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

Anonymous authors
Paper under double-blind review

ABSTRACT

Among various video analysis tasks, human action recognition (HAR) is one of the cornerstones, which aims to recognize (classify) a human action automatically. In dark background conditions, it is more challenging to accurately identify human activity categories. This project aims to enhance video frames with very dark backgrounds using gamma correction, and then to extract spatial features from the frames using the pretrained Resnet50 model. Subsequently, the Long Short-Term Memory (LSTM) network is employed to capture the temporal sequence relationship between video frames. By integrating the temporal and spatial features of the video frames, the fused features are fed into a Support Vector Machine (SVM) model for training. Finally, the trained SVM model is used to classify the data in the validation set, providing model evaluation metrics such as precision, recall, and F1-score.

1 INTRODUCTION

Video data is one of the most common forms of data widely used in all aspects of our daily life. With the rapid growth of video data (500 hours of videos uploaded to YouTube daily alone), automatic video analysis has become a crucial task in dealing with these vast number of videos. Among various video analysis tasks, human action recognition (HAR) is one of the cornerstones, which aims to recognize (classify) a human action automatically. The emergence of various large-scale video datasets, along with the continuous development of deep neural networks have vastly promoted the development of HAR, with increasing application in diverse fields, e.g., security surveillance, autonomous driving, and smart home.

Despite the rapid progress made by current HAR research, most research aims to improve the performance on existing HAR datasets constrained by several factors, one of which concerns the fact that videos in existing datasets are shot under a non-challenging environment, with adequate illumination and contrast. This leads to the observable fragility of the proposed methods, which are not capable to generalize well to adverse environments, including dark environments with low illumination. Take security surveillance as an example: automated HAR models could play a vital role in anomaly detection. However, anomaly actions are more common at nighttime and in dark environments, yet current HAR models are obscured by darkness, and are unable to recognize actions effectively. It is therefore highly desirable to explore methods that are robust and could cope with dark environments.

2 METHODOLOGY

2.1 GAMMA CORRECTION

Gamma correction is a commonly used technique in image processing to adjust the brightness of an image. It is based on the fact that the human visual system perceives brightness in a nonlinear manner, so adjusting the brightness of an image through a nonlinear transformation can make it appear more natural to the human eye. The essence of gamma correction is to apply a nonlinear transformation function, usually a power law transform, to each pixel value of the image.

Mathematical Expression of Gamma Correction:

$$V_{out} = A \cdot V_{in}^{\gamma}$$

where V_{in} is the input pixel value, V_{out} is the output pixel value, $\gamma$is the gamma value, and A is a constant (usually set to 1). For 8-bit images, the range of V_{in} and V_{out} is typically [0, 255]

2.2 UNIFORM SAMPLING

Uniform sampling is a systematic sampling method where each item in the population has an equal chance of being selected, and the selections are spaced evenly across the population. In the context of data processing or statistical analysis, it often refers to the practice of selecting samples at regular intervals. This method is particularly useful when dealing with large datasets or continuous data streams where it's impractical to analyze every single data point.

Key Characteristics:

1. Equal Probability: Every item in the dataset has the same probability of being selected
2. Even Spacing: Samples are chosen based on a fixed interval, ensuring a uniform distribution across the entire dataset.
3. Simplicity: It's straightforward to implement, especially when the population size and desired sample size are known.

2.3 RANDOM SAMPLING

Random sampling is a technique used to select a subset of data, or "sample," from within a statistical population in such a way that each member of the subset has an equal probability of being chosen. It is one of the most basic forms of probability sampling and is considered a cornerstone of statistical analysis because it helps to ensure that the sample is representative of the population, thereby minimizing bias

Key Characteristics:

1. Equal Probability: Each member of the population has an equal chance of being included in the sample, unlike in stratified or convenience sampling.
2. Independence: The selection of one sample does not influence or alter the chance of selecting another sample.
3. Randomness: The selection process is entirely random, often utilizing random number generators or drawing lots.

