Motion Recognition of simple exercises using Computer Vision solutions

UP935745, *Meng Computer Science*

Abstract— This research explores the application of computer vision techniques for recognizing physical body exercises in real-time using RGB-D data. The study adopts a Bag-of-Words representation combined with SIFT features to extract discriminative information from exercise video frames. The data is then trained on Support Vector Machine (SVM) and Random Forest classifiers. The experiments demonstrate that the proposed approach achieves moderate accuracy in exercise recognition and counting. However, challenges arise from dataset limitations and pre-processing issues. Future work could focus on improving data acquisition, pre-processing methods, and feature extraction to enhance model performance.

Index Terms— Deep Learning, RGB-D Data, Bag-of-Words, SIFT Features, SVM, Random Forest, Motion History Images, Exercise Recognition, Computer Vision, Real-time.

1 Introduction

The fitness industry has experienced remarkable growth in recent times. The increasing awareness of healthy living has attracted a considerable influx of new customers seeking to embrace an active lifestyle. Despite facing challenges during the recent health crisis, the industry continues on an upward trajectory, poised for further expansion. However, not everyone can keep up with the rising costs of gym memberships and personal trainers, particularly for newcomers. Consequently, many individuals opt for online fitness classes or engage in home-based workouts.

The internet offers a vast repository of educational fitness videos and exercise tutorials. However, self-assessment of performance remains challenging without proper guidance and feedback. In this context, this report aims to introduce computer vision and machine learning solutions that could allow individuals to assess their exercises effectively from the comfort of their homes. Specifically, this project endeavours to introduce a human body motion recognition model, capable of identifying various exercises. The primary objective is to develop a machine learning model that can detect these exercises from sequences of RGB-D images. The model will be trained on existing data gathered from public access datasets online. Additional goal of this project is to measure the impact of classification of data from two separate datasets in the same classifier.

2 Existing Solutions

2.1 Project 1

The first project is titled the Recognition and Repetition Counting for Complex Physical Exercises with Deep Learning. This project attempts to build a machine learning model to recognise and count repetitions of various complex exercises that can differ slightly based on the variation used.

The project utilises deep learning, specifically Convolutional Neural Networks (CNNs), to extract meaningful features from exercise frames. The frames are preprocessed to remove background noise and focus on the exercise regions of interest. CNNs are employed to learn relevant patterns and features from the preprocessed frames to achieve high accuracy in recognition and repetition counting.

This is achieved using RGB-D images and frame data. The depth image data is filtered and masked to remove unwanted noise. Temporal Convolutional Networks (TCN) are integrated into the model which allow them to account for sequential motion data that is inherently present in exercises. The developed model in this project was evaluated using 5-fold cross validation and has achieved a test accuracy of an impressive 99.96%

2.2 Project 2

Similarly, to project 1, this project focuses on real-time recognition and repetition counting of physical exercises, targeting applications in fitness tracking and personal training. The methodology of this implementation falls in line with the concepts and algorithms studied throughout this computer vision module.

Like project 1, the authors of this research project use preprocessing techniques on the acquired RGB-D data to allow for more accurate training of the classification model. However, unlike the other implementation. The SIFT extraction algorithm is used to extract key features of each frame, which are then input into the Bag of Words (BoW) representation model. Following that, the BoW is used to train the SVM classifier to build the recognition model.

This implementation methodology has scored a 98.4%.

3 Approach

3.1 Approach Outline

Based on review of existing solutions, the following list of steps has been identified:

1. Data acquisition: For the purposes of this research project, video or image RGB-D data must be acquired. The depth attribute of this data type is the core variable that computer vision solutions rely on. A dataset must be acquired consisting of numerous repetitions of each class of defined exercises.

2. Data preprocessing - depth information files must be pre-processed for the purposes of boosting the efficacy of the training algorithms. This is done through the removal of unwanted elements of each depth image such as background and objects.

3. Feature Extraction - frame data must be analysed using feature extraction algorithms such as SIFT to obtain descriptors and keypoints that identify unique patterns of depth within the images.

4. Feature Representation - extracted feature data of frames must be concatenated and represented using tools such as the Bag of Words and MHI. This will allow us to store movement and displacement information of a sequence of frames in 1 data point.
5. Classification Model Training - feature data for each repetition of each exercise must be input into and and tested by classification algorithms such as SVM. The resulting model should be tested and fine tuned to improve its accuracy.

