Improving SVM Performance on Motion History Images via Feature Augmentation*

Thomas Melanson*[1]

***Abstract*— This project aims to use Hu Moments from Motion History Images and augment both the MHI and Hu moment features to produce better output on a linear classifier, namely SVM. Overall, with few datasets, data augmentation resulted in significant improvements and, with the right data, could achieve much stronger results than random and, on some features, also on par with current SVM approaches.**

I. INTRODUCTION

A. *MHI / MEI extraction*

The first step in extracting the features from the video was to create a Motion Energy Image, or MEI. MEIs are designed to capture the total amount of motion over a series of frames. The motion type can then be deduced from the shape of the resulting picture.

The video is converted to a series of difference images between frames, given by the following equation:

$D(I_i, I_{i-1}) = |I_i - I_{i-1}| > \theta$

where D is the resulting binary image (1 is the right-hand result resolves as True, 0 otherwise), I_i and I_{i-1} are the current image and the immediate prior image, and θ is a constant value determined to be the minimum non-noise value. For this project, it was found that $\theta = 10$ pixels resulted in filtering the most amount of noise while keeping the original image shape.

For a given action, the final MEI is computed as the union of every difference image within the last τ frames.

Because the raw difference image was subject to noise, filtering was used to gather the best recreation. As described later, two major tools were used to accomplish this, morphological filtering and Gaussian blurring.

However, in order to fully determine the image history, one must also know the time frame of different steps to an action. Motion History Images, or MHIs, achieve this by weighting more recent difference images higher than older ones. Given a filtered difference image D, the MHI for a given frame is defined recursively:

$$H_i = \begin{cases} max(H_{i-1}, 0) & D' > 0 \\ \tau & otherwise \end{cases}$$

B. *Hu Moment variables*

There are several major issues with using raw images as vectors for a classifier input, including:

Images as input are very brittle to changes in the image position and scale of the subject Images are extremely large in size, often consisting of hundreds or even thousands of variables An image vector would also contain background pixel data which could deter a classifier.

Because of this, this project used Hu moments to condense the image into a size-7 vector while also being scale, position, and rotation invariant.

Hu moments were introduced in 1962 by Hu et al, and are functions of the scale and position invariant η [5]. η uses the moments M_{ij} to compute the image centroids $\bar{x}, \bar{y}$ in the following steps:

$$M_{ij} = \Sigma_{i,j} x^i y^j I_{ij}$$

$$\bar{x} = \frac{M_{10}}{M_{00}}$$

$$\bar{y} = \frac{M_{01}}{M_{00}}$$

Once the moment are computed, position invariant moments can be introduced by finding the moment around the centroid:

$$\mu_{pq} = \Sigma_{p,q} (x - \bar{x})^p (y - \bar{y})^q I_{pq}$$

From there, scale invariance is introduced by dividing by a power of the total moment (denoted by μ_{00}):

$$\eta_{pq} = \frac{\mu_{pq}}{\mu_{00}^{1+\frac{p+q}{2}}}$$

From there, the Hu moments add rotational invariance with the following operations:

$h_1 = \eta_{20} + \eta_{02}$

$h_2 = (\eta_{20} - \eta_{02})^2 + 4\eta_{11}^2$

$h_3 = (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \eta_{03})^2$

$h_4 = (\eta_{30} + \eta_{12})^2 + (\eta_{21} + \eta_{03})^2$

$h_5 = (\eta_{30} - 3\eta_{12})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] + (3\eta_{21} - \eta_{03})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2]$

$h_6 = (\eta_{20} - \eta_{02})[(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] + 4\eta_{11}(\eta_{30} + \eta_{12})(\eta_{21} + \eta_{03})$

$h_7 = (3\eta_{21} - \eta_{03})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] - (\eta_{30} - 3\eta_{12})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2]$

After examining the feature vector, however, it became apparent that not all of the Hu moments were weighted equally. Higher order moments, by being raised to higher powers, became orders of magnitude smaller than lower order moments. Wong and Hall rectified this issue by taking the signed log of the moments.

Comparison of the Hu moments are done in the Results section.