2.4 LONG SHORT-TERM MEMORY

Long Short-Term Memory (LSTM) is a special type of Recurrent Neural Network (RNN) used for processing and predicting sequence data tasks. From the perspective of video analysis, it offers a powerful method for capturing the temporal dynamics and long-term dependencies within video data. Essentially, a video is a collection of consecutive frames, each frame being an independent image, and the entire sequence collectively describes visual events or behaviors over a period of time. Traditional image processing techniques typically only handle single frames and are unable to capture this temporal continuity and dependency. However, LSTMs, with their unique recurrent network structure, are capable of processing sequence data, making them particularly suited for video analysis tasks.

Key advantages of LSTM in video analysis:

1. Capturing Time Series Information: LSTMs are capable of effectively remembering and utilizing long-term temporal information in video frame sequences through their internal states and gating mechanisms, which is crucial for understanding video content.
2. Handling Videos of Different Lengths: Videos can vary in length, and LSTMs can flexibly process input sequences of different lengths, making them applicable to video analysis tasks of various scales and durations.

3. Understanding Complex Actions: By learning long-term dependencies, LSTMs are able to recognize and understand complex human behaviors and actions occurring in videos, even when these actions span multiple time steps.

2.5 SUPPORT VECTOR MACHINE

Support Vector Machine (SVM) is a powerful supervised learning algorithm widely used for recognizing and classifying patterns in video content. Although originally designed for binary classification problems, SVM can effectively address multi-class video classification issues through the use of one-vs-all or one-vs-one strategies.

For video features that are not linearly separable, SVM can employ kernel functions (such as the Radial Basis Function, RBF) to project the data into a higher-dimensional space where a hyperplane can effectively separate different categories.

Although basic SVM is a binary classifier, it can be extended to multi-class video classification tasks using one-vs-all or one-vs-one strategies. The one-vs-all strategy trains an SVM classifier for each category, while the one-vs-one strategy trains a classifier for every pair of categories.

SVM performs well with small training datasets and has good generalization capabilities for high-dimensional data. With the appropriate kernel function, it can effectively handle nonlinear problems

3 EXPERIMENTS

3.1 DATA PREPROCESSING

3.1.1 VIDEO FRAME EXTRACTION

I use the 'train.txt' file to obtain the paths and labels of videos in the training set. Then, using the cv2 library, perform frame extraction on each video through both Uniform Sampling and Random Sampling, and save the extracted frames in two separate lists.

In human action recognition, uniform sampling refers to extracting frames from a video at equal time intervals. For instance, if a video is recorded at 30 fps, we might extract one frame every second. This method ensures a uniform distribution of sampled frames over time and is suitable for scenarios where the actions are consistent and uniformly distributed throughout the entire video.

Random sampling, on the other hand, involves extracting frames from the video at random intervals. This method does not take into account the temporal spacing between frames, which may result in more frames being sampled from certain parts of the video and fewer from others. This approach might be better suited to situations where the actions vary irregularly, or where we wish to enhance the model's generalization capabilities through randomness.

If the actions in the video are uniformly distributed over time, uniform sampling should be considered first; if the movements are fast and random, or if a higher generalization capability of the model is desired, then random sampling might be a better choice.

3.1.2 IMAGE ENHANCEMENT.

I set the range of gamma values from 0.8 to 2.2 first, and then apply gamma correction to the extracted video frames. By comparing the brightness variations of the images, we ultimately determine that the value of gamma to be used is 1.2.

4 FEATURE EXTRACTION.

1. Model Initialization: A pre-trained ResNet50 model is loaded with weights, and its last fully connected layer is replaced by a custom layer (which is then commented out), which indicates that the model is being used for feature extraction rather than for final classification.

   ResNet50 is a pre-trained deep convolutional neural network model that has been trained on the ImageNet dataset, acquiring the ability to extract features from millions of images.

ResNet50 is favored for several reasons: on one hand, its residual learning framework effectively addresses the vanishing gradient problem in deep networks; on the other hand, it has demonstrated excellent performance across a variety of computer vision tasks, including image classification, detection, and localization. Through transfer learning, we can leverage these pre-trained features as a starting point, adapting to specific video analysis tasks, which usually saves a significant amount of time and resources compared to training a model from scratch.

