised learning, demonstrating superior action recognition performance but at the cost of increased computational demands and reduced interpretability.

1 STEP 1 – FRAME SAMPLING

Take Jump_8_1.mp4 as an example.

Figure 1: Uniform Sampling

Both methods reduce the total number of frames analyzed, easing computational load.

Uniform sampling maintains a structured, evenly spaced view of the entire video, capturing each phase of the exercise sequence systematically. However, random sampling does not guarantee even coverage of the video. It might miss critical phases of the exercise sequence or over-represent less significant actions.

So, we decide to use uniform sampling, which ensures that every part of the video is represented, capturing the complete range of activities.

2 STEP 2 – FEATURE EXTRACTION

In our approach, we leverage the ResNet-50, a pre-trained deep learning model, due to its effectiveness in image recognition tasks. This model is widely recognized for its ability to handle complex

Figure 2: Random Sampling

patterns in images, making it suitable for extracting features from video frames. ResNet-50 stands out for its deep architecture which helps in learning detailed features, essential for analyzing varied visual elements in videos.

The feature extraction involves normalizing the frames to zero-mean and unit standard deviation. This step is critical, especially for darker frames, to ensure consistent visibility and contrast across all frames. The ResNet-50 model processes each normalized frame and outputs a 64/256/512/1024/2048-dimensional feature vector. This dimensionality provides a comprehensive representation of each frame's visual characteristics.

For subsequent training phases, these features are averaged across all frames to obtain a unified feature representation of the video. This averaged feature set is saved as a NumPy array, facilitating efficient use in later stages of analysis or model training. This approach ensures that the raw video data is transformed into a format ready for deeper analysis or predictive modeling.

3 Step 3 – Classifier Training and Evaluation

In our study, we employed the Support Vector Machine (SVM) classifier for video classification due to its efficiency in handling high-dimensional spaces, typical in video datasets where each frame contributes to a complex feature vector. SVM's ability to find a hyperplane for class separation is especially effective in both binary and multi-class scenarios, making it a suitable choice for this application. However, SVMs come with their own set of limitations, including computational intensity in handling large datasets and sensitivity to kernel choice and parameter tuning.

For evaluation, the SVM classifier was trained using features extracted from a set of training videos, followed by a similar feature extraction from a validation set to assess generalization capabilities. The classifier's performance was measured against ground truth labels from the validation set.

Finally, we achieved a validation accuracy of 93.3% and a test accuracy of 9.375%.

4 Step 4 – Effects of Leveraging Image Enhancements

In this section, our exploration focused on the impact of image enhancements on our HAR model's performance. We implemented histogram equalization to improve visibility in low-light video frames. This enhancement was crucial in adapting the frames for effective use of the pre-trained ResNet-50 model, which was initially trained on the ImageNet dataset. By normalizing the frames with the mean [0.485, 0.456, 0.406] and standard deviation [0.229, 0.224, 0.225], as commonly used for ImageNet, we ensured compatibility with the pre-trained weights of ResNet-50. This approach led to enhanced feature extraction from the videos, resulting in a noticeable improvement in the model's accuracy for action recognition.

Finally, we achieved a validation accuracy of 96.7% and a test accuracy of 16.67%.

Below is an example of enhanced frames

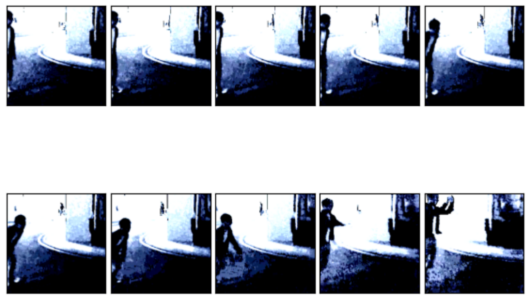

Figure 3: Enhanced Frames

5 Step 5 – Improving the HAR Model to Enable End-to-end Training

We developed an end-to-end Human Action Recognition (HAR) model using the Vision Transformer (ViT) as the backbone (Dosovitskiy et al., 2020), supplemented by MoCo v3 (Fan et al., 2021) inspired self-supervised learning on the Something-Something V2 dataset. This approach differs from traditional methods by incorporating the Transformer architecture, known for its effectiveness in processing sequences and capturing global dependencies.

The training involved a two-phase approach: self-supervised learning on the diverse Something-Something V2 dataset to learn a generalized representation of human actions, followed by fine-tuning on our specific dataset to tailor the model to our classification needs. This process allowed the model to capture complex patterns in video data, enhancing its understanding of human actions.

When compared to the previously used SVM + ResNet model, our ViT-based model demonstrated superior performance in recognizing human actions. However, this advancement comes with increased computational demands and lower interpretability. The Transformer architecture, while powerful, requires more computational resources, and its complex nature makes the model's decision-making process less transparent than the more straightforward SVM + ResNet approach. This poses challenges, particularly in resource-constrained settings and applications where model interpretability is crucial.