Additionally, both vectorization and caching were applied to optimize runtime. Because each operation is linearly independent between features and samples, an entire dataset of MHIs or MEIs can be converted into hu moments through

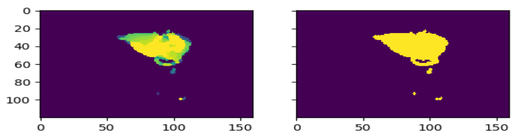


Fig. 1. Left: MHI of a boxing frame. Right: MEI of the same frame.

element-wise operations. As for caching, the M_{00}, $M10$, and M_{11} used to compute every scale invariant vector is precomputed at the start of the Hu method, and passed as an argument into the μ and η functions.

C. Tau iteration

In Bobick et al's work [3], the τ variable was optimized for each action as a hyperparameter during training. In order to prevent unnecessary calculations of MHI, the MHI for the maximum tau value was considered first. From there, any arbitrary $\tau_n < \tau_{max}$ could be computed as $H_\tau = max(H_{\tau_{max}} - (\tau_{max} - \tau), 0)$.

D. Blurring and morphological filters

The idea behind filtering the image is that changes in image due to noise are overall more isolated and fragmented than changes due to subject movement. Hence, the two methods used aim to reduce isolated / fragmented deltas in the following ways:

Gaussian blurring aims to mix pixel values in a given area, thereby smoothing a single aberrant pixel over an area and decreasing its magnitude Morphological opening adds a constraint that any notable pixel difference must be in a neighborhood of pixel differences, thereby removing isolated pixel changes in an image

Gaussian blurring is the convolution of the raw image by a Gaussian variable $G(\mu, \sigma)$. The variable is limited to a 3x3 kernel to prevent calculating variables that are too small to be significant. For this experiment, μ was set to 0 and σ was set to 1. The Gaussian variable was not emphasized because of its tendency to be a double-edged sword (removing noise at the cost of less observable image differences).

From there, morphological filtering can filter out noise by constraining active bits to be close together. Morphological opening, in particular, is a method used to remove false positives outside of a binary shape, and consists of a combination of erosion followed by a dilation. This morphological process is similar to convolution in that a kernel is dragged over an image, but works with logical operators rather than vector multiplication. For erosion, for a given pixel (x,y) to be set to 1, each pixel in a window of the kernel dimensions must be high when the pixel corresponding to the kernel centered at (x,y) is also high. For dilation, it means that at least one of the pixels corresponding to the kernel high bits are also high. In succession, erosion aims to remove the noise, then dilation recreates any shapes that were not thrown out as noise.

E. K Nearest Neighbor

K Nearest Neighbor, or kNN, which simply records the training data as a series of locuses and ranks a new training data point by its k closest neighbors, was used as a baseline due to its ease of use. This baseline algorithm was compared to Support Vector Machines, which have been used increasingly in machine learning in the past decade [1].

F. Support Vector Machines

kNN, however, is naive in that it is likely to overfit the training set. SVMs, on the other hand, use the data to create boundaries between data sets. This is generally more practical in that it provides a strong classifier without the need for a large and finely tuned training dataset. In general, the support vectors are also sparser and therefore use less space than the kNN points.

The basic original concept by Vlapnik and Siegelmann [2] was to solve for the minimum value of $\frac{1}{2}||w||^2$ which satisfies $y = \cdot w, x + b$ subject to the constraints $y \geq 1$ or $y \leq -1$ depending on the training label. The solution found was to use a weighted sum of support vectors. Because the operation is linear, calculating w and computing a dot product with x can be rearranged as a weighted sum of the dot products of support vectors x_i and the input x. Said another way:

$$w \cdot x = \Sigma_i \alpha(x_i \cdot x)$$

$$b = y - (w \cdot x)$$

where α is the weight of each support vector.

The model above was extended later to include a kernel function $K(x_i, x)$, which would use a function $\phi(x)$ to extend it to hyper dimensional space and therefore allow an SVM to become a non-linear classifier. Additionally, in modern implementations of SVM, the optimized program changes to $\frac{1}{2}||w||^2 + C\sum\gamma_i$, where larger values of C shrink the allowed margin of error in order to enforce higher classification accuracy.