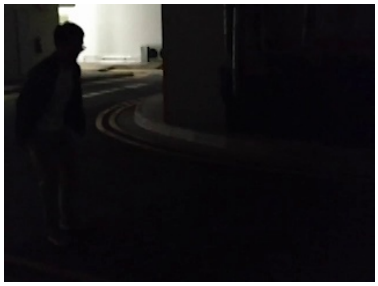
Within the ResNet50 model, if we remove the top fully connected layer, we can obtain a 2048-dimensional feature vector. This feature vector provides a rich representation for each image frame, capturing important visual features of the image.

2. Model Evaluation Mode: The model is set to evaluation mode using model.eval(), which is important to ensure that certain layers like dropout layers and batch normalization layers behave correctly during inference.
3. Preprocessing Pipeline: A series of transformations are defined to preprocess the frames before feeding them into the model. This includes resizing the frame to 256 pixels, center cropping it to 224 pixels, converting it to a tensor, and normalizing it with specified mean=[0.485, 0.456, 0.406] and standard deviation=[0.229, 0.224, 0.225].For video frames that have not undergone gamma correction, mean=[0.07,0.07,0.07] and standard deviation =[0.1,0.09,0.08]
4. Feature Extraction Loop: Within the torch no grad context (which disables gradient calculations for efficiency during inference), the function loops through a sequence of frames. Each frame is converted from a NumPy array to a PIL image, preprocessed, and unsqueezed to add a batch dimension, making it a 4D tensor expected by the model.
5. GPU Compatibility: If a GPU is available , the input tensor and model are moved to the GPU for faster computation.
6. Model Forward Pass: The preprocessed frame is passed through the model to extract features. The output is then moved back to the CPU and flattened into a one-dimensional tensor before being appended to the list
7. After all frames have been processed, the features are stacked to create a single tensor. This tensor represents the extracted features from the video frames.
8. LSTM Feature Extraction: The spatial features are then passed through an LSTM network (defined by a class LSTMFeatureExtractor) for further processing to capture temporal dependencies.
9. Feature Storage: Finally, the extracted features are intended to be stored in a list

5 CLASSIFIER TRAINING AND EVALUATION

Convert the list containing feature vectors and the list containing labels into numpy arrays, then use StandardScaler to standardize the feature vector array, scaling all features to the same level. Next, apply PCA to the standardized feature vector array to reduce dimensions while preserving 0.95 of the variance of the original data, in order to eliminate the interference of noise on the experimental results. Subsequently, create a random forest classifier using RandomForestClassifier, setting the number of trees to 50. The random forest model is trained on the dataset after dimensionality reduction through principal component analysis. Once the model is trained, the importance score of each feature can be obtained. Sort the feature importance scores and reverse the indices so that the most important features are at the forefront. Then select the top 30 features with the highest importance scores to form a new dataset. Define the parameter grid for SVM, with parameter C ranging from 0.001 to 10, and set the kernel to linear. Initialize the classifier, and use grid search with 10-fold cross-validation to find the best parameters and refit the entire training set.

Perform the same data preprocessing, feature extraction, and feature selection on the validation set, then input the data from the validation set into the already trained SVM model, and use the predict method to generate an array of labels predicted by the model. Afterward, use the classification report method to obtain the model's scores.



(a) no gamma correct

(b) gamma correct

```
D:\Users\admin\anaconda3\envs\NLP\python.exe D:\HAR\main.py
              precision    recall  f1-score   support

           0       0.07      0.08      0.07       214
           1       0.14      0.21      0.17       224
           2       0.13      0.03      0.05       256
           3       0.18      0.18      0.18       268
           4       0.00      0.00      0.00       206
           5       0.14      0.28      0.18       243

    accuracy                           0.13      1411
   macro avg       0.11      0.13      0.11      1411
weighted avg       0.11      0.13      0.11      1411


进程已结束,退出代码0
```