References

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.

Haoqi Fan, Bo Xiong, Karttikeya Mangalam, Yanghao Li, Zhicheng Yan, Jitendra Malik, and Christoph Feichtenhofer. Multiscale vision transformers. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 6824–6835, 2021.

A Appendix

A.1 Code of Section 1

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

def uniform_sampling(video_path, sample_size):
    """
    Uniform Sampling function
    :param video_path: Path to the video
    :param sample_size: Number of frames to sample
    :return: List of sampled frames
    """
    cap = cv2.VideoCapture(video_path)
    total_frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
    step = total_frames // sample_size

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

    cap.release()
    return frames

def random_sampling(video_path, sample_size):
    """
    Random Sampling function
    :param video_path: Path to the video
    :param sample_size: Number of frames to sample
    :return: List of sampled frames
    """
    cap = cv2.VideoCapture(video_path)
    total_frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))

    frames = []
    sampled_indices = np.random.choice(range(total_frames), size=
        sample_size, replace=False)
    for i in sampled_indices:
        cap.set(cv2.CAP_PROP_POS_FRAMES, i)
        ret, frame = cap.read()
        if ret:
            frames.append(frame)

    cap.release()
    return frames

def create_montage(frames, title):
    """
    Create and display a montage of frames
    :param frames: List of frames
    :param title: Title of the montage
    """
    rows = int(np.ceil(np.sqrt(len(frames))))
    fig, axarr = plt.subplots(rows, rows)
    fig.suptitle(title)

    for i in range(rows * rows):
        row = i // rows
        col = i % rows
        axarr[row, col].axis('off')
        if i < len(frames):
            axarr[row, col].imshow(cv2.cvtColor(frames[i], cv2.
                COLOR_BGR2RGB))
```

```

    plt.show()

video_path = '/content/train/Jump/Jump_8_1.mp4'
sample_size = 10  # Number of frames to sample

uniform_frames = uniform_sampling(video_path, sample_size)
random_frames = random_sampling(video_path, sample_size)

# Create and display montages
create_montage(uniform_frames, "Uniform Sampling Montage")
create_montage(random_frames, "Random Sampling Montage")
```

A.2 CODE OF SECTION 2

```
import torch
import torchvision.transforms as transforms
import torchvision.models as models
import numpy as np
import cv2

def extract_features(video_frames, model_name='resnet50', cut_layer=-2):
    """
    Extract features from video frames using a part of a pre-trained
        model.

    :param video_frames: List of frames from a video
    :param model_name: Name of the pre-trained model to use
    :param cut_layer: Index of the layer where the model is cut
    :return: Numpy array of extracted features
    """

    model = models.__dict__[model_name](pretrained=True)

    if cut_layer != -1:
        model = torch.nn.Sequential(*list(model.children())[:cut_layer])
    model.eval()

    preprocess = transforms.Compose([
        transforms.ToPILImage(),
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.07, 0.07, 0.07], std=[0.1, 0.09,
            0.08]),
    ])

    features = []
    with torch.no_grad():
        for frame in video_frames:
            input_tensor = preprocess(frame)
            input_batch = input_tensor.unsqueeze(0)

            if torch.cuda.is_available():
                input_batch = input_batch.to('cuda')
                model.to('cuda')

            output = model(input_batch)
            features.append(output.squeeze().cpu().numpy())

    return features

def uniform_sampling(video_path, sample_size):
```

```
    """
    Uniform Sampling function
    :param video_path: Path to the video
    :param sample_size: Number of frames to sample
    :return: List of sampled frames
    """
    cap = cv2.VideoCapture(video_path)
    total_frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
    step = total_frames // sample_size

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

    cap.release()
    return frames


video_path = 'EE6222 train and validate 2023/train/Jump/Jump_8_1.mp4'
sample_size = 10  # Number of frames to sample

uniform_frames = uniform_sampling(video_path, sample_size)


features = extract_features(uniform_frames, cut_layer=8)

np.save('video_features.npy', features)
```

A.3 Code of Section 3

```
import os
import torch
import torchvision.transforms as transforms
import torchvision.models as models
import numpy as np
import cv2
from tqdm import tqdm
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score


def uniform_sampling(video_path, sample_size):
    """
    Uniform Sampling function
    :param video_path: Path to the video
    :param sample_size: Number of frames to sample
    :return: List of sampled frames
    """
    cap = cv2.VideoCapture(video_path)
    total_frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
    step = max(1, total_frames // sample_size)

    frames = []
    for i in range(0, total_frames, step):
        if len(frames) >= sample_size:
            break
        cap.set(cv2.CAP_PROP_POS_FRAMES, i)
        ret, frame = cap.read()
```

```
            if ret:
                frames.append(frame)

        cap.release()
        return frames


    def extract_features(video_frames, model_name='resnet50', cut_layer=-2):
        """
        Extract features from video frames using a part of a pre-trained
            model.

