

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

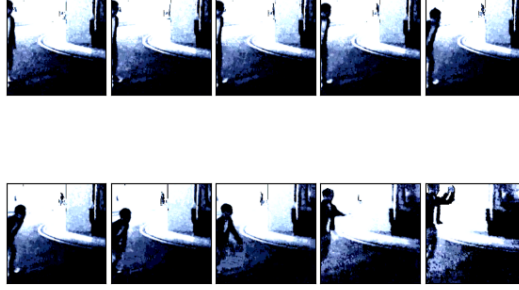


Figure 3: Enhanced Frames

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

```
1 import cv2
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 def uniform_sampling(video_path, sample_size):
6     """
7     Uniform Sampling function
8     :param video_path: Path to the video
9     :param sample_size: Number of frames to sample
10    :return: List of sampled frames
11    """
12    cap = cv2.VideoCapture(video_path)
13    total_frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
14    step = total_frames // sample_size
15
16    frames = []
17    for i in range(0, total_frames, step):
18        cap.set(cv2.CAP_PROP_POS_FRAMES, i)
19        ret, frame = cap.read()
20        if ret:
21            frames.append(frame)
22
23    cap.release()
24    return frames
25
26 def random_sampling(video_path, sample_size):
27     """
28     Random Sampling function
29     :param video_path: Path to the video
30     :param sample_size: Number of frames to sample
31     :return: List of sampled frames
32     """
33    cap = cv2.VideoCapture(video_path)
34    total_frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
35
36    frames = []
37    sampled_indices = np.random.choice(range(total_frames), size=
38        sample_size, replace=False)
39    for i in sampled_indices:
40        cap.set(cv2.CAP_PROP_POS_FRAMES, i)
41        ret, frame = cap.read()
42        if ret:
43            frames.append(frame)
44
45    cap.release()
46    return frames
47
48 def create_montage(frames, title):
49     """
50     Create and display a montage of frames
51     :param frames: List of frames
52     :param title: Title of the montage
53     """
54    rows = int(np.ceil(np.sqrt(len(frames))))
55    fig, axarr = plt.subplots(rows, rows)
56    fig.suptitle(title)
57
58    for i in range(rows * rows):
59        row = i // rows
60        col = i % rows
61        axarr[row, col].axis('off')
62        if i < len(frames):
63            axarr[row, col].imshow(cv2.cvtColor(frames[i], cv2.
64                COLOR_BGR2RGB))
```

```
63     plt.show()
64
65
66 video_path = '/content/train/Jump/Jump_8_1.mp4'
67 sample_size = 10  # Number of frames to sample
68
69 uniform_frames = uniform_sampling(video_path, sample_size)
70 random_frames = random_sampling(video_path, sample_size)
71
72 # Create and display montages
73 create_montage(uniform_frames, "Uniform_Sampling_Montage")
74 create_montage(random_frames, "Random_Sampling_Montage")
```

A.2 CODE OF SECTION 2

```
1  import torch
2  import torchvision.transforms as transforms
3  import torchvision.models as models
4  import numpy as np
5  import cv2
6
7  def extract_features(video_frames, model_name='resnet50', cut_layer=-2):
8      """
9      Extract features from video frames using a part of a pre-trained
10         model.
11
12         :param video_frames: List of frames from a video
13         :param model_name: Name of the pre-trained model to use
14         :param cut_layer: Index of the layer where the model is cut
15         :return: Numpy array of extracted features
16         """
17
18     model = models.__dict__[model_name](pretrained=True)
19
20     if cut_layer != -1:
21         model = torch.nn.Sequential(*list(model.children())[:cut_layer])
22     model.eval()
23
24     preprocess = transforms.Compose([
25         transforms.ToPILImage(),
26         transforms.Resize(256),
27         transforms.CenterCrop(224),
28         transforms.ToTensor(),
29         transforms.Normalize(mean=[0.07, 0.07, 0.07], std=[0.1, 0.09,
30             0.08]),
31     ])
32
33     features = []
34     with torch.no_grad():
35         for frame in video_frames:
36             input_tensor = preprocess(frame)
37             input_batch = input_tensor.unsqueeze(0)
38
39             if torch.cuda.is_available():
40                 input_batch = input_batch.to('cuda')
41                 model.to('cuda')
42
43             output = model(input_batch)
44             features.append(output.squeeze().cpu().numpy())
45
46     return features
47
48 def uniform_sampling(video_path, sample_size):
```

```

47     """
48     Uniform Sampling function
49     :param video_path: Path to the video
50     :param sample_size: Number of frames to sample
51     :return: List of sampled frames
52     """
53     cap = cv2.VideoCapture(video_path)
54     total_frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
55     step = total_frames // sample_size
56
57     frames = []
58     for i in range(0, total_frames, step):
59         cap.set(cv2.CAP_PROP_POS_FRAMES, i)
60         ret, frame = cap.read()
61         if ret:
62             frames.append(frame)
63
64     cap.release()
65     return frames
66
67 video_path = 'EE6222_train_and_validate_2023/train/Jump/Jump_8_1.mp4'
68 sample_size = 10 # Number of frames to sample
69
70 uniform_frames = uniform_sampling(video_path, sample_size)
71
72
73 features = extract_features(uniform_frames, cut_layer=8)
74
75 np.save('video_features.npy', features)
76