Interestingly enough, in a study done by Colas[1] showed that SVMs perform worse over large datasets, on the same level as kNN and significantly slower. This could explain the overall worse performance by SVMs over the kNN data, as there are thousands of MHIs in the training dataset used.

II. Methodology

A. Visual Component

Because training and testing the data via machine learning approaches takes a long time and does not in itself provide much intuition about potential dataset flaws, using it to debug feature extraction itself is inefficient. Instead, most of the image extraction, filtering, and MHI / MEI computation employed a qualitative approach. There are sanity checks in the code base, which make sure the output to these functions are not returning extreme values such as $Nonetype$, all-zero outputs or all-one outputs, but debugging was done by

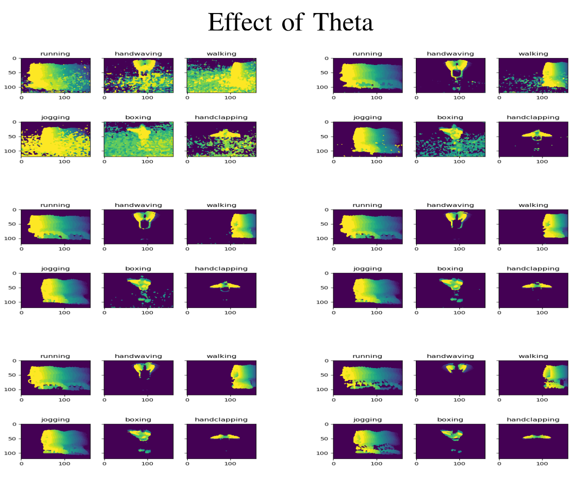


Fig. 2. Comparison of 2, 5, 7, 10, 12, and 25 pixel thetas. The maximum seems to be around 10 pixels, with too much noise below 5 and too little data above 12.

writing and observing output with the naked eye rather than through sanity check.

1) Data collection and initial filtering: Datasets in this project were implemented as a layered dictionary that could map index, action, and d value (default 'd1') to a video file. Originally, the idea was to actually reference the video images themselves, but due to memory constraints, images were generally loaded in one or two videos at a time. From there, a method retrieved a list of images from a video file.

2) MHI: As the major image pre-processing step, MHI (as well as MEI) became its own class. The object was initialized with the needed parameters, such as τ and θ, as well as some flags determining if morphological opening and closing was needed. Then, each image was added to the MHI individually, and a running track of previous values of H, as well as the previous image, was kept. Any frame that was either frame 0 of an action or did not contain an image delta above θ For this experiment, it was assumed that the Motion History Image inputs would be processed one video at a time, which meant several MHIs could be returned in succession.

3) Morphological Filtering: To test the morphological filtering step, I first sanity-checked the MEI output to make sure the filter didn't filter out every frame (all-zero) or include so much noise as to have no discernable information (all-ones). From there, I recorded MHI videos of subjects under the following filters:

- Raw thresholded output of D.
- Output subject to a $3x3$ elliptical morphological opening.
- Output subject to a $3x3$ filter closing followed by a $3x3$ elliptical filter opening

In theory, the morphological opening would filter out any remaining noise from the images. The morphological close would solidify any existing shapes and make sure the MHI would be free of holes.

Overall, when looking at slides such as Figures 5 and 6, I was convinced that a morphological close followed by an open would maintain the shape of the original images best.

B. SVM classifier

In this experiment, an SVM with an RBF classifier is optimized, allowing for nonlinear boundaries between the Hu moment data. The optimal classifier was then compared to a kNN trained on the same data.

