HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION COMMUNICATION TECHNOLOGY

CAPSTONE PROJECT II REPORT:

HAND-TRACKING COMPUTER CONTROL

Name	ID
Bui Khac Chien	20220059

TABLE OF CONTENTS

Abstract 3
1. Introduction 3
1.1. Problem Description 3
1.2. Goals and Targets 5
2. Dataset and Evaluation 6
2.1. Dataset 6
2.2. Data Preprocessing 9
2.2.1. Skeleton detecting 9
2.1.2. Skeleton-guided RGB frame cropping 10
2.3. Evaluation Criteria 11
3. Methodology 12
3.1. DSCNet 12
3.2. Resnet 14
3.2.1. 3D Resnet 14
3.2.2. TSM (Temporal Shift Module) with Resnet backbone: 15
3.2.3. STMEM (Short-Term Motion Extract Module) 18
3.3. MSSTNet (Multi-scale spatial-temporal convolutional neural network) 19
3.4. Implementation & Training 23
3.4.1. Resnet 23
3.4.1.1. Resnet3D 23
3.4.1.2. STMEM + TSM-ResNet: 24
3.4.2. MSSTNET 25
4. Result 28
4.1. Experimental Setup 29
4.2. Comparative Performance 29
4.3. Model-Specific Observations 29
5. Application 30
5.1. Image enhancement 30
6. Discussion and Conclusion 32
6.1. Discussion 32
6.1.1. Comparative Analysis 32
6.1.2. Comparison with State-of-the-Art (SOTA) 32
6.2. Conclusion 33
7. References 34

Abstract

RGB-based human action recognition often underperforms in complex and dynamic environments due to sensitivity to background clutter and lighting variation. In contrast, skeleton-based data offers structural robustness, making it an effective complementary modality. As a result, multi-modal approaches combining RGB and skeleton data have gained significant traction in recent research. In this report, we investigate Dense-Sparse Complementary Network (DSCNet), which integrates RGB and skeleton modalities while maintaining low computational overhead. DSCNet adopts dense sampling for RGB frames and sparse sampling for skeleton sequences, balancing data richness and efficiency. It uses skeleton joints to guide spatial cropping of RGB frames, reducing background interference, while the Short-Term Motion Extraction Module (STMEM) compresses the RGB stream temporally. The skeleton stream is processed via a Multi-Scale Spatial–Temporal Network (MSSTNet) to capture motion dynamics. To better understand modality-specific contributions, we evaluate the RGB and skeleton branches of DSCNet independently in addition to the full model. Experiments conducted on the Jester dataset using Accuracy, Precision, Recall, and AUC demonstrate the individual strengths of each modality and the effectiveness of their fusion. The report concludes by addressing several open challenges in multi-modal action recognition and benchmarking our methods against current state-of-the-art models.

Keywords: Action recognition, RGB-based, Skeleton-based

1. Introduction

1.1. Problem Description

Action recognition constitutes a fundamental task in computer vision with significant practical applications spanning intelligent surveillance, human - computer interaction, rehabilitation, and clinical assessment. In specialized domains such as sports and physical therapy, action recognition systems offer potential as expert alternatives, providing detailed performance feedback or quantitative evaluations of recovery progress.

Recent advancements have explored diverse data modalities - such as RGB video, skeleton coordinates, depth maps, and optical flow - to enhance recognition accuracy. RGB data offer rich visual cues like color and texture but are susceptible to variations in background and lighting. In contrast, skeleton data provide robust, abstract representations of human posture and movement, less affected by environmental noise. As such, integrating both RGB and skeleton modalities has emerged as a promising direction, enabling the exploitation of complementary features.

With the rise of deep learning, action recognition models have significantly progressed. For RGB inputs, techniques such as the two-stream network and Temporal Segment Networks (TSN) have improved temporal modeling, while 3D CNNs capture spatio-temporal features more naturally. Lightweight 2D-CNN alternatives enhanced with temporal modules have also gained attention due to their computational efficiency.

For skeleton-based recognition, approaches largely fall into three categories: RNNs for sequential modeling, CNNs that convert joint movements into structured images, and GCNs that naturally capture the relational dynamics among joints. GCNs, in particular, have become a dominant paradigm due to their alignment with the graph structure of human skeletons.

However, many existing approaches primarily emphasize the integration of network components to enable deep feature interactions, while often neglecting the complementary nature of RGB and skeleton modalities at the data level. Moreover, they tend to adopt established single-modality architectures without specific optimization for multimodal contexts, which can result in suboptimal recognition accuracy and increased computational cost.

In this report, we examine the effectiveness of multimodal model DSCNet which better leverage the complementary characteristics of RGB and skeleton data. To enhance comprehension of modality-specific contributions, we assess the RGB and skeleton branches of DSCNet separately, alongside the complete model. By conducting comprehensive experiments on the 20Bn-Jester dataset and evaluating multiple performance metrics, including Accuracy, Precision, Recall, F1 and AUC, we aim to assess how these models address the limitations of conventional single-modality or weakly integrated approaches. This analysis provides insights into their strengths and limitations in the context of multimodal action recognition.

- Section 2 presents an overview of the 20Bn-Jester dataset, details the skeleton feature extraction process, and outlines the evaluation metrics used.
- Section 3 describes the methodological frameworks and training procedures.
- Section 4 analyzes and compares the performance of the selected models.
- Section 5 demonstrates the application of the model in a YouTube-based setting, including video preprocessing and related steps.
- Section 6 concludes the report by summarizing the current findings, highlighting state-of-the-art models, and discussing future research directions.

1.2. Goals and Targets

a. Primary Goals: Comprehend the Fundamentals of Action Recognition

- *Establish an Effective Pipeline:* Identify an efficient workflow to recognize gestures using video or webcam input, ensuring high accuracy and real-time performance.
- *Understand Landmark Extraction:* Learn how to extract body landmarks and their connections and understand how to store these values for use as model input or in preprocessing.
- *Analyze the DSCNet Frameworks:* Conduct an in-depth analysis DSCNet, comparing their performance using standard evaluation metrics.

b. Secondary Goals:

- *Ensure System Robustness:* Handle gesture variations, background noise and lighting changes to improve generalization.

c. Target Objectives:

- *Develop a Hand-Tracking Computer Control Application: Create a user-friendly application that allows users to control software - initially focusing on YouTube - with low latency and high accuracy.*

- *Explore Cross-Field Applications*: Extend the application's utility to other domains, including social media, robotics, and education, showcasing its versatility and real-world relevance.