```

A.3 CODE OF SECTION 3

```

1  import os
2  import torch
3  import torchvision.transforms as transforms
4  import torchvision.models as models
5  import numpy as np
6  import cv2
7  from tqdm import tqdm
8  from sklearn.model_selection import train_test_split
9  from sklearn.svm import SVC
10 from sklearn.naive_bayes import GaussianNB
11 from sklearn.metrics import accuracy_score
12
13
14 def uniform_sampling(video_path, sample_size):
15     """
16     Uniform Sampling function
17     :param video_path: Path to the video
18     :param sample_size: Number of frames to sample
19     :return: List of sampled frames
20     """
21     cap = cv2.VideoCapture(video_path)
22     total_frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
23     step = max(1, total_frames // sample_size)
24
25     frames = []
26     for i in range(0, total_frames, step):
27         if len(frames) >= sample_size:
28             break
29         cap.set(cv2.CAP_PROP_POS_FRAMES, i)
30         ret, frame = cap.read()

```

```

31         if ret:
32             frames.append(frame)
33
34     cap.release()
35     return frames
36
37
38 def extract_features(video_frames, model_name='resnet50', cut_layer=-2):
39     """
40     Extract features from video frames using a part of a pre-trained
41     model.
42
43     :param video_frames: List of frames from a video
44     :param model_name: Name of the pre-trained model to use
45     :param cut_layer: Index of the layer where the model is cut
46     :return: Numpy array of extracted features
47     """
48
49     device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
50
51     model = models.__dict__[model_name](pretrained=True)
52     if cut_layer != -1:
53         model = torch.nn.Sequential(*list(model.children())[:cut_layer])
54     model = model.to(device)
55     model.eval()
56
57
58     preprocess = transforms.Compose([
59         transforms.ToPILImage(),
60         transforms.Resize(256),
61         transforms.CenterCrop(224),
62         transforms.ToTensor(),
63         transforms.Normalize(mean=[0.07, 0.07, 0.07], std=[0.1, 0.09,
64             0.08]),
65     ])
66
67     all_features = []
68     with torch.no_grad():
69         for frame in video_frames:
70             input_tensor = preprocess(frame)
71             input_batch = input_tensor.unsqueeze(0).to(device)
72
73             output = model(input_batch)
74
75             flattened_output = output.view(output.size(0), -1)
76             all_features.append(flattened_output.squeeze().cpu().numpy())
77
78
79     all_features_flat = np.concatenate(all_features, axis=0)
80     return all_features_flat
81
82
83 def process_videos(video_dir, category, sample_size):
84     video_features = []
85     video_labels = []
86
87
88     video_files = os.listdir(video_dir)
89     pbar = tqdm(total=len(video_files), desc=f"Processing_{category}")
90
91     for video_file in video_files:
92         video_path = os.path.join(video_dir, video_file)
93