1) Augmented Hu moments: The signed log moments proposed by Wong and Hall do bring more data to higher order weights. However, this actually results in skewing the data to weigh the higher orders more heavily. To perfectly balance the weights, I used a series of increasing roots for higher order moments, while keeping the signed idea from the paper. This signed root system was designed from the original Hu moments as follows:

$$h'_1 = h_1$$
$$h'_2 = sgn(h_2)\sqrt{|h_2|}$$
$$h'_3 = h_3^{\frac{1}{3}}$$
$$h'_4 = h_4^{\frac{1}{3}}$$
$$h'_5 = sgn(h_5) * |h_5|^{\frac{1}{6}}$$
$$h'_6 = sgn(h_6) * |h_6|^{\frac{1}{4}}$$
$$h'_6 = sgn(h_7) * |h_7|^{\frac{1}{6}}$$

As seen in Figure 3, this solution allowed the moments to be weighted equally, without relying on knowing constants such as the variable mean (which may not be readily available at runtime)

For this experiment, three different versions of the Hu moment were considered. First was the Hu moments without modification as the control group. Second was the often-used signed log scaling proposed by Wong and Hall [7]. Third was a custom set of roots designed to equalize the range of values in the dataset:

2) GridCV: When comparing features under SVM, a CV grid search was used to determine the best solution. The grid only searched through one parameter, C, over a log range of $\{1, 10, 100, 1000\}$. Linear kernels, as well as $C > 1000$, were considered, but not used in the final experiment due to timing constraints. In addition to the overall score, the penalty parameter C of the optimal solution was recorded and compared for the sake of determing how wide the class margins were.

III. RESULTS

A. Hu moment augmentation

Overall, the Hu moments were able to be better separated with the given metrics. From the figure below, it is clear that the while the log moments do emphasize the higher order points, they do so at the expense of the lower order points. The formulation implemented in the paper does the best job evening the weights of the 7 hu moments across action types.

When comparing the overall accuracy over a small dataset (MHI images gathered over the video sequences for 13 and 14), both the signed log and the signed root equations far outperformed the regular dataset (which defaulted to guessing walking every time even with C=1000). Although

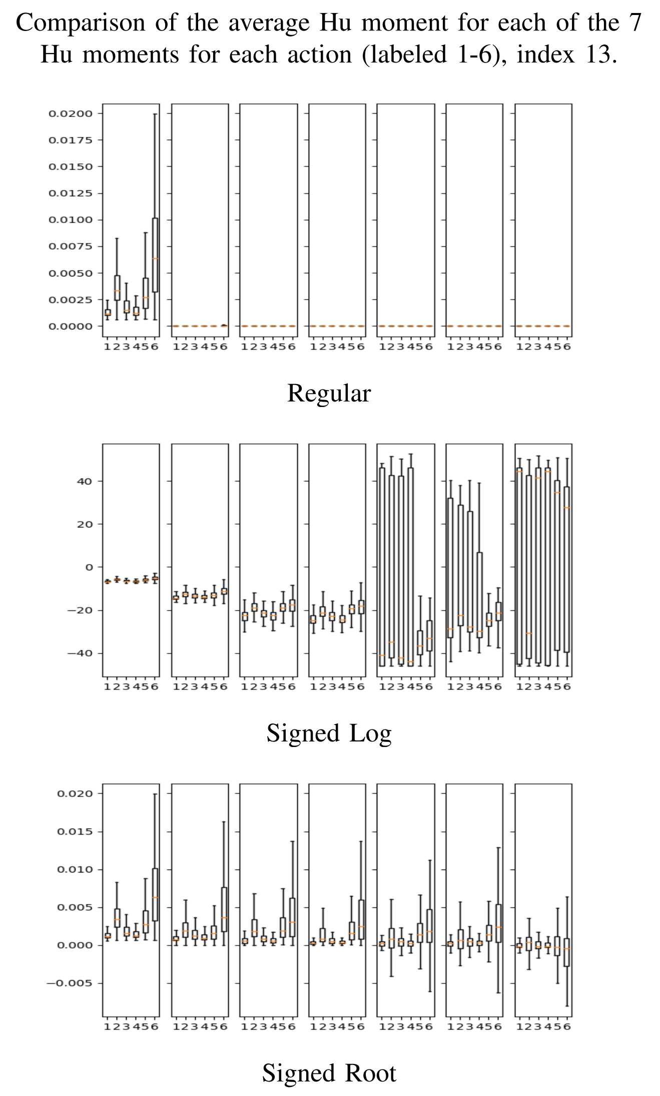


Fig. 3. Using the logarithmic scale creates a drastic increase in scale size, but it scales the higher order magnitudes more than the lower order. However, using roots can scale the values equally.

the results between the signed log and the signed root extractions are roughly similar, the Hu moments for each of the classes are roughly similar relative to each other (notice the similarity in the box plots for the signed root in part 3).