3.2 Data Acquisition

There are limitations when it comes to the data acquisition for this project. One of them is the lack of recording hardware required to record a unique dataset specific for the purposes of this project. The other is the dependency on the availability of public datasets that contain data relevant to the classification model that is being explored in this project.

There are few datasets that match the description, however there are still some datasets that involve human motion that we can acquire some data from. Namely, it is the research database presented in Berkeley Multimodal Human Action Database (MHAD) [5] and a large public dataset provided by ROSE Lab at the Nanyang Technological University, Singapore and its researchers [3] [4].

Data acquired from both are a series of **.ppm** and **.pgm** files extracted from video files recorded using Microsoft Kinect with ***640x480*** resolution at ***30 FPS***. The acquired data are from the following categories:

- Jumping
- Lateral raises
- Jumping Jacks
- Other

The ".pgm" format files contain the depth data attribute . Thus, most of the operations explored in this report will be relating to manipulation of the .pgm files.

Data acquired from MHAD consists of 4 recordings of the following exercises:

jumping, lateral_raises and jumping_jacks.

Each of the recordings contains 5 repetitions of its respective exercise class. As there are multiple repetitions in the same recording video, not every repetition is uniform. Through manual inspection, each repetition of these exercises can be closely approximated with 30 sequential frames (or 1 second of real time footage) the start and end of which have been manually selected. This results in 20 data points for each assigned class.

Other contains data points that are not related and is meant to represent all the other exercises not related to the ones mentioned above. Additionally, most of the data points belonging to "other" are obtained from ROSE lab [4]

3.3 Data Preparation

As discussed earlier, using various noise reduction algorithms can help ensure the efficiency and efficacy of our feature extraction algorithms.

The pre-processing methods that were applied in order were:

1. **Binary Masking**
2. **data normalisation**
3. **data type conversion to an 8 bit format**
4. **Noise Filtering: MedianBlur**

As our objective involves using SIFT and BoW, depth frames have to be 8uint prior to being used in a feature extraction algorithm. However, as converting from 16uint to 8uint reduces the number of possible depth layers a frame can have, it is important to modify the original frame to attempt to remove unwanted elements. Additionally, without pre-processing, the 8uint conversion outputs an undesirable depth file as can be seen in Fig 1

Fig 1. Depth frame conversion without normalisation

Fig 1. Depth frame conversion without normalisation

Thus, a 8uint Binary Mask is applied to remove a large number of unwanted elements. Through trial and error, a minimum and maximum value for a range of depths is acquired. The range is layers **85 ± 20** This is the range within which the human is detected and thus we can assign the value of 0 to all other layers.

Next it is important to normalise the masked depth frame for its conversion to 8uint. Normalising the depth values allows for an approximate translation of depth values from a 65535-maximum number to 255. Fig 2 demonstrates the conversion without normalisation.

Fig 2 masked depth image converted from 16 uint to 8 uint

Fig 3 masked depth image converted with normalization

Finally, after normalisation is performed, a median blur filter is applied. Applying the median blur filter smooths out some of the "jagged" edges on the resulting depth image - reducing several keypoints being generated on unwanted elements. Fig 4 shows a fully preprocessed depth image with later sift keypoints drawn.

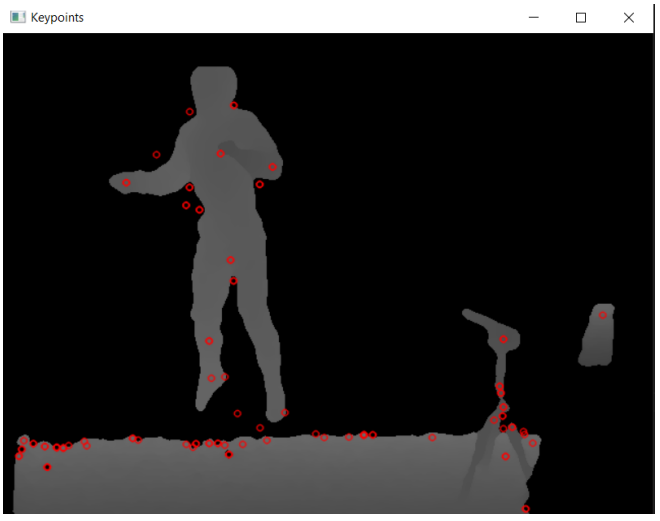


Fig 4 Masked, normalised and filtered 8 bit depth frame

These operations were performed on a list of depth frames using the following functions:

```python
def load_and_preprocess_depth_images(depth_folder):
    depth_images = []
    for file in glob.glob(os.path.join(depth_folder, '*.pgm')):
        depth_image = cv2.imread(file, cv2.IMREAD_UNCHANGED)

        # Preprocess the depth_image
        masked_depth_image = mask_and_normalise(depth_image)

        # Convert the depth_image to 8-bit unsigned integer
        depth_image = cv2.convertScaleAbs(masked_depth_image)

        denoised_depth_image = cv2.medianBlur(depth_image, 9)

        depth_images.append(denoised_depth_image)

    return depth_images
```