```

```

94         uniform_frames = uniform_sampling(video_path, sample_size)
95
96         # cut_layer=8, features=2048,7,7
97         # cut_layer=7, features=1024,14,14
98         # cut_layer=6, features=512,28,28
99         # cut_layer=5, features=256,56,56
100        # cut_layer=4, features=64,56,56
101        features = extract_features(uniform_frames, cut_layer=8)
102        video_features.append(features)
103        video_labels.append(category)
104
105
106        pbar.update(1)
107
108    pbar.close()
109    return video_features, video_labels
110
111
112    def parse_label_file(label_file):
113        labels = {}
114        with open(label_file, 'r') as file:
115            for line in file:
116                _, label, filename = line.strip().split()
117                labels[filename] = int(label)
118        return labels
119
120
121    def process_test_videos(test_dir, labels, sample_size):
122        video_features = []
123        video_labels = []
124
125        video_files = os.listdir(test_dir)
126        pbar = tqdm(total=len(video_files), desc="Processing_Test_Videos")
127
128        for video_file in video_files:
129            if video_file in labels:
130                video_path = os.path.join(test_dir, video_file)
131                uniform_frames = uniform_sampling(video_path, sample_size)
132                features = extract_features(uniform_frames, cut_layer=8)
133                video_features.append(features)
134                video_labels.append(labels[video_file])
135
136                pbar.update(1)
137
138        pbar.close()
139        return video_features, video_labels
140
141
142
143    categories = {
144        'Jump': 'EE6222_train_and_validate_2023/train/Jump/',
145        'Run': 'EE6222_train_and_validate_2023/train/Run/',
146        'Sit': 'EE6222_train_and_validate_2023/train/Sit/',
147        'Stand': 'EE6222_train_and_validate_2023/train/Stand/',
148        'Turn': 'EE6222_train_and_validate_2023/train/Turn/',
149        'Walk': 'EE6222_train_and_validate_2023/train/Walk/'
150    }
151
152
153    all_features = []
154    all_labels = []
155    for label, dir_path in categories.items():
156        features, labels = process_videos(dir_path, label, 10)
157        all_features.extend(features)
158        all_labels.extend(labels)

```



```

159
160
161 label_to_index = {label: index for index, label in enumerate(categories.
    keys())}
162 all_labels = [label_to_index[label] for label in all_labels]
163
164
165 X_train, X_test, y_train, y_test = train_test_split(all_features,
    all_labels, test_size=0.2, random_state=42)
166
167
168 svm_classifier = SVC()
169 svm_classifier.fit(X_train, y_train)
170
171
172 bayes_classifier = GaussianNB()
173 bayes_classifier.fit(X_train, y_train)
174
175
176 svm_predictions = svm_classifier.predict(X_test)
177 bayes_predictions = bayes_classifier.predict(X_test)
178
179 print("SVM_Accuracy:", accuracy_score(y_test, svm_predictions))
180 print("Bayes_Accuracy:", accuracy_score(y_test, bayes_predictions))
181
182 labels = parse_label_file('EE6222_train_and_validate_2023/validate.txt')
183 test_dir = 'EE6222_train_and_validate_2023/validate'
184 test_features, test_labels = process_test_videos(test_dir, labels, 10)
185
186
187 test_features = np.array(test_features)
188
189
190 test_predictions = svm_classifier.predict(test_features)
191
192
193 print("Test_Accuracy:", accuracy_score(test_labels, test_predictions))

```

A.4 CODE OF SECTION 4

```

1 import os
2 import torch
3 import torchvision.transforms as transforms
4 import torchvision.models as models
5 import numpy as np
6 import cv2
7 from tqdm import tqdm
8 from sklearn.model_selection import train_test_split
9 from sklearn.svm import SVC
10 from sklearn.metrics import accuracy_score
11
12
13 def enhance_image(frame):
14     yuv = cv2.cvtColor(frame, cv2.COLOR_BGR2YUV)
15     yuv[:, :, 0] = cv2.equalizeHist(yuv[:, :, 0])
16     enhanced_frame = cv2.cvtColor(yuv, cv2.COLOR_YUV2BGR)
17     return enhanced_frame
18
19
20 def uniform_sampling(video_path, sample_size, enhance=False):
21     cap = cv2.VideoCapture(video_path)
22     total_frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
23     step = max(1, total_frames // sample_size)