2. Dataset and Evaluation

2.1. Dataset

For this project, we make use of the 20BN - Jester dataset hosted on Kaggle, which is a subset of the complete JESTER dataset. This dataset comprises a large collection of densely annotated video clips that capture individuals performing predefined hand gestures in front of a laptop or webcam. It was built through crowdsourcing efforts involving a large number of contributors. The dataset is specifically designed to support the development and training of machine learning models for hand gesture recognition - including actions such as *sliding two fingers downward*, *swiping left or right*, *drumming fingers, etc.*

The dataset includes the following components:

Three main folders:

- Train: contains 50,400 directories
- Validation: contains 7,047 directories
- Test: contains 6,981 directories

Each directory is named after a unique video_id (e.g., "1", "100", etc.) and contains 37 frames representing a single gesture video.

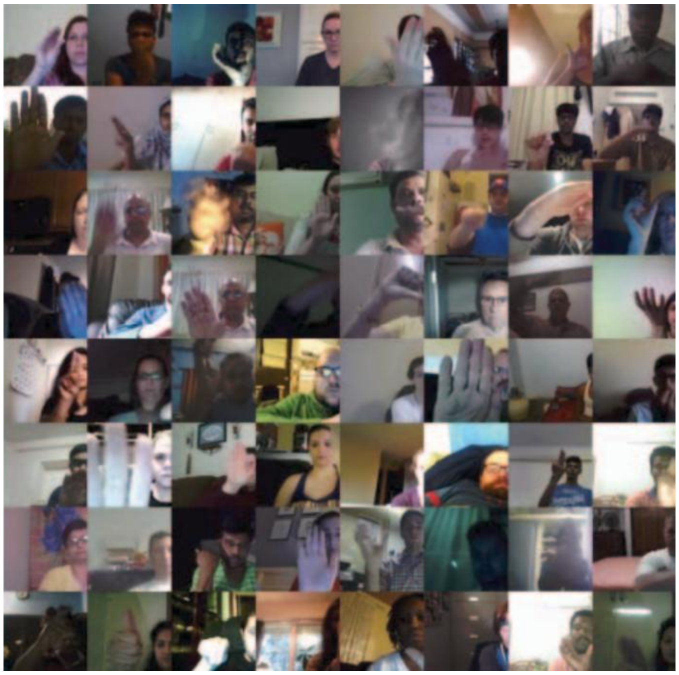


Figure 1: Examples of videos from our dataset. Each image corresponds to a randomly sampled frame from a randomly sampled video. The image shows a large variance of the appearance of peoples, background scenes and occlusion in the videos.

Three annotation files (CSV format):

- Train.csv, Validation.csv, and Test.csv: Each file maps video_id to their corresponding gesture labels.

In total, the dataset spans 27 gesture classes, with two additional categories - *"No gesture"* and *"Do other things"* - provided to help models differentiate between defined gestures and miscellaneous or unrecognized hand movements.

For the purpose of integrating gesture control with the YouTube application, we selected 18 specific gesture labels. These are summarized in the following table, along with their intended functions and corresponding keyboard shortcuts.

No.	Label	Functionality	Keyboard Shortcut
1	Doing other things	X	X
2	No gesture	X	X

3	Rolling Hand Backward	Seek to previous chapter	Ctrl + ←
4	Rolling Hand Forward	Seek to next chapter	Ctrl + →
5	Shaking Hand	Stop detect gesture	X
6	Sliding Two Fingers Down	Decrease Volume	↓
7	Sliding Two Fingers Left	Seek backward 5 secs	←
8	Sliding Two Fingers Right	Seek forward 5 secs	→
9	Sliding Two Fingers Up	Increase Volume	↑
10	Stop Sign	Toggle play/pause	k
11	Swiping Down	Toggle captions	c
12	Swiping Left	Switch to previous tab	Alt + ←
13	Swiping Right	Switch to next tab	Alt + →
14	Swiping Up	Next video	Shift + n
15	Thumb Down	Toggle mute	m
16	Thumb Up	Toggle full screen	f
17	Turning Hand Clockwise	Increase playback rate	Shift + .
18	Turning Hand Counterclockwise	Decrease playback rate	Shift + ,

Table 1: 18 gesture labels along with their functions and corresponding keyboard shortcuts