Fig 5. Preprocessing method

```python
def mask_and_normalise(depth_frame):
    threshold_value = 0

    # Create a binary mask based on the threshold value
    mask = (depth_frame > threshold_value).astype(np.uint8) * 255

    # Apply the mask to the depth frame using bitwise AND operation
    masked_depth_frame = cv2.bitwise_and(depth_frame, depth_frame, mask=mask)

    # Normalize the masked depth frame to the 8-bit range (0 to 255)
    normalized_depth_frame = cv2.normalize(masked_depth_frame, None, 0, 255, cv2.NORM_MINMAX)

    #focus mask
    normalized_depth_frame = filter_mask(normalized_depth_frame, 85)

    return normalized_depth_frame
```

Fig 6. Masking and normalisation function

```python
def filter_mask(depth_frame, desired_grey, tolerance=20):
    # Create a binary mask for the desired shade of grey
    lower_grey = max(0, desired_grey - tolerance)
    upper_grey = min(255, desired_grey + tolerance)
    mask = cv2.inRange(depth_frame, lower_grey, upper_grey)

    # Apply the mask to the depth frame using bitwise AND operation
    masked_depth_frame = cv2.bitwise_and(depth_frame, depth_frame, mask=mask)

    return masked_depth_frame
```

Fig 7. Mask create function

3.4 Feature Extraction (SIFT & BoW)

Feature extraction is carried out through the SIFT algorithm. Having pre-processed and acquired an array of 8 uint depth frames, one way of proceeding is through the generation of key points within each of the frames and their subsequent input into the Bag of Word trainer algorithm. This is done through the function:

extract_sift_and_bow_from_exercise()

which can be seen in Fig 8.

There are 30 frames per repetition and 20 repetitions per class. Each frame is assigned a BoW label and the corresponding SIFT descriptor is added to the BoW descriptor array. Finally, a BoW cluster and dictionary are initialised:

```
extract_sift_from_exercise(j_jacks_path, "jumping_jack")
extract_sift_from_exercise(jumping_path, "jump")
extract_sift_from_exercise(lateral_raise_path, "lateral_raise")
extract_sift_and_bow_from_exercise(squat_path, "squat")
extract_sift_and_bow_from_exercise(other_path_path, "other")

dictionary = bow_trainer.cluster()
```

```
#extracts sift and bow features from the total data for an exercise
def extract_sift_and_bow_from_exercise(exercise_folder, exercise_label):

    # Get a list of subdirectories (repetition folders) in exercise_folder
    repetition_folders = os.listdir(exercise_folder)

    count = 0

    # Process all rep samples
    print(repetition_folders)
    for repetition_folder in repetition_folders:
        depth_folder = os.path.join(exercise_folder, repetition_folder, 'depth')

        depth_images = load_and_preprocess_depth_images(depth_folder)
        #process 1 rep
        for depth_image in depth_images:
            keypoint, descriptor = sift.detectAndCompute(depth_image, None)

            if descriptor is not None and descriptor.shape[0] > 0:
                all_descriptors.append(descriptor)
                bow_trainer.add(descriptor)
            else:
                print("No descriptors extracted from this repetition",)
            # show frame

            bow_descriptor = bow_descriptor_extractor.compute(depth_image, keypoint)
            if bow_descriptor is not None and bow_descriptor.shape[0] > 0:
                dataset_bow_descriptors.append(bow_descriptor)
                dataset_bow_labels.append(exercise_label)
            else:
                print("bow descriptor error")
        count += 1
        print("rep processed", count)


    print("Exercise class processed:", exercise_label)
```

Fig 8. Feature extraction mechanism

4. CLASSIFICATION

4.1 SVM and Random Forest

Using the BoW trainer and labels that were initialised earlier, we can create and train a **Support Vector Machine (SVM) and Random Forest classifiers**. These two algorithms have been chosen for their robustness to noise errors as well as their innate mechanisms to counteract overfitting. The dataset that was employed for testing in this project has been relatively small and therefore - particularly vulnerable to overfitting errors.