```

```

24     frames = []
25     for i in range(0, total_frames, step):
26         if len(frames) >= sample_size:
27             break
28         cap.set(cv2.CAP_PROP_POS_FRAMES, i)
29         ret, frame = cap.read()
30         if ret:
31             if enhance:
32                 frame = enhance_image(frame)
33             frames.append(frame)
34     cap.release()
35     return frames
36
37
38 def extract_features(video_frames, model_name='resnet50', cut_layer=-2):
39     device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
40     model = models.__dict__[model_name](pretrained=True)
41     if cut_layer != -1:
42         model = torch.nn.Sequential(*list(model.children())[:cut_layer])
43     model = model.to(device)
44     model.eval()
45     preprocess = transforms.Compose([
46         transforms.ToPILImage(),
47         transforms.Resize(256),
48         transforms.CenterCrop(224),
49         transforms.ToTensor(),
50         # transforms.Normalize(mean=[0.07, 0.07, 0.07], std=[0.1, 0.09,
51             0.08]),
52         transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229,
53             0.224, 0.225]),
54     ])
55     all_features = []
56     with torch.no_grad():
57         for frame in video_frames:
58             input_tensor = preprocess(frame)
59             input_batch = input_tensor.unsqueeze(0).to(device)
60             output = model(input_batch)
61             flattened_output = output.view(output.size(0), -1)
62             all_features.append(flattened_output.squeeze().cpu().numpy())
63     all_features_flat = np.concatenate(all_features, axis=0)
64     return all_features_flat
65
66
67 def process_videos(video_dir, category, sample_size, enhance=False):
68     video_features = []
69     video_labels = []
70     video_files = os.listdir(video_dir)
71     pbar = tqdm(total=len(video_files), desc=f"Processing_{category}")
72     for video_file in video_files:
73         video_path = os.path.join(video_dir, video_file)
74         uniform_frames = uniform_sampling(video_path, sample_size,
75             enhance)
76         features = extract_features(uniform_frames, cut_layer=8)
77         video_features.append(features)
78         video_labels.append(category)
79         pbar.update(1)
80     pbar.close()
81     return video_features, video_labels
82
83
84 def parse_label_file(label_file):
85     labels = {}
86     with open(label_file, 'r') as file:
87         for line in file:
88             _, label, filename = line.strip().split()

```

```

86         labels[filename] = int(label)
87     return labels
88
89
90 def process_test_videos(test_dir, labels, sample_size, enhance=False):
91     video_features = []
92     video_labels = []
93     video_files = os.listdir(test_dir)
94     pbar = tqdm(total=len(video_files), desc="Processing_Test_Videos")
95     for video_file in video_files:
96         if video_file in labels:
97             video_path = os.path.join(test_dir, video_file)
98             uniform_frames = uniform_sampling(video_path, sample_size,
99                                               enhance)
100             features = extract_features(uniform_frames, cut_layer=8)
101             video_features.append(features)
102             video_labels.append(labels[video_file])
103             pbar.update(1)
104     pbar.close()
105     return video_features, video_labels
106
107 categories = {
108     'Jump': 'EE6222_train_and_validate_2023/train/Jump/',
109     'Run': 'EE6222_train_and_validate_2023/train/Run/',
110     'Sit': 'EE6222_train_and_validate_2023/train/Sit/',
111     'Stand': 'EE6222_train_and_validate_2023/train/Stand/',
112     'Turn': 'EE6222_train_and_validate_2023/train/Turn/',
113     'Walk': 'EE6222_train_and_validate_2023/train/Walk/'
114 }
115
116 all_features = []
117 all_labels = []
118 for label, dir_path in categories.items():
119     features, labels = process_videos(dir_path, label, 10, enhance=True)
120     all_features.extend(features)
121     all_labels.extend(labels)
122
123 label_to_index = {label: index for index, label in enumerate(categories.
124     keys())}
125 all_labels = [label_to_index[label] for label in all_labels]
126
127 X_train, X_test, y_train, y_test = train_test_split(all_features,
128     all_labels, test_size=0.2, random_state=42)
129
130 svm_classifier = SVC()
131 svm_classifier.fit(X_train, y_train)
132
133 svm_predictions = svm_classifier.predict(X_test)
134 print("SVM_Accuracy:", accuracy_score(y_test, svm_predictions))
135
136 labels = parse_label_file('EE6222_train_and_validate_2023/validate.txt')
137 test_dir = 'EE6222_train_and_validate_2023/validate'
138 test_features, test_labels = process_test_videos(test_dir, labels, 10,
139     enhance=True)
140 test_features = np.array(test_features)
141 test_predictions = svm_classifier.predict(test_features)
142 print("Test_Accuracy:", accuracy_score(test_labels, test_predictions))

```