Out of the three functions, only the signed log moments were optimized at the smallest value of C. Although this may be just a scaling issue, it is important that the optimal C value to be low in order to have a generalized classifier.

B. Tau per action

In the Bobbick paper, it was mentioned that an optimal value for τ was chosen for each action [5]. To determine the values for this on the custom dataset, I used the same toy dataset from before (collecting video feed from 13 and 14), split it into a train and test dataset and trained SVC classifier with $C = 1$ and an rbf kernel. The results, below, prove that, for a global variable, $\tau = 30$ would be the overall most optimal solution, especially when classifying actions such as running and handwaving. However, shorter actions,

TABLE I
TEST ACCURACY FOR HU MOMENT OPERATIONS

Data type	Validation Score	C
Hu moment	0.231	1000
Signed Log Hu moment	0.420	1
Signed Root Hu moment	0.441	1000

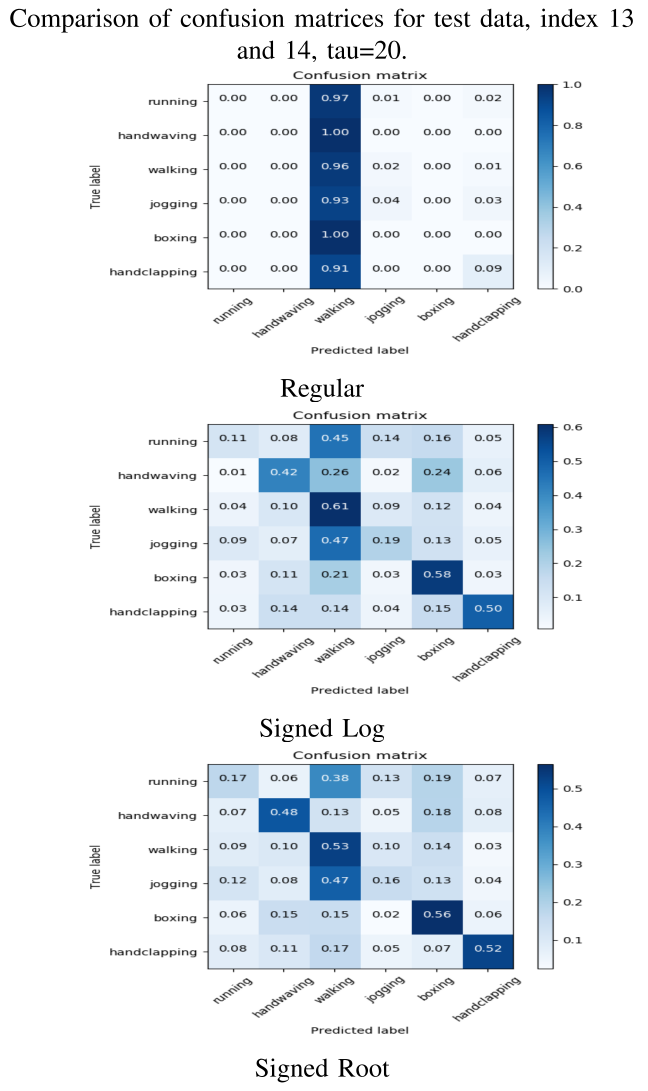


Fig. 4. As shown here, distinguishing the features more does in fact improve classification.

such as hand-clapping and boxing, peaked between $\tau = 10$ and $\tau = 20$.