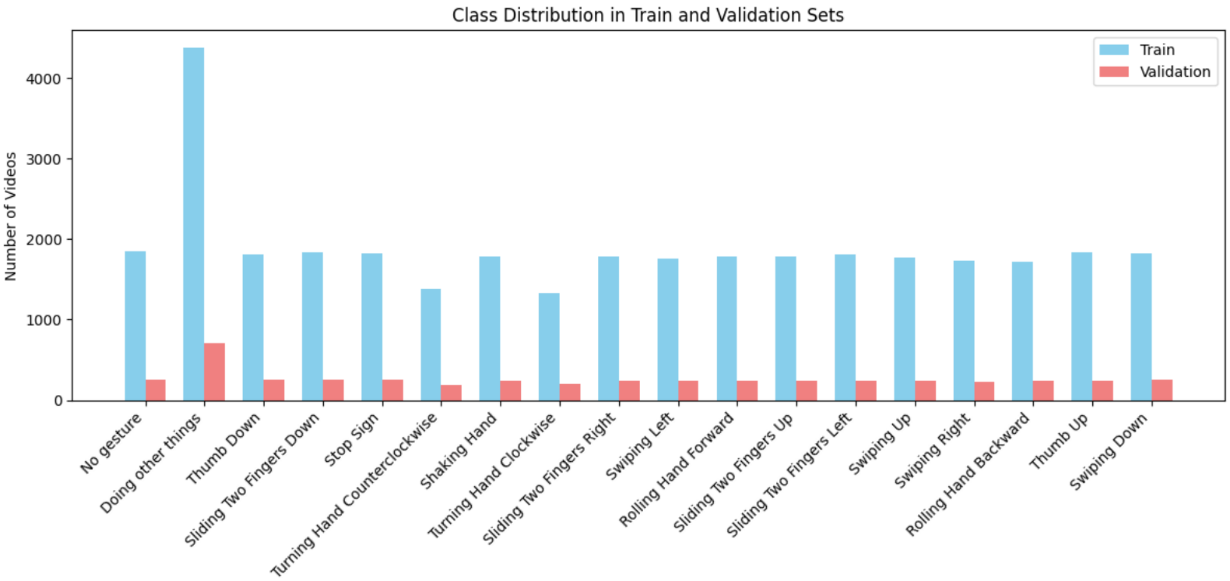


Figure 2: Class distribution of gesture class in Train and Validation sets

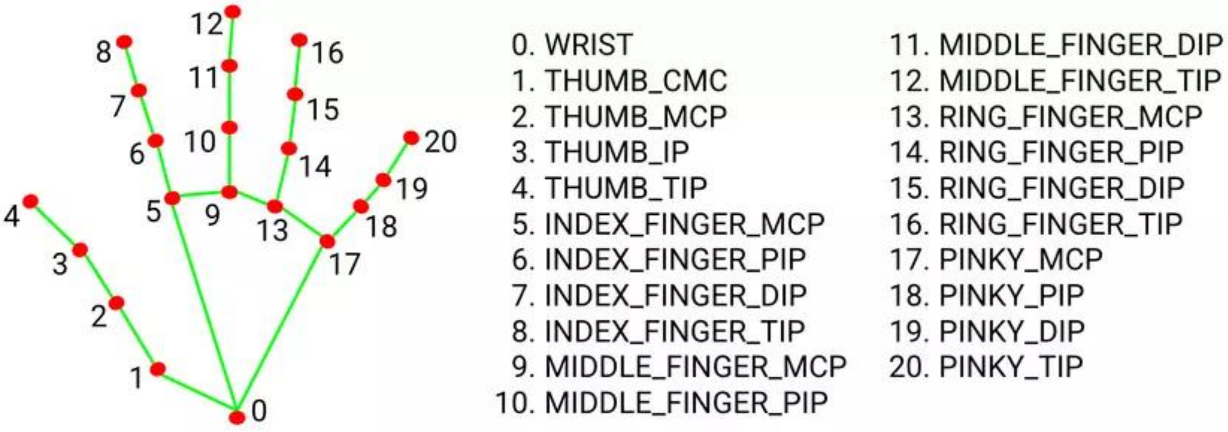
2.2. Data Preprocessing

First, sparse pose estimation is performed on the RGB frames. The extracted skeleton data is then used not only as input for the skeleton-based recognition but also to guide the cropping of relevant regions in the RGB frames, thereby enhancing spatial focus and improving recognition accuracy for RGB-based model.

2.2.1. Skeleton detecting

To extract human skeletal features, our team employed Google's Mediapipe framework, a versatile and lightweight machine learning solution developed by Google AI Edge. Mediapipe offers cross-platform compatibility and real-time inference, making it ideal for applications on mobile, desktop, web, and IoT devices. Notably, it is open-source and free to use, which facilitates easy integration and deployment.

In our implementation, we use Mediapipe's *Hand Detection* and *Human Pose Estimation* functionalities with RGB image inputs (assuming mirrored input from a front-facing camera). These functions return the (x, y, z) coordinates of body landmarks. For the Hand Detection module, we construct a skeletal model of the hand consisting of 21 key points per hand (Figure 3).



0. WRIST
1. THUMB_CMC
2. THUMB_MCP
3. THUMB_IP
4. THUMB_TIP
5. INDEX_FINGER_MCP
6. INDEX_FINGER_PIP
7. INDEX_FINGER_DIP
8. INDEX_FINGER_TIP
9. MIDDLE_FINGER_MCP
10. MIDDLE_FINGER_PIP
11. MIDDLE_FINGER_DIP
12. MIDDLE_FINGER_TIP
13. RING_FINGER_MCP
14. RING_FINGER_PIP
15. RING_FINGER_DIP
16. RING_FINGER_TIP
17. PINKY_MCP
18. PINKY_PIP
19. PINKY_DIP
20. PINKY_TIP

Figure 3: The hand skeleton consists of 21 key landmarks that represent important points on the hand.

Regarding pose landmarks, since our work focuses specifically on hand-related gestures, we extract only six upper-body landmarks: *Left/Right Shoulder, Left/Right Elbow,* and *Left/Right Wrist.*

Each video consists of 37 frames. For each video, we extract features from the relevant landmarks and store the data in a tensor of shape (T, V, C) = (37, 48, 3), where:

- T is the number of frames,
- V is the number of joints (including 6 body pose landmarks and 21 for each left and right hand) and
- C represents the 3D coordinates (x, y, z). The same process is applied to both the training and validation datasets.

2.1.2. Skeleton-guided RGB frame cropping

Due to the presence of numerous action-irrelevant background elements in RGB frames, directly extracting meaningful motion information becomes challenging. To address this, we employ a skeleton-guided cropping strategy.

In this approach, let $x \in X$, $y \in Y$ denote the horizontal and vertical coordinates of the detected skeleton joints. Based on these coordinates, we determine the active region of the skeleton in the RGB frame, defined by the intervals $[min(X),\ max(X)]$ and $[min(Y),\ max(Y)]$ as illustrated by the blue bounding box in the lower-left corner of Figure 4. These define the width and height of the region where most of the motion occurs:

$$w^j = max(X) - min(X) \text{ and } h^j = max(Y) - min(Y)$$

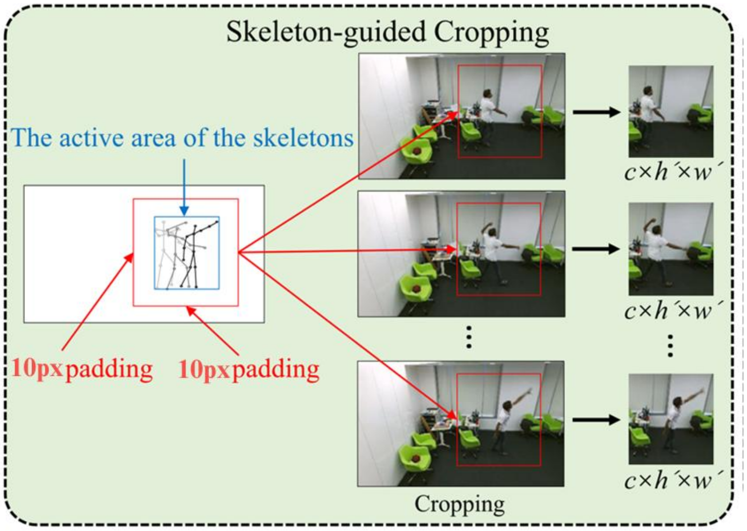


Figure 4. Cropping image

To ensure that the cropped region captures the full human body and allows for data augmentation, padding is applied on both the shorter side and the longer side of the bounding box, each extended by 10 pixels. Furthermore, to prevent overflow beyond the image boundaries, the resulting box is clipped to remain within the original frame dimensions. The final cropping intervals for both horizontal and vertical axes can be expressed as:

$$[max(min(X) - 10,\ 1),\ min(max(X) + 10,\ w)]$$

and

$$[max(min(Y) - 10,\ 1),\ min(max(Y) + 10,\ h)]$$

Where the w and h are the width and height of the original RGB frame. The red bounding box in the bottom left of Figure 4 indicates this final cropping region. This region is then consistently applied to all sampled RGB frames, ensuring uniformity in background context.