For the calibration of train and test data - 70% percent of the data set for training and the remaining 30% for testing. Fig 9 shows the parameters passed to the SVM for the classification algorithm. Fig 10, in turn, shows the random forest classifier parameters.

```
# SVM
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score, cross_val_predict

SVM = SVC(kernel='rbf', gamma=0.1, C=500)
X_train = np.array(X_train)
y_train = np.array(y_train)

SVM.fit(X_train, y_train)
X_test = np.array(X_test)
n_samples = X_test.shape[0]
X_test = X_test.reshape(n_samples, -1)
y_test = np.array(y_test)

y_pred = SVM.predict(X_test)

# Accuracy
accuracy = accuracy_score(y_test, y_pred)
svm_report = classification_report(y_test, y_pred)
print("Accuracy:", accuracy)
print("SVM Classification Report:")
print(svm_report)

cmx = confusion_matrix(y_test, y_pred)
# Get the class labels from LabelEncoder
#class_labels = label_encoder.classes_

# Display the confusion matrix as a heatmap
plt.figure(figsize=(3, 3))
sns.heatmap(cmx, annot=True, fmt='d', cmap='Blues', xticklabels=classes, yticklabels=classes)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
✓ 0.4s
```

Fig 9 SVM parameters

```
# RANDOM FOREST

from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(flattened_descriptors, encoded_labels, test_size=0.3, random_state=42)

rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
rf_classifier.fit(X_train, y_train)
y_pred = rf_classifier.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

rf_report = classification_report(y_test, y_pred)
print("Random Forest Classification Report:")
print(rf_report)

cmx = confusion_matrix(y_test, y_pred)

classes = label_encoder.classes_

plt.figure(figsize=(5, 5))
sns.heatmap(cmx, annot=True, fmt='d', cmap='Greens', xticklabels=classes, yticklabels=classes)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```

Fig 10 Random Forest Parameters

5. Result Analysis

5.1 SVM and RF Initial Results