(c) no gamma correct with uniform sampling

```
              precision    recall  f1-score   support

           0       0.29      0.35      0.32        17
           1       0.17      0.20      0.18        15
           2       0.21      0.20      0.21        15
           3       0.31      0.25      0.28        16
           4       0.37      0.41      0.39        17
           5       0.18      0.12      0.15        16

    accuracy                           0.26        96
   macro avg       0.25      0.26      0.25        96
weighted avg       0.26      0.26      0.26        96


进程已结束,退出代码0
```

(d) gamma correct with uniform sampling

```
              precision    recall  f1-score   support

           0       0.10      0.12      0.11        17
           1       0.20      0.20      0.20        15
           2       0.17      0.20      0.18        15
           3       0.12      0.12      0.12        16
           4       0.18      0.12      0.14        17
           5       0.07      0.06      0.07        16

    accuracy                           0.14        96
   macro avg       0.14      0.14      0.14        96
weighted avg       0.14      0.14      0.13        96
```

(e) no gamma correct with random sampling

```
              precision    recall  f1-score   support

           0       0.15      0.12      0.13        17
           1       0.13      0.13      0.13        15
           2       0.27      0.27      0.27        15
           3       0.16      0.19      0.17        16
           4       0.32      0.35      0.33        17
           5       0.20      0.19      0.19        16

    accuracy                           0.21        96
   macro avg       0.20      0.21      0.21        96
weighted avg       0.21      0.21      0.21        96
```

(f) gamma correct with random sampling

6 CONCLUSION

From the image, it can be observed that after applying gamma correction to video frames with dark backgrounds, the classification performance of the SVM is enhanced. Furthermore, between the two sampling methods used in this experiment, uniform sampling slightly outperforms random sampling.