This skeleton-based cropping technique effectively focuses the model's attention on the regions of actual movement, thereby enhancing the performance of the RGB-based model in action recognition tasks.

2.3. Evaluation Criteria

To assess the performance of our action recognition models, we utilize four key evaluation metrics: Accuracy, Precision, Recall, and AUC (Area Under the Curve). Each metric provides a different lens through which to measure model effectiveness.

a. Accuracy

Accuracy measures the overall correctness of the model. It is calculated as the ratio of correctly predicted actions to the total number of predictions.

$$\text{Accuracy} = \frac{\text{TP}+\text{TN}}{\text{TP}+\text{TN}+\text{FP}+\text{FN}}$$

This metric gives a general view of performance but may be misleading when classes are imbalanced.

b. Precision

Precision focuses on the quality of positive predictions. It is the proportion of true positive predictions among all positive predictions.

$$\text{Precision} = \frac{\text{TP}}{\text{TP+FP}}$$

High precision means fewer false positives, which is critical in applications where incorrect positive results are costly.

c. Recall

Recall measures the model’s ability to detect all relevant instances. It is the ratio of true positives to all actual positives.

$$\text{Recall} = \frac{\text{TP}}{\text{TP+FN}}$$

In real-world applications, especially in gesture-based control systems, missing a user's intended action (false negative) can significantly degrade the user experience.

d. F1

F1 Score is the harmonic mean of Precision and Recall. It balances the trade-off between false positives and false negatives.

$$\text{F}_1 = \frac{\text{2.Precision.Recall}}{\text{Precision+Recall}}$$

In gesture recognition or action detection systems, F1 Score provides a more balanced measure of performance, ensuring that both the detection of correct actions and the minimization of errors are considered.

e. AUC (Area Under the ROC Curve)

AUC provides a comprehensive view of model performance across different thresholds. It represents the probability that the model ranks a randomly chosen positive instance higher than a randomly chosen negative one.

- An AUC of 1.0 indicates perfect separation.
- An AUC of 0.5 implies no discriminative ability.

3. Methodology

3.1. DSCNet

In this part, we investigate DSCNet, an action recognition framework that integrates both RGB and skeleton-based modalities. The objective is to harness the complementary strengths of multimodal data while maintaining low computational overhead. The overall architecture of DSCNet is illustrated in Figure 5.

Given a video segmented into t = 37 frames, where each frame consists of channel (c), height (h), width (w) dimensions, we first apply the preprocessing step that we have mention in part 2.2.1. The resulting data is structured in the format (batch size, T, V, C). This preprocessed skeleton data is then fed into MSSTNet for skeleton-based action recognition. Simultaneously, it is utilized to guide the cropping of relevant regions from the RGB frames.

The cropped RGB frames are divided into k = 6 segments, each containing 6 frames (excluding the final frame). These segments serve as input to the STMEM, which generates k motion maps. The resulting motion maps are then fed into an RGB-based action recognition backbone network. The final prediction is derived by fusing the classification outputs from both the skeleton and RGB streams, enabling a robust and comprehensive understanding of the performed actions.

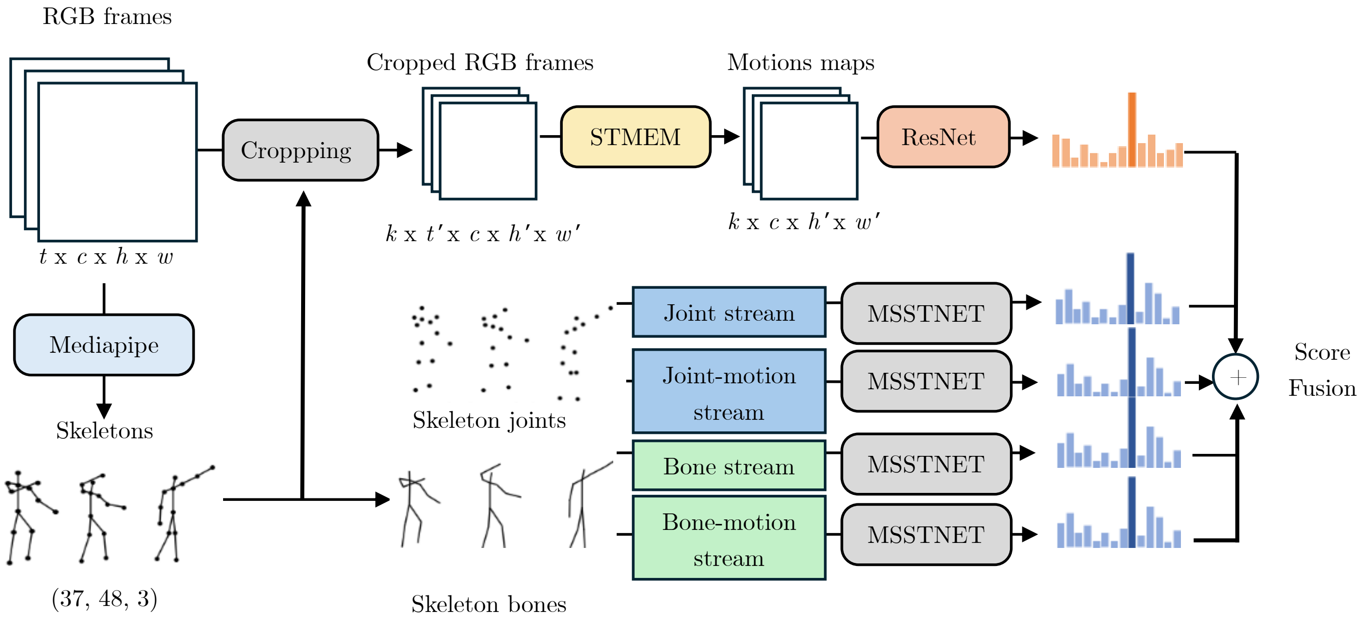


Figure 5: The framework of DSCNet

We consider the F1-scores for each of the 18 classes, including one F1-score from an RGB-based model and four F1-scores from skeleton-based models. During prediction, let the RGB-based model output a softmax probability vector

$(p_1, p_2, \dots p_{18})$, and each of the four skeleton-based models output softmax probability vectors $\left(q_1^i, q_2^i, \dots, q_{18}^i\right)$ where i ∈ {1, 2, 3, 4}.