Firstly, the classifiers were trained without the addition of the "other" class. As it consists of non-uniform data due to having data from both data sets, it could potentially increase the inaccuracies and erroneous testing results. The depth images recorded for both datasets were carried out with different external parameters such as size of the room, distance from the camera and the difference in hardware. Thus, running the classifiers without the "other" classification can give us some idea about the base accuracy of the main data acquired from the MHAD dataset [5].

Accuracy: 0.6666666666666666
SVM Classification Report:

	precision	recall	f1-score	support
0	0.62	0.55	0.58	187
1	0.65	0.76	0.70	169
2	0.73	0.71	0.72	184
accuracy			0.67	540
macro avg	0.67	0.67	0.67	540
weighted avg	0.67	0.67	0.66	540

Fig 11 SVM classification report

Accuracy: 0.6444444444444445
Random Forest Classification Report:

	precision	recall	f1-score	support
0	0.56	0.51	0.53	168
1	0.65	0.73	0.69	184
2	0.71	0.68	0.70	188
accuracy			0.64	540
macro avg	0.64	0.64	0.64	540
weighted avg	0.64	0.64	0.64	540

Fig 12 Random Forest classification report

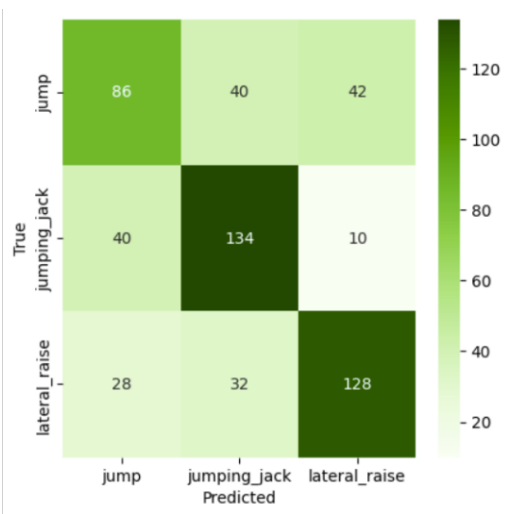


Fig 14 Random Forest Confusion matrix

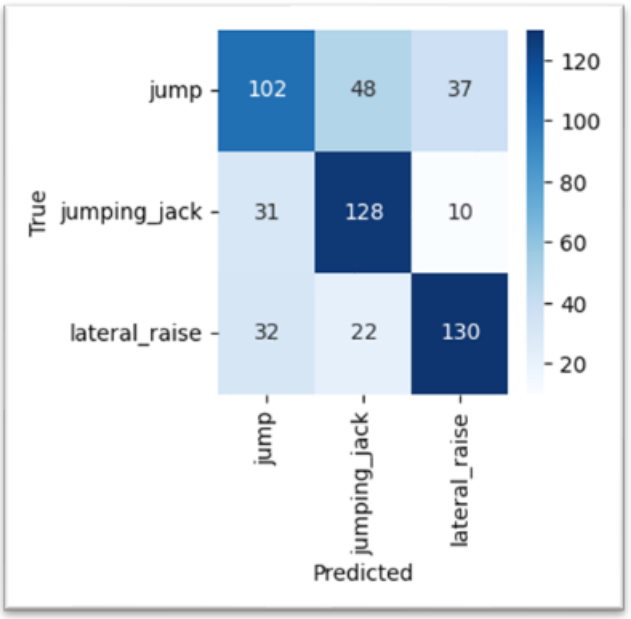


Fig 13 SVM confusion matrix

Both classifiers performed similarly, both approximately 65% accuracy with the SVM model being slightly more accurate with higher precision and f1-score values.

5.2 Cross-Validation

Both the SVM and Random Forest models were analysed with the **K-folds** sampling algorithm. Despite these two classifiers having some internal measures against the overfitting problem, it is still not invulnerable to it. Therefore, a k-folds resampler was run to verify the average accuracy of each classification method. The function for k-folds is shown below in Figure 17.

Accuracy: 0.6516666666666666
SVM Cross Validation Classification Report:

	precision	recall	f1-score	support
0	0.58	0.56	0.57	599
1	0.65	0.71	0.68	601
2	0.72	0.69	0.70	600
accuracy			0.65	1800
macro avg	0.65	0.65	0.65	1800
weighted avg	0.65	0.65	0.65	1800

Fig 15 SVM cross validation report

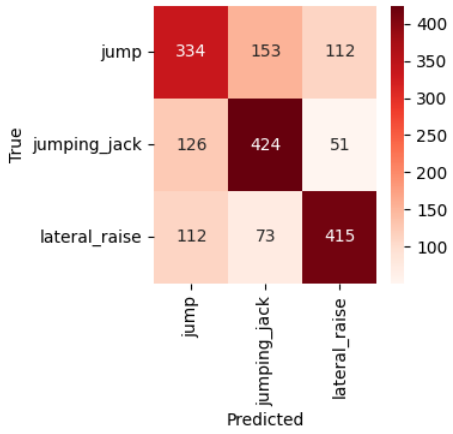


Fig 16 SVM cross validation confusion matrix