        :param video_frames: List of frames from a video
        :param model_name: Name of the pre-trained model to use
        :param cut_layer: Index of the layer where the model is cut
        :return: Numpy array of extracted features
        """

        device = torch.device("cuda" if torch.cuda.is_available() else "cpu")


        model = models.__dict__[model_name](pretrained=True)
        if cut_layer != -1:
            model = torch.nn.Sequential(*list(model.children())[:cut_layer])
        model = model.to(device)
        model.eval()


        preprocess = transforms.Compose([
            transforms.ToPILImage(),
            transforms.Resize(256),
            transforms.CenterCrop(224),
            transforms.ToTensor(),
            transforms.Normalize(mean=[0.07, 0.07, 0.07], std=[0.1, 0.09,
                0.08]),
        ])


        all_features = []
        with torch.no_grad():
            for frame in video_frames:
                input_tensor = preprocess(frame)
                input_batch = input_tensor.unsqueeze(0).to(device)

                output = model(input_batch)

                flattened_output = output.view(output.size(0), -1)
                all_features.append(flattened_output.squeeze().cpu().numpy())


        all_features_flat = np.concatenate(all_features, axis=0)
        return all_features_flat


    def process_videos(video_dir, category, sample_size):
        video_features = []
        video_labels = []


        video_files = os.listdir(video_dir)
        pbar = tqdm(total=len(video_files), desc=f"Processing {category}")

        for video_file in video_files:
            video_path = os.path.join(video_dir, video_file)

```

```
        uniform_frames = uniform_sampling(video_path, sample_size)

        # cut_layer=8, features=2048,7,7
        # cut_layer=7, features=1024,14,14
        # cut_layer=6, features=512,28,28
        # cut_layer=5, features=256,56,56
        # cut_layer=4, features=64,56,56
        features = extract_features(uniform_frames, cut_layer=8)
        video_features.append(features)
        video_labels.append(category)


        pbar.update(1)

    pbar.close()
    return video_features, video_labels


def parse_label_file(label_file):
    labels = {}
    with open(label_file, 'r') as file:
        for line in file:
            _, label, filename = line.strip().split()
            labels[filename] = int(label)
    return labels


def process_test_videos(test_dir, labels, sample_size):
    video_features = []
    video_labels = []

    video_files = os.listdir(test_dir)
    pbar = tqdm(total=len(video_files), desc="Processing Test Videos")

    for video_file in video_files:
        if video_file in labels:
            video_path = os.path.join(test_dir, video_file)
            uniform_frames = uniform_sampling(video_path, sample_size)
            features = extract_features(uniform_frames, cut_layer=8)
            video_features.append(features)
            video_labels.append(labels[video_file])

            pbar.update(1)

    pbar.close()
    return video_features, video_labels



categories = {
    'Jump': 'EE6222 train and validate 2023/train/Jump/',
    'Run': 'EE6222 train and validate 2023/train/Run/',
    'Sit': 'EE6222 train and validate 2023/train/Sit/',
    'Stand': 'EE6222 train and validate 2023/train/Stand/',
    'Turn': 'EE6222 train and validate 2023/train/Turn/',
    'Walk': 'EE6222 train and validate 2023/train/Walk/'
}


all_features = []
all_labels = []
for label, dir_path in categories.items():
    features, labels = process_videos(dir_path, label, 10)
    all_features.extend(features)
    all_labels.extend(labels)
```

```


label_to_index = {label: index for index, label in enumerate(categories.keys())}
all_labels = [label_to_index[label] for label in all_labels]


X_train, X_test, y_train, y_test = train_test_split(all_features, all_labels, test_size=0.2, random_state=42)


svm_classifier = SVC()
svm_classifier.fit(X_train, y_train)


bayes_classifier = GaussianNB()
bayes_classifier.fit(X_train, y_train)


svm_predictions = svm_classifier.predict(X_test)
bayes_predictions = bayes_classifier.predict(X_test)

print("SVM␣Accuracy:", accuracy_score(y_test, svm_predictions))
print("Bayes␣Accuracy:", accuracy_score(y_test, bayes_predictions))

labels = parse_label_file(’EE6222␣train␣and␣validate␣2023/validate.txt’)
test_dir = ’EE6222␣train␣and␣validate␣2023/validate’
test_features, test_labels = process_test_videos(test_dir, labels, 10)


test_features = np.array(test_features)


test_predictions = svm_classifier.predict(test_features)


print("Test␣Accuracy:", accuracy_score(test_labels, test_predictions))
```

A.4 CODE OF SECTION 4