The ensemble probability for class j is computed as:

$$r_j = \frac{p_j \cdot f_{1j-RGB} + \sum_{i=1}^{4} q_j^i f_{1j-skeleton}^i}{\sum_{i=1}^{4} f_{1j-skeleton}^i + f_{1j-RGB}}$$

3.2. Resnet

Residual Networks (ResNet) are a family of deep convolutional neural networks introduced by He et al. in 2015 to address the vanishing gradient problem that arises in very deep architectures. The key innovation in ResNet is the use of *residual connections*, which allow the network to learn residual functions with reference to the layer inputs, rather than trying to learn unreferenced functions directly. This design enables the training of extremely deep networks (e.g., ResNet-50, ResNet-101) without performance degradation, making ResNet highly effective for tasks such as image classification, object detection, and feature extraction.

3.2.1. 3D Resnet

Base on the idea of using ResNet models for 2D array problems, we investigate on wheather or not we can use or build an similary model for 3D array problems (video). Finally, we exploit the model which is already built from [8].

In detail, in our project, our 3D Resnet-50 has ability to handle the sequence of frames (pictures) in the time series, so we config the data input of the model with the size [32, 128, 128, 3]; corresponding to depth (number of frames), height, width and channels respectively. This architecture extends the widely used 2D ResNet-50 by replacing all the convolutional and pooling layer with the 3D counterparts. The network begins with an initial 3D convolutional layer using a *7×7×7 kernel* with *64 filters* and *stride 2*, followed by a 3D max pooling layer with a *3×3×3 kernel* and *stride 2*. The core of the model comprises four main stages: *Conv2_x*, *Conv3*_x, *Conv4_x*, and *Conv5_x*, each consisting of multiple bottleneck residual blocks, specifically *[3, 4, 6, 3]* subsequently . Each bottleneck block contains three layers: a *1×1×1* convolution for dimensionality reduction, a *3×3×3* convolution for spatial-temporal feature extraction, and another *1×1×1* convolution to restore dimensionality. Shortcut connections are included to facilitate gradient flow, with projection shortcuts applied when dimensions change.

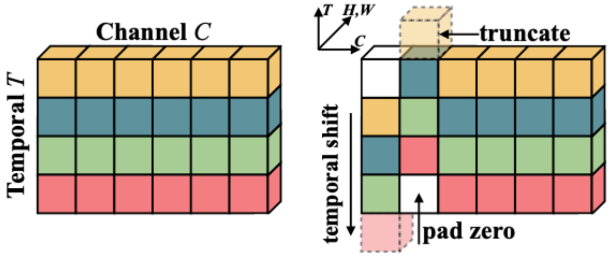
The number of filters increases across stages *(256 in Conv2_x, 512 in Conv3_x, 1024 in Conv4_x and 2048 in Conv)*, allowing the model to learn increasingly abstract and complex features. This hierarchical structure makes the 3D ResNet-50 well-suited for tasks involving high-dimensional data specially for our video classification task

Table 2: ResNet-50 structure

Layer name	3D ResNet-50
Conv1	7x7x7, 64, stride 2
	3x3x3 maxpool, stride 2
Conv2_x	$\begin{bmatrix} 1\,x\,1\,x\,1, & 64 \\ 3\,x\,3\,x\,3, & 64 \\ 1\,x\,1\,x\,1, & 256 \end{bmatrix}$ x 3
Conv3_x	$\begin{bmatrix} 1\,x\,1\,x\,1, & 128 \\ 3\,x\,3\,x\,3, & 128 \\ 1\,x\,1\,x\,1, & 512 \end{bmatrix} x\ 4$
Conv4_x	$\begin{bmatrix} 1\,x\,1\,x\,1, & 256 \\ 3\,x\,3\,x\,3, & 256 \\ 1\,x\,1\,x\,1, & 1024 \end{bmatrix} x\ 6$
Conv5_x	$\begin{bmatrix} 1\,x\,1\,x\,1, & 512 \\ 3\,x\,3\,x\,3, & 512 \\ 1\,x\,1\,x\,1, & 2048 \end{bmatrix} x\ 3$
	Global Average Pooling 3D
	Dense (Fully Connected)

3.2.2. TSM (Temporal Shift Module) with Resnet backbone:

To efficiently model temporal relationships without significantly increasing computational cost, we also experiment with another advance technique called the Temporal Shift Module (TSM), which will be applied into a standard 2D ResNet backbone [5]. TSM enables temporal modelling by shifting part of the feature maps along the temporal dimension during the forward pass. Specifically, for a video input split into consecutive frames, TSM shifts a fraction of the channels forward in time, another fraction backward, and leaves the rest unchanged. This simple but effective operation allows temporal information exchange across frames without introducing additional parameters or computational overhead.



(a) The original tensor without shift.

(b) Offline temporal shift (bi-direction).

Figure 6: Temporal Shift Module (TSM)

By inserting the TSM operation into residual blocks of a 2D ResNet, the model gains the ability to model motion patterns across adjacent frames while maintaining the efficiency and optimization benefits of pre-trained 2D convolutional networks. Unlike 3D CNNs, which are computationally expensive, TSM offers a lightweight alternative that balances temporal modelling capability with real-time inference needs. In our implementation, TSM is applied at multiple layers of the ResNet to ensure both low-level and high-level temporal features are captured, enhancing the model's sensitivity to subtle hand motion changes over time.

As stated in the TSM paper [5], TSM model have better performance when apply non-local module, which enhance the model’s ability to capture long-range temporal and spatial dependencies. Unlike standard convolutional or recurrent operations that operate on local neighborhoods, the non-local operation computes the response at a given position as a weighted sum of the features at all positions in the input.

$$y_i = \frac{1}{C(x)} \sum_j f(x_i, x_j) \,.\, g(x_j)$$

Where:

- i: index of an output position (in space, time, or spacetime)
- j: index that enumerates all possible positions.
- x: input signal (image, sequence, video; often their features)
- y: output signal of the same size as x.
- $f(x_i, x_j)$: affinity score between location i and j.

- $g(x_j)$: transformed representation the input features at location j.