```
#resampling
def kfolds(model, x, y, splits):
    kf = KFold(n_splits=splits, shuffle=True)
    scores = cross_val_score(model, x, y, cv=kf, scoring="accuracy")
    predict = cross_val_predict(model, x, y, cv=kf)
    return scores, predict
[93]  ✓ 0.0s
```

Figure 17 K-folds algorithm

```
Accuracy: 0.6716666666666666
Random Forest Cross-Validation Classification Report:
              precision    recall  f1-score   support

           0       0.64      0.57      0.60       599
           1       0.69      0.74      0.71       601
           2       0.69      0.70      0.69       600

    accuracy                           0.67      1800
   macro avg       0.67      0.67      0.67      1800
weighted avg       0.67      0.67      0.67      1800
```

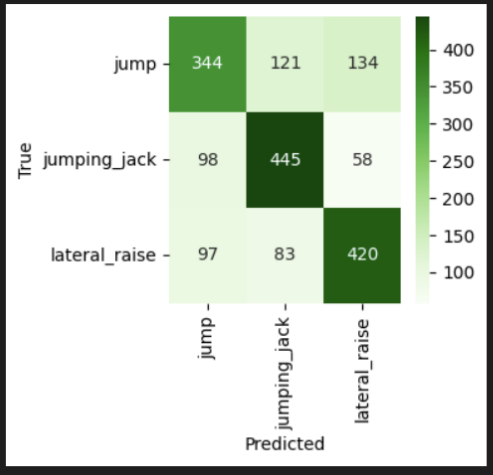


Fig 18 Random forest cross validation

Upon review of the cross validated results, the random forest model now provides better results. Regardless the overall average accuracy is below 70% - a mediocre result. Without the addition of the other class, it still appears to perform worse than expected.

5.3 Combining the datasets

After adding the "other" class consisting of 15 random 30 frame human motions from the ROSE lab [4] and 5 random human motions obtained from the main MHAD dataset [5]. Referencing Appendix B, the accuracies of the cross-validated training algorithms:

	SVM	Random Forest
Precision	0.66	0.61
Recall	0.67	0.65
F1-score	0.67	0.63
Accuracy	0.589	0.593

pproximately 59% accuracy for each model. Evaluating this result against the models' performance without the "other" class, the classifiers performed **~6% worse.**

5.4 Evaluation

The random forest model performs better than the SVM in cross-validated testing, making it the superior classifier. The system performed about averagely despite an increase in average accuracy that still falls short of the intended criterion of 70%. Even without the "other" class, the classifier outputs fall short of initial expectations, indicating that more optimisations may be needed.

These results highlight the need for further investigation and potential improvements in the training and feature representation methodologies to achieve a higher level of accuracy.

Overall, the study lays the groundwork for recognizing and classifying complex physical exercises, but further refinement and optimization are essential to achieve the desired level of reliability for practical applications.

5.5 Areas of improvement and Limitations

He observed suboptimal performance metrics imply potential shortcomings in the design and configuration of the selected feature extraction and classification algorithms. A contributing factor could be the heavy reliance on public databases for obtaining relevant RGB-D data. Unfortunately, acquiring suitable data proved challenging, resulting in limited quantities scattered across multiple datasets.