A APPENDIX

```python
import os
import cv2
import numpy as np
import torch
import torchvision.models as models
import torchvision.transforms as transforms
from PIL import Image
from torch import nn
from torchvision.models.resnet import ResNet50_Weights
from sklearn.svm import SVC
from sklearn.metrics import classification_report
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
from sklearn.decomposition import PCA
from sklearn.ensemble import RandomForestClassifier
```

```python
train_data_path = './EE6222 train and validate 2024/train.txt'
validate_data_path = './EE6222 train and validate 2024/validate.txt'
path_train = './EE6222 train and validate 2024/train/'
path_validate = './EE6222 train and validate 2024/validate/'
video_paths_train = []
video_paths_validate = []
path_o_train = './original_train/'
path_v_validate = './original_validate/'
save_path_train = './gamma_train/'
save_path_validate = './gamma_validate/'
gamma1 = 1.2
path_label_train = {}
train_label = []
validate_label = []
path_label_validate = {}
```

```python
with open(train_data_path, 'r') as file:
    for line in file:
        components = line.strip().split('\t')
        video_id, class_label, video_path = components
        video_paths_train.append(path_train + video_path)
        path_label_train[path_train + video_path] = int(class_label)

with open(validate_data_path, 'r') as file:
    for line in file:
        components = line.strip().split('\t')
        video_id, class_label, video_path = components
        video_paths_validate.append(path_validate + video_path)
        path_label_validate[path_validate + video_path] = int(class_label)

frames_train = []
all_features_train = []
frames_validate = []
all_features_validate = []
```

```python
def adjust_gamma(image, gamma):
    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 uniform_sample(video_path1, list1, sampling_rate):
    cap = cv2.VideoCapture(video_path1)
    total_frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
    for j in range(0, total_frames, sampling_rate):
        # Set the current frame position of the video file
        cap.set(cv2.CAP_PROP_POS_FRAMES, j)
        ret, frame = cap.read()
        if ret:
            gamma_corrected = adjust_gamma(frame, gamma=gamma1)
            rgb_frame = cv2.cvtColor(gamma_corrected, cv2.COLOR_BGR2RGB)
            list1.append(rgb_frame)
    cap.release()
```

```python
def random_sampling(video_path1, list1, num_samples):
    cap = cv2.VideoCapture(video_path1)
    total_frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
    frame_indices = np.random.choice(total_frames, size=num_samples, replace=False)
    frame_indices.sort()
    for idx, frame_index in enumerate(frame_indices):
        # Set the current frame position of the video file to the sample index
        cap.set(cv2.CAP_PROP_POS_FRAMES, frame_index)
        ret, frame = cap.read()
        if ret:
            rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
            list1.append(rgb_frame)
    cap.release()
```

```python
class LSTMFeatureExtractor(nn.Module):
    def __init__(self, input_size, hidden_size, num_layers):
        super(LSTMFeatureExtractor, self).__init__()
        self.lstm = nn.LSTM(input_size, hidden_size, num_layers, batch_first=True)

    def forward(self, x):
        out, (hn, cn) = self.lstm(x)
        return hn[-1]
```

```python
def extract_features(frame_s):
    model = models.resnet50(weights=ResNet50_Weights.DEFAULT)
    model = torch.nn.Sequential(*(list(model.children())[:-1]))
    model.eval()

    preprocess = transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
    ])

    with torch.no_grad():
        list1 = []
        for frame1 in frame_s:
            frame = Image.fromarray(frame1)
            input_tensor = preprocess(frame).unsqueeze(0)
            if torch.cuda.is_available():
                input_tensor = input_tensor.to('cuda')
                model.to('cuda')
            output = model(input_tensor)
            output = output.cpu().flatten()
            list1.append(output)

        spatial_features_tensor = torch.stack(list1)
        input_size = 2048  # Size of input features
        hidden_size = 512  # LSTM hidden layer size
        num_layers = 3
        lstm_feature_extractor = LSTMFeatureExtractor(input_size, hidden_size, num_layers)
        lstm_features = lstm_feature_extractor(spatial_features_tensor)
        return lstm_features
```

```python
def main():
    for i in video_paths_train:
        list1 = []
        # uniform_sample(i, list1, 5)
        random_sampling(i, list1, 6)
        frames_train.append(list1)
        train_label.append(path_label_train[i])

    for i in frames_train:
        all_features_train.append(extract_features(i))

    x_train = np.array(all_features_train)
    y_train = np.array(train_label)
    scaler = StandardScaler()
    features_scaled = scaler.fit_transform(x_train)
    pca = PCA(n_components=0.95)  # 保留95%的方差
    X_pca = pca.fit_transform(features_scaled)
    rf = RandomForestClassifier(n_estimators=50)
    rf.fit(X_pca, y_train)
    importances = rf.feature_importances_
    indices = np.argsort(importances)[::-1]
    X_selected = X_pca[:, indices[:50]]
    param_grid = {
        'C': [0.001, 0.01, 0.1, 1, 10],
        'kernel': ['linear']
    }
```

```python
    clf = SVC()
    grid_search = GridSearchCV(clf, param_grid, refit=True, verbose=2, cv=10)
    grid_search.fit(X_selected, y_train)
    print("最佳參數組合: ", grid_search.best_params_)
    print("最佳模型分數: ", grid_search.best_score_)
    for i in video_paths_validate:
        list1 = []
        # uniform_sample(i, list1, 5)
        random_sampling(i, list1, 6)
        frames_validate.append(list1)
        validate_label.append(path_label_validate[i])

    for i in frames_validate:
        all_features_validate.append(extract_features(i))

    x_validate = np.array(all_features_validate)
    y_validate = np.array(validate_label)
    scaler = StandardScaler()
    features_scale = scaler.fit_transform(x_validate)
    pca = PCA(n_components=0.95)  # 保留95%的方差
    X_pca = pca.fit_transform(features_scale)
    rf = RandomForestClassifier(n_estimators=50)
    rf.fit(X_pca, y_validate)
    importances = rf.feature_importances_
    indices = np.argsort(importances)[::-1]
    X_selected = X_pca[:, indices[:50]]
    predictions = grid_search.predict(X_selected)
    print(classification_report(y_validate, predictions, zero_division=1))


if __name__ == "__main__":
    main()
```