So, the output at i aggregates all j, weighted by how related they are to i. This allows the model to directly relate distant spatial or temporal features, which is particularly beneficial for hand action recognition tasks where crucial motion cues may occur far apart in time or across different regions of the frame.

In order to incorporate this non-local operation into any existing model, Wang et al. (2017b) also introduce a way to wrap the operation into a non-local block [6]. They define the block as:

$$z_i = W_z y_i + x_i$$

Where y_i is given in the previous equation and "$+x_i$" denotes a residual connection. The whole block can be represented like in Fig 7 where: "⊗" denotes matrix multiplication, and "⊕" denotes element-wise sum; The blue boxes denote 1×1×1 convolutions.

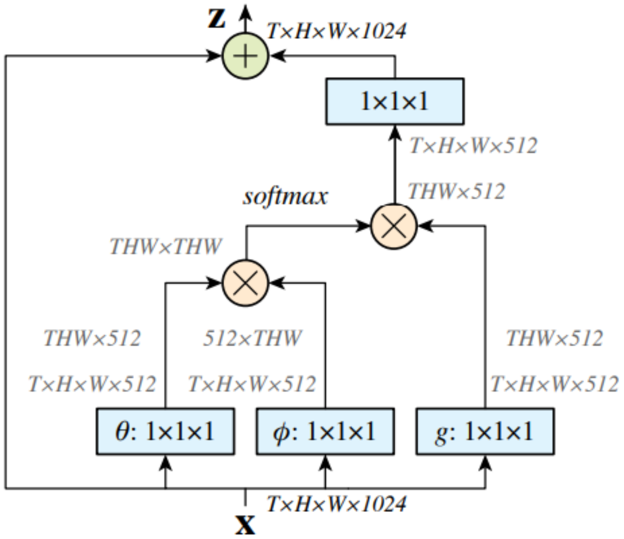


Figure 7: An example spacetime non-local block

The module is inserted after selected ResNet blocks, where it enriches feature representations by enabling global context modeling. This mechanism complements the local modeling and the short-range shifts of TSM, ultimately helping the network better understand complex and subtle hand gestures.

3.2.3. STMEM (Short-Term Motion Extract Module)

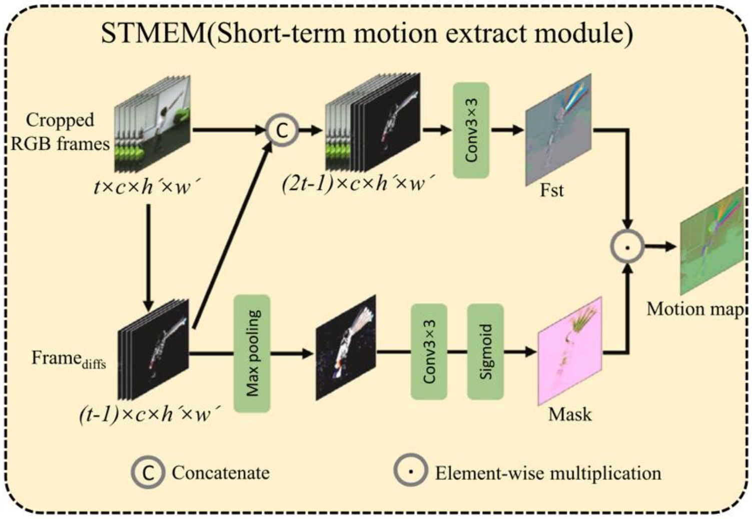


Figure 8. STMEM Architecture

Apart from directly feeding densely sampled RGB frames into the backbone network which incurs high computational costs, we experiment with the Short-Term Motion Extract Module (STMEM) to extract motion information segmentally. This approach compresses dense RGB frame sequences into a sparse representation, substantially reducing the data volume fed into the backbone. Within this module, convolutional neural networks are employed to capture motion cues over short temporal intervals. Notably, STMEM can be seamlessly integrated with backbone networks to form an end-to-end trainable action recognition framework, allowing its parameters to be optimized jointly for improved adaptability across diverse actions.

For t cropped frames in a video clip, we first calculate the frame differences:

$$Frame_{diffs} = \{frame_2 - frame_1,\ frame_3 - frame_2, \ldots,\ frame_t - frame_{t-1}\}.$$

A convolution layer is then used to extract the spatial-temporal features (Fst) of RGB frames and $Frame_{diffs}$.

$$Fst = Conv3(Concat(RGB\ frames,\ Frame_{diffs}))$$

where Conv3 denotes the convolution layer with the kernel size of 3 × 3.

Concatenating RGB frames retains sufficient spatial features compared to using $Frame_{diffs}$. alone. Then, to enhance crucial motion information, we use $Frame_{diffs}$ to generate a time attention mask through a maxpooling layer (temporal pooling), a convolution layer, and a sigmoid function. The motion map is obtained by multiplying the mask and Fst in element levels. The entire process can be described as:

$$\text{Motion map} = \sigma\big(Conv3(Concat(RGB\ frames,\ Frame_{diffs}))\big) \odot Fst$$

where maxpool denotes the maxpooling operation on the time dimension, σ denotes the sigmoid function, $\odot$ represents element-wise multiplication.

3.3. MSSTNet (Multi-scale spatial-temporal convolutional neural network)

We adopt MSSTNet as the baseline network for the skeleton modality. MSSTNet comprises seven MSST modules and two pooling modules, utilizing one-dimensional convolutions at multiple scales to effectively model the temporal and spatial dynamics of skeletal data. Originally designed for dense skeleton sequences, MSSTNet operates on inputs with a temporal length of 200 frames. However, in our implementation, we use sparser skeleton data with a temporal length of 37 frames.

Within MSST modules, stride settings are applied using convolution kernels of sizes 3 × 3, 5 × 1, 7 × 1, and 11 × 1, alongside an initial 1 × 1 convolution to manage the temporal resolution. All other convolutional layers maintain a stride of 1. The spatial resolution is primarily adjusted through the pooling modules and the stride of temporal convolutions, although for all MSST modules except the fifth, the stride is reduced to 1 to preserve temporal fidelity.

The skeleton data can be interpreted as a three-dimensional tensor of size H × W × C, where the height (H) is analogous to the temporal dimension (i.e., the number of frames, T), the width (W) represents spatial elements such as joints or bones (V), and the three channels (C) correspond to the spatial coordinates (x, y, z), similar to the RGB channels in an image (Figure 9).