However, when each of the vectors were trained with moments of MHIs trained with the optimal values, suboptimal results were obtained for both the handclapping and boxing

IV. CONCLUSION

The purpose of this paper was to use techniques to augment the dataset such that a linear classifier, such as SVM, could do as well or better than a non-linear classifier such as kNN. From the evidence given by the validation output, this seems to be the case. Data augmentation converts

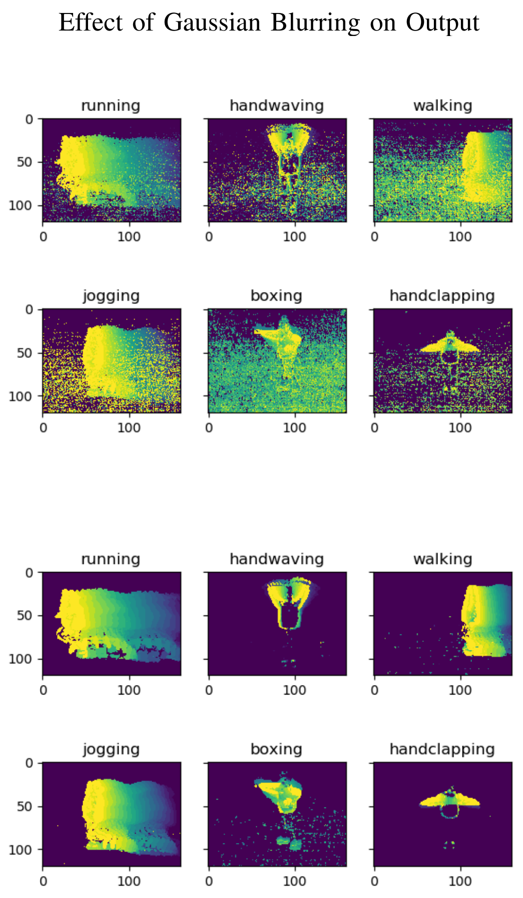


Fig. 5. As shown in the image, blurring with a Gaussian filter can reduce a significant amount of background noise, with little effect on the desired motion to capture

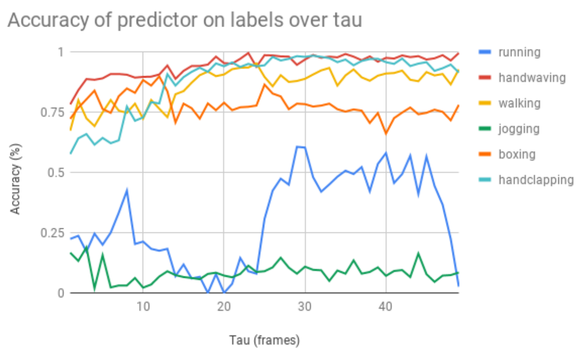


Fig. 6. From the data, one can see the peak τ for the following classes are mostly around 30, with the exception of walking and boxing.