This situation introduced potential errors in calibration and training of the classifiers, hindering the overall model accuracy. Ideally, the project would have benefited from recording and calibrating new data tailored specifically for its requirements.

Another aspect that might have affected the model accuracy is the Binary Masking and preprocessing applied to the source depth data. Although an improvement over using raw data, the preprocessing approach may not have been entirely efficient. A notable issue was the presence of numerous keypoints clustered in areas of the depth image that were of little relevance to the exercise classification task. This issue hindered the model's ability to focus solely on the regions of interest.

To enhance the project's overall performance, future iterations could address these challenges by prioritising the collection of tailored data and exploring more efficient preprocessing techniques. Additionally, advanced masking methodologies could be explored to precisely isolate the relevant regions of interest and improve the accuracy of the feature extraction process. Such enhancements would likely contribute to more reliable and precise exercise recognition and repetition counting results.

REFERENCES

[1] A. Soro, G. Brunner, S. Tanner, and R. Wattenhofer, "Recognition and Repetition Counting for Complex Physical Exercises with Deep Learning," *Sensors*, vol. 19, no. 3, p. 714, Feb. 2019, doi: https://doi.org/10.3390/s19030714.

[2] T. Alatiah and C. Chen, "Recognizing Exercises and Counting Repetitions in Real Time," *arXiv:2005.03194 [cs]*, May 2020, Available: https://arxiv.org/abs/2005.03194

[3] A. Shahroudy, J. Liu, T.-T. Ng, and G. Wang, "NTU RGB+D: A Large Scale Dataset for 3D Human Activity Analysis," *arXiv:1604.02808 [cs]*, Apr. 2016, Available: https://arxiv.org/abs/1604.02808

[4] J. Liu, A. Shahroudy, M. L. Perez, G. Wang, L.-Y. Duan, and A. Kot Chichung, "NTU RGB+D 120: A Large-Scale Benchmark for 3D Human Activity Understanding," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–1, 2019, doi: https://doi.org/10.1109/tpami.2019.2916873.

[5] F. Ofli, R. Chaudhry, G. Kurillo, R. Vidal and R. Bajcsy, "Berkeley MHAD: A comprehensive Multimodal Human Action Database," 2013 IEEE Workshop on Applications of Computer Vision (WACV), Clearwater Beach, FL, USA, 2013, pp. 53-60, doi: 10.1109/WACV.2013.6474999.

7.2 Acknowledgments

Research conducted in this report relies heavily on work done previously. Various data points consisting of RGB-D format data have been acquired from public access datasets provided by the following sources:

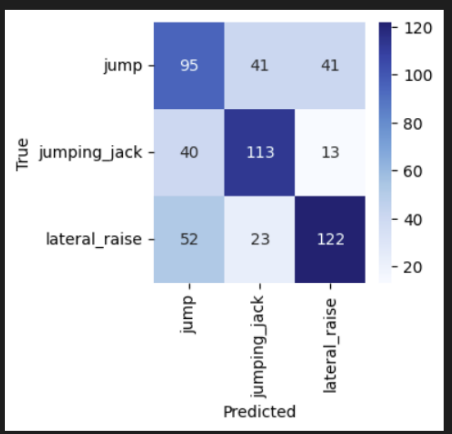