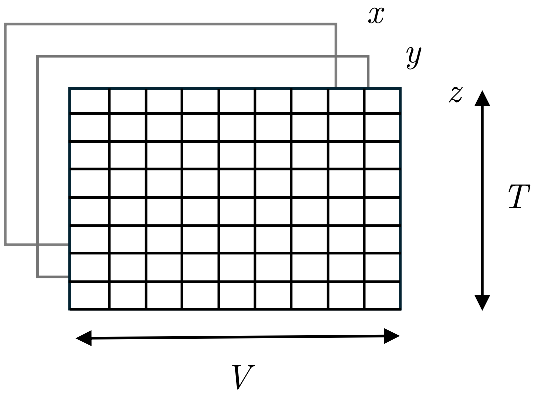


Figure 9: Illustration of skeleton

The architectural configuration of MSST modules can be summarized as follows:

- Input is fed into 5 parallel branches.
- Each branch begins with a 1×1 convolution to reduce dimensions.
- Branches have different convolutional patterns:
  - Branch 1: Simple 1×1 convolution (preserves spatial information).
  - Branch 2: 1×1 followed by separate 3×1 and 1×3 convolutions (small receptive field).
  - Branch 3: 1×1 followed by separate 5×1 and 1×5 convolutions (medium receptive field).
  - Branch 4: 1×1 followed by separate 7×1 and 1×7 convolutions (larger receptive field).
  - Branch 5: 1×1 followed by separate 11×1 and 1×11 convolutions (largest receptive field).
- All branches are concatenated at the end to produce output feature maps.
- Each convolution is typically followed by batch normalization and ReLU activation.

The output of each module is the input of the next module. The detailed layout of MSSTNET is shown in Table 3 and Figure 10. MSSTNET is capable of learning motion patterns over short, medium, and long temporal intervals, and can distinguish between similar actions that differ in their motion details.

Table 3: Architecture of the MSSTNet

Module	Stride	1 x 1	3 x 3	5 x 1	7 x 1	11 x 1	Input size
				1 x 5	1 x 7	1 x 11	
MSST module 1	(1, 1)	44	60	60	60	60	(T, V, 3)
MSST module 2	(1, 1)	48	80	80	80	80	(T, V, 284)
MSST module 3	(1, 1)	56	120	120	120	120	(T, V, 288)
Avg pool	(1, 2)						(T, V, 536)
MSST module 4	(1, 1)	160	160	160	160	160	(T, V/2, 536)
MSST module 5	(2, 1)	72	200	200	200	200	(T, V/2, 800)
Avg pool	(1, 2)						(T/2, V/2, 872)
MSST module 6	(1, 1)	240	240	240	240	240	(T/2, V/4, 872)
MSST module 7	(1, 1)	320	320	320	320	320	(T/2, V/4, 1200)
Global Avg pool	(1, 1)						(T/2, V/4, 1600)
Full Connection							(1, 1, 1600)

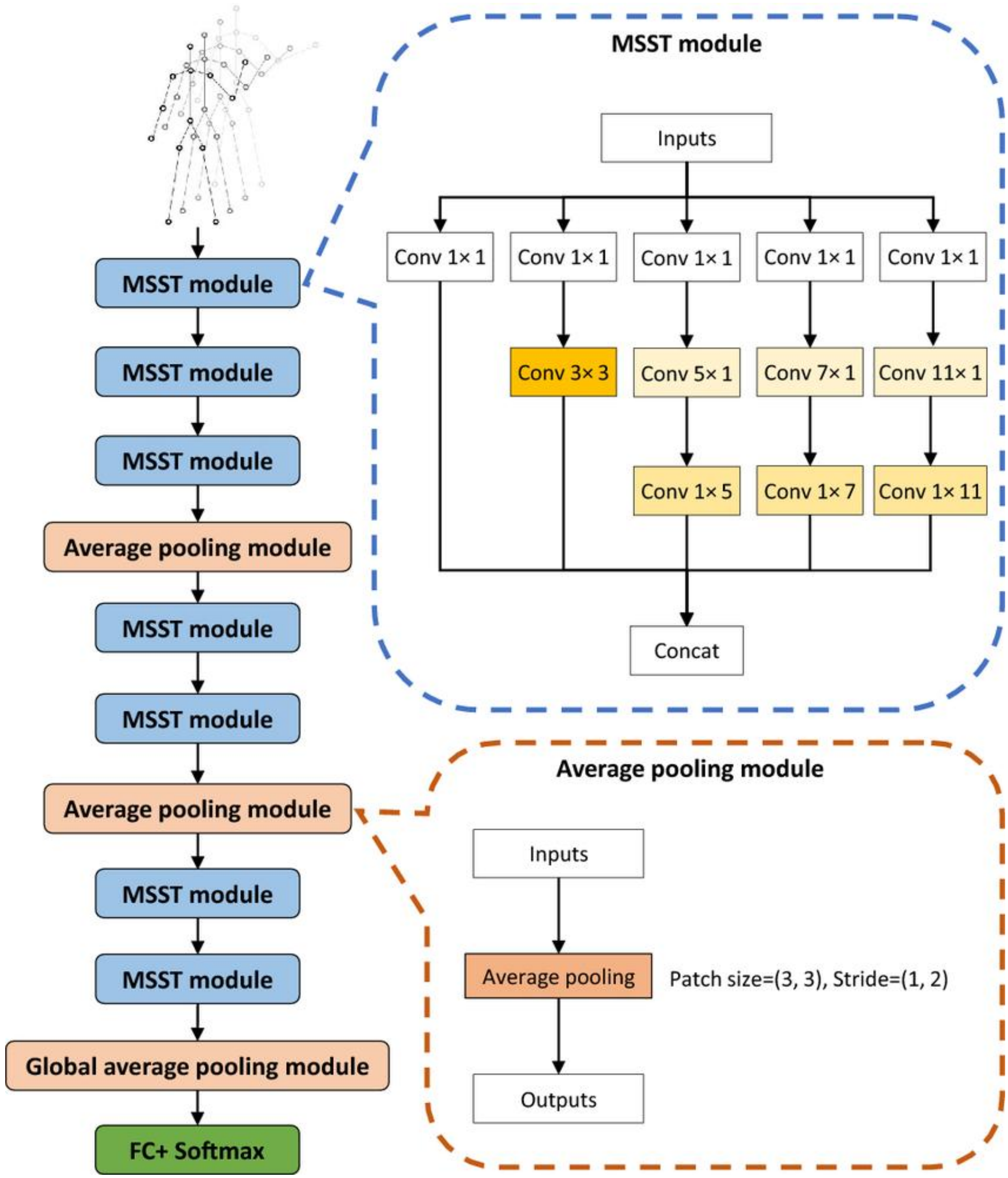


Figure 10: MSST module consists of five convolutional branches. The entire network is composed of multiple MSST modules and average pooling modules. Pooling reduces the spatial size of the feature map and suppresses overfitting

3.4. Implementation & Training

3.4.1. Resnet

In this section, we describe the implementation, training, and experimentation process for two RGB-based models: a 3D ResNet and a STMEM + TSM-ResNet. The goal is to compare the performance of a standard ResNet model with an enhanced ResNet architecture that incorporates more advanced modules.