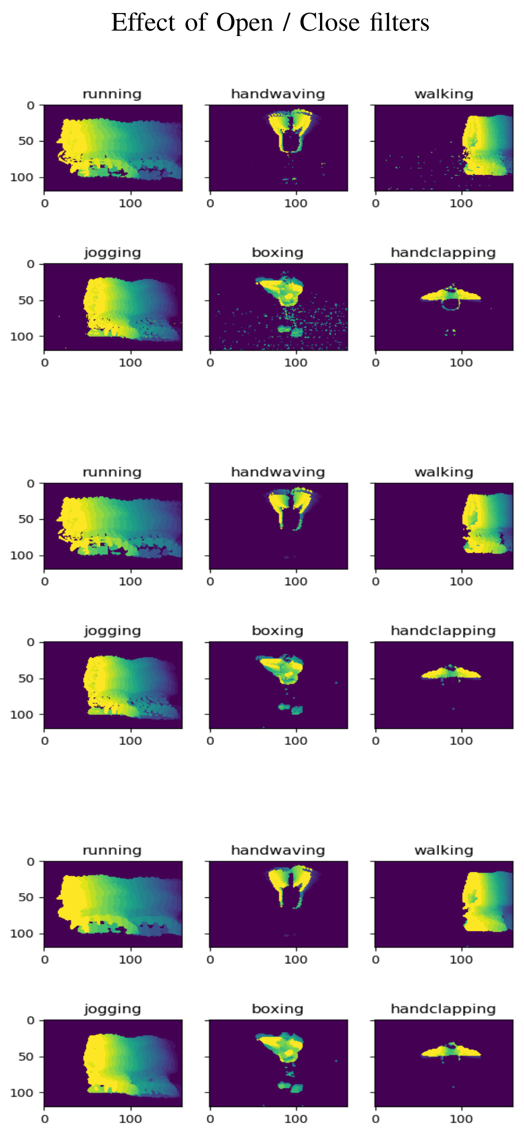


Fig. 7. An open filter can reduce the amount of noise present in the image. Additionally, using a close filter beforehand can also reduce smooth the shape of the image.

SVM from just above random test performance to 51 percent, which is a notable improvement and around the same or better than a naive kNN algorithm with the same data. As for previous work on the data, the results of this current approach do not have the accuracy reflected on some features in Schuldt and Latpev's work [6], but since the majority of the classification accuracies for each of the labels were around 60 percent, my classification performed as well on the average case.

V. DISCUSSION

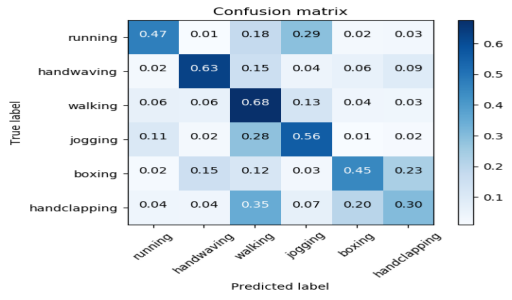
For this project, I did not realize the training sequences were labelled by frames until later on in the project. Because of this, a lot of the data was collected as a constant stream directly from a video feed. Over another iteration, I would

TABLE II

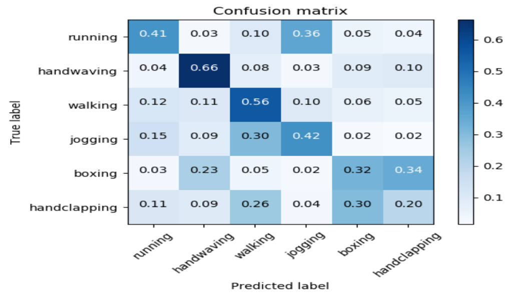
TEST ACCURACY OF SVM VS. KNN

Classifier Type	Validation Score
SVM (C=10.0, Augmented Data)	0.515
kNN (Augmented Data)	0.430

Comparison of confusion matrices for test data, index 13 and 14, tau=20.



SVM, C=10.0



kNN, k=49

Fig. 8. Table showing data-augmented SVM learning at a higher rater than kNN on the same data.

repeat several of the initial data analysis over the official action sequences rather than the parsed video feed.

Further improvement to the SVM could be done via boosting. This takes previous ”weak” classifiers and iterates over them until a more accurate classifier is made.

REFERENCES

[1] Colas, F. P. R. (2009). Data mining scenarios for the discovery of subtypes and the comparison of algorithms. Leiden Institute of Advanced Computer Science (LIACS), Faculty of Science, Leiden University.

[2] Cortes, C., & Vapnik, V. (1995). Support-vector networks. Machine learning, 20(3), 273-297.

[3] Davis, J. W., & Bobick, A. F. (1997, June). The representation and recognition of human movement using temporal templates. In Computer Vision and Pattern Recognition, 1997. Proceedings., 1997 IEEE Computer Society Conference on (pp. 928-934). IEEE.

[4] Gonzales, Rafael C. & Woods, Richard E. (2017). Digital Image Processing. Pearson.

[5] Hu, M. K. (1962). Visual pattern recognition by moment invariants. IRE transactions on information theory, 8(2), 179-187.

[6] Schuldt, C., Laptev, I., & Caputo, B. (2004). Recognizing human actions: a local SVM approach. In Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on (Vol. 3, pp. 32-36). IEEE.

[7] Wong, R. Y., & Hall, E. L. (1978). Scene matching with invariant moments. Computer Graphics and Image Processing, 8(1), 16-24.