1. F. Ofli, R. Chaudhry, G. Kurillo, R. Vidal and R. Bajcsy. Berkeley MHAD: A Comprehensive Multimodal Human Action Database. In Proceedings of the IEEE Workshop on Applications on Computer Vision (WACV), 2013. [5]
2. (Portions of) the research in this paper used the NTU RGB+D (or NTU RGB+D 120) Action Recognition Dataset made available by the ROSE Lab at the Nanyang Technological University, Singapore. See references [3] and [4]

8. APPENDICES

A. Classifier results with no "other" class

Accuracy: 0.6111111111111112
SVM Classification Report:

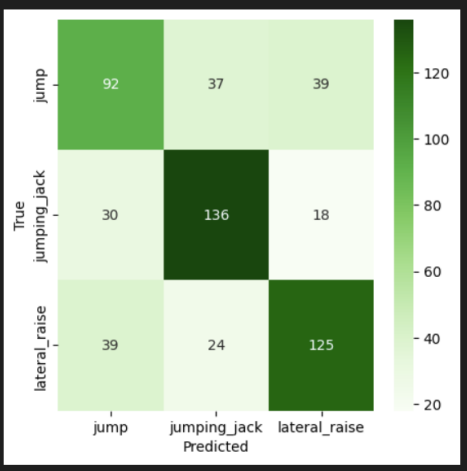
	precision	recall	f1-score	support
0	0.51	0.54	0.52	177
1	0.64	0.68	0.66	166
2	0.69	0.62	0.65	197
accuracy			0.61	540
macro avg	0.61	0.61	0.61	540
weighted avg	0.62	0.61	0.61	540



SVM

Accuracy: 0.6537037037037037
Random Forest Classification Report:

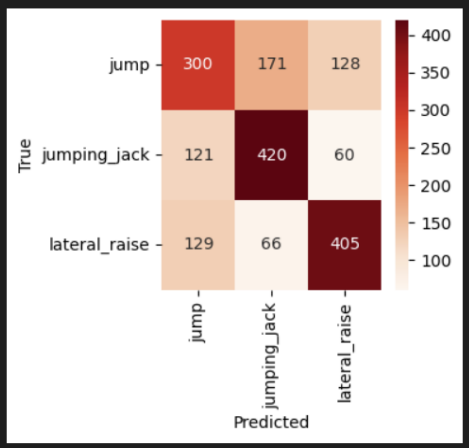
	precision	recall	f1-score	support
0	0.57	0.55	0.56	168
1	0.69	0.74	0.71	184
2	0.69	0.66	0.68	188
accuracy			0.65	540
macro avg	0.65	0.65	0.65	540
weighted avg	0.65	0.65	0.65	540



RF

Accuracy: 0.625
SVM Cross Validation Classification Report:

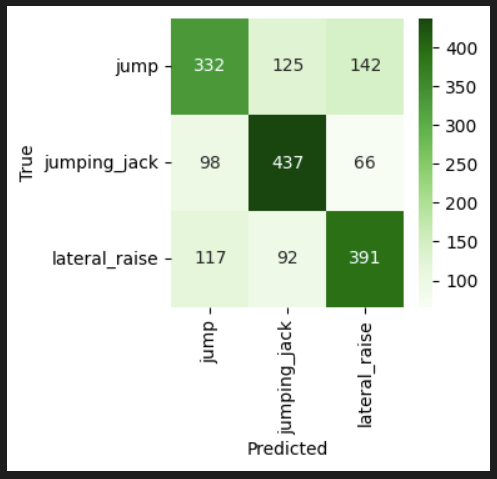
	precision	recall	f1-score	support
0	0.55	0.50	0.52	599
1	0.64	0.70	0.67	601
2	0.68	0.68	0.68	600
accuracy			0.62	1800
macro avg	0.62	0.62	0.62	1800
weighted avg	0.62	0.62	0.62	1800



SVM Cross validation

Accuracy: 0.6444444444444445
Random Forest Cross-Validation Classification Report:

	precision	recall	f1-score	support
0	0.61	0.55	0.58	599
1	0.67	0.73	0.70	601
2	0.65	0.65	0.65	600
accuracy			0.64	1800
macro avg	0.64	0.64	0.64	1800
weighted avg	0.64	0.64	0.64	1800

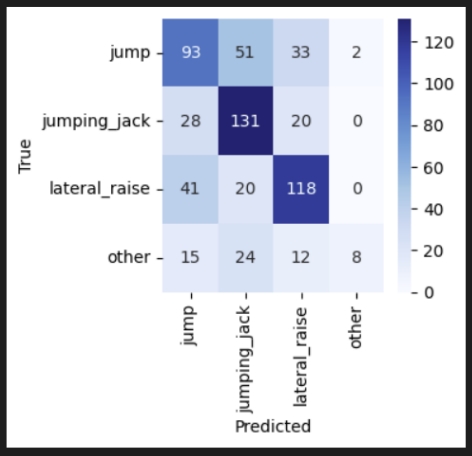


RF Cross Validation

B. Classifiers with Complete “other”

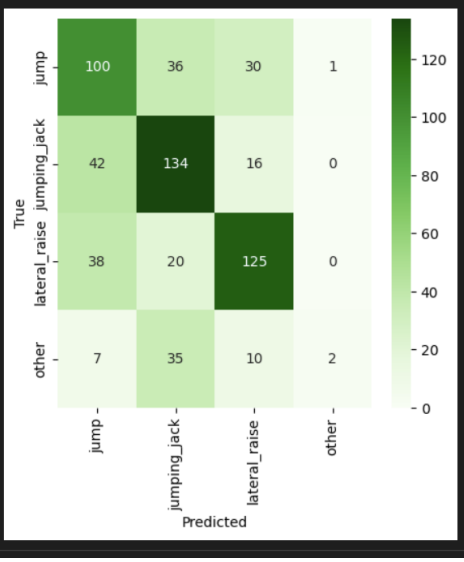
Accuracy: 0.587248322147651
SVM Classification Report:

	precision	recall	f1-score	support
0	0.53	0.52	0.52	179
1	0.58	0.73	0.65	179
2	0.64	0.66	0.65	179
3	0.80	0.14	0.23	59
accuracy			0.59	596
macro avg	0.64	0.51	0.51	596
weighted avg	0.60	0.59	0.57	596



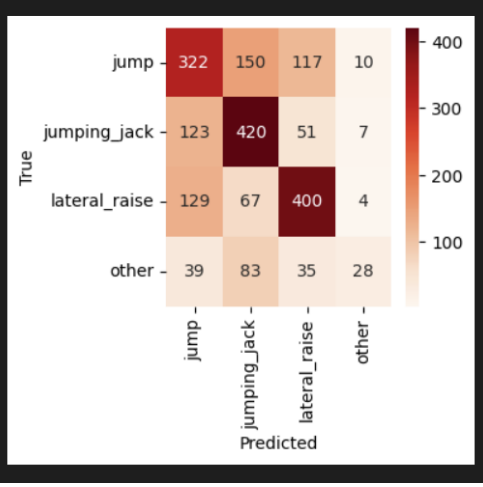
Accuracy: 0.6057046979865772
Random Forest Classification Report:

	precision	recall	f1-score	support
0	0.53	0.60	0.56	167
1	0.60	0.70	0.64	192
2	0.69	0.68	0.69	183
3	0.67	0.04	0.07	54
accuracy			0.61	596
macro avg	0.62	0.50	0.49	596
weighted avg	0.61	0.61	0.58	596



Accuracy: 0.5894206549118388
SVM Cross Validation Classification Report:

	precision	recall	f1-score	support
0	0.53	0.54	0.53	599
1	0.58	0.70	0.64	601
2	0.66	0.67	0.67	600
3	0.57	0.15	0.24	185
accuracy			0.59	1985
macro avg	0.59	0.51	0.52	1985
weighted avg	0.59	0.59	0.58	1985



Accuracy: 0.5934508816120907
Random Forest Cross-Validation Classification Report:

	precision	recall	f1-score	support
0	0.57	0.60	0.58	599
1	0.60	0.70	0.65	601
2	0.61	0.65	0.63	600
3	0.50	0.04	0.08	185
accuracy			0.59	1985
macro avg	0.57	0.50	0.49	1985
weighted avg	0.59	0.59	0.57	1985