```
import os
import torch
import torchvision.transforms as transforms
import torchvision.models as models
import numpy as np
import cv2
from tqdm import tqdm
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score


def enhance_image(frame):
    yuv = cv2.cvtColor(frame, cv2.COLOR_BGR2YUV)
    yuv[:, :, 0] = cv2.equalizeHist(yuv[:, :, 0])
    enhanced_frame = cv2.cvtColor(yuv, cv2.COLOR_YUV2BGR)
    return enhanced_frame


def uniform_sampling(video_path, sample_size, enhance=False):
    cap = cv2.VideoCapture(video_path)
    total_frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
    step = max(1, total_frames // sample_size)
```

```
    frames = []
    for i in range(0, total_frames, step):
        if len(frames) >= sample_size:
            break
        cap.set(cv2.CAP_PROP_POS_FRAMES, i)
        ret, frame = cap.read()
        if ret:
            if enhance:
                frame = enhance_image(frame)
            frames.append(frame)
    cap.release()
    return frames


def extract_features(video_frames, model_name='resnet50', cut_layer=-2):
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    model = models.__dict__[model_name](pretrained=True)
    if cut_layer != -1:
        model = torch.nn.Sequential(*list(model.children())[:cut_layer])
    model = model.to(device)
    model.eval()
    preprocess = transforms.Compose([
        transforms.ToPILImage(),
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        # transforms.Normalize(mean=[0.07, 0.07, 0.07], std=[0.1, 0.09, 0.08]),
        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
    ])
    all_features = []
    with torch.no_grad():
        for frame in video_frames:
            input_tensor = preprocess(frame)
            input_batch = input_tensor.unsqueeze(0).to(device)
            output = model(input_batch)
            flattened_output = output.view(output.size(0), -1)
            all_features.append(flattened_output.squeeze().cpu().numpy())
    all_features_flat = np.concatenate(all_features, axis=0)
    return all_features_flat


def process_videos(video_dir, category, sample_size, enhance=False):
    video_features = []
    video_labels = []
    video_files = os.listdir(video_dir)
    pbar = tqdm(total=len(video_files), desc=f"Processing {category}")
    for video_file in video_files:
        video_path = os.path.join(video_dir, video_file)
        uniform_frames = uniform_sampling(video_path, sample_size, enhance)
        features = extract_features(uniform_frames, cut_layer=8)
        video_features.append(features)
        video_labels.append(category)
        pbar.update(1)
    pbar.close()
    return video_features, video_labels


def parse_label_file(label_file):
    labels = {}
    with open(label_file, 'r') as file:
        for line in file:
            _, label, filename = line.strip().split()
```

```python
            labels[filename] = int(label)
    return labels


def process_test_videos(test_dir, labels, sample_size, enhance=False):
    video_features = []
    video_labels = []
    video_files = os.listdir(test_dir)
    pbar = tqdm(total=len(video_files), desc="Processing Test Videos")
    for video_file in video_files:
        if video_file in labels:
            video_path = os.path.join(test_dir, video_file)
            uniform_frames = uniform_sampling(video_path, sample_size, enhance)
            features = extract_features(uniform_frames, cut_layer=8)
            video_features.append(features)
            video_labels.append(labels[video_file])
            pbar.update(1)
    pbar.close()
    return video_features, video_labels


categories = {
    'Jump': 'EE6222 train and validate 2023/train/Jump/',
    'Run': 'EE6222 train and validate 2023/train/Run/',
    'Sit': 'EE6222 train and validate 2023/train/Sit/',
    'Stand': 'EE6222 train and validate 2023/train/Stand/',
    'Turn': 'EE6222 train and validate 2023/train/Turn/',
    'Walk': 'EE6222 train and validate 2023/train/Walk/'
}

all_features = []
all_labels = []
for label, dir_path in categories.items():
    features, labels = process_videos(dir_path, label, 10, enhance=True)
    all_features.extend(features)
    all_labels.extend(labels)

label_to_index = {label: index for index, label in enumerate(categories.keys())}
all_labels = [label_to_index[label] for label in all_labels]

X_train, X_test, y_train, y_test = train_test_split(all_features, all_labels, test_size=0.2, random_state=42)

svm_classifier = SVC()
svm_classifier.fit(X_train, y_train)

svm_predictions = svm_classifier.predict(X_test)
print("SVM Accuracy:", accuracy_score(y_test, svm_predictions))

labels = parse_label_file('EE6222 train and validate 2023/validate.txt')
test_dir = 'EE6222 train and validate 2023/validate'
test_features, test_labels = process_test_videos(test_dir, labels, 10, enhance=True)
test_features = np.array(test_features)
test_predictions = svm_classifier.predict(test_features)
print("Test Accuracy:", accuracy_score(test_labels, test_predictions))
```