3.4.1.1. Resnet3D

Preprocessing of the 37-frame video data was necessary to align with the 3D ResNet-50 model's 32-frame input depth, five frames ([5, 12, 19, 26, 33], indexed 0-36) were removed. These frames were chosen for their central locations and equal spacing, optimizing the distribution within the reduced dataset. We also have to resize the cropped frame into (128 x 128 x 3) corresponding to (H x W x C) to fit the model.

The output of each layer and residual block are presented in the table below:

Layer/Block	Output Shape (D × H × W × C)	Note
Input	[32, 128, 128, 3]	
Conv1	[16, 64, 64, 64]	7×7×7 conv, stride=2
MaxPool	[8, 32, 32, 64]	3×3×3 pool, stride=2
Conv2_x	[8, 32, 32, 256]	3 bottleneck blocks
Conv3_x	[4, 16, 16, 512]	4 bottleneck blocks, first block stride=2
Conv4_x	[2, 8, 8, 1024]	6 bottleneck blocks, first block stride=2
Conv5_x	[1, 4, 4, 2048]	3 bottleneck blocks, first block stride=2
Global AvgPool	[1, 1, 1, 2048]	Avarage Maxpooling
Flatten	[2048]	

For the trainning session, our model was trained on train_data from 20bn_jester dataset. Using the *“categorical_ crossentropy”* a specific kind of widely used cross_entropy loss function for multiple classes classification.

$$L = -\sum_{i=1}^{C} y_i log(\widehat{y_i})$$

- L: Loss value
- C: number of class
- y_i: the ground truth value
- $\widehat{y_i}$: the value predicted from model

We use the optimizer ADAM from tensorflow.keras library with default arguments such as learning rate=0.001. We train the model through 48 epoches with batch size of 4 videos.

3.4.1.2. STMEM + TSM-ResNet:

Originally, we experimented with STMEM + ResNet3D architecture. However, during training, we observed that the model began to overfit quite early even though we have already applied various regularization techniques. We concluded that inserting STMEM before a ResNet3D backbone makes the overall architecture excessively complex. Since STMEM already captures temporal dependencies to a certain extent, combining it with a 3D ResNet is simply overkill. Therefore, we reverted to the original idea from DSCNet [4], which uses a Temporal Shift Module (TSM) with a ResNet backbone to process the output from STMEM.

Before training, we removed the last frame from each video to retain 36 frames, making it easier to divide the sequence into segments later. We then applied random brightness/contrast adjustments and color jittering to the cropped frames as data augmentation techniques. Each frame was resized to 128×128 pixels, as in the ResNet3D setup, and the frames were then stacked to form a video tensor.

This video tensor was divided into 6 segments, each containing 6 frames. These segments were passed through STMEM, where each segment was transformed and compressed into a motion map rich in both spatial and temporal information. As a result, we obtained 6 motion maps per video.

The motion maps were reshaped into a tensor of shape (B×S, 3, H, W), where B is the batch size, S is the number of segments, and 3 represents the number of channels in each motion map. This tensor, along with the segment count, was passed into the TSM module, which performed temporal shifting before feeding the data into a 2D ResNet model.

As discussed in the methodology section regarding the effect of non-local modules, we also integrated non-local blocks into our model. These blocks were inserted into the second and third layers of the ResNet architecture.

The model was initialized with ResNet-50 weights pre-trained on ImageNet, which were recalculated to adapt for RGB difference inputs. Training was performed using the Cross Entropy Loss function and the ADAM optimizer, with a learning rate of 1e-4 and a weight decay of 1e-5.

3.4.2. MSSTNET

Initially, a joint-stream dataset was extracted from video inputs using MediaPipe. From this dataset, we derived multiple data streams to capture various aspects of human motion. The joint-motion stream was computed by taking the difference in joint coordinates between consecutive frames, effectively capturing the temporal dynamics of joint movement. The bone stream was constructed to represent the skeletal structure, where bones are defined as the connections between adjacent joints. Subsequently, the bone-motion stream was obtained by calculating the temporal differences in bone vectors across successive frames, thereby modeling the dynamic behavior of the skeletal structure.

As a result, four distinct data representations of human skeletal motion were generated. Each of these streams was independently input into a separate MSSTNet model, enabling specialized feature extraction tailored to each motion representation. The outputs from the four MSSTNet models were then ensemble together to enhance the overall prediction accuracy through multi-stream integration.

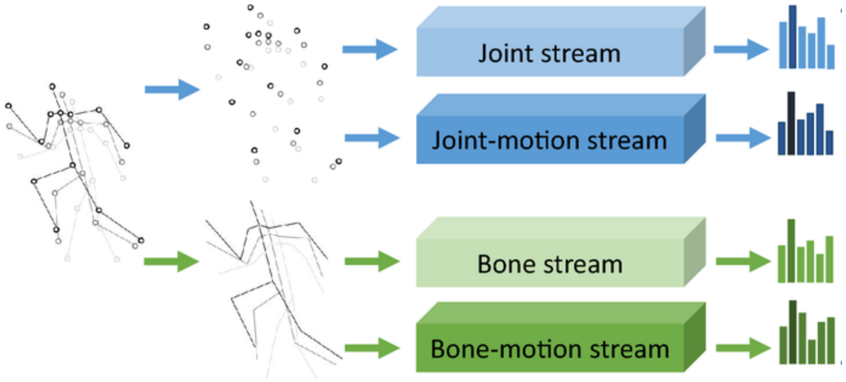


Figure 11: Input for MSSTNet model

The model was trained using the Adam optimizer, initialized with a learning rate of 0.01. Adam was selected for its adaptive learning rate mechanism, which offers improved convergence compared to conventional stochastic gradient descent methods. The categorical cross-entropy loss function was employed, reflecting the multi-class classification nature of the task, while categorical accuracy was used as the primary evaluation metric.

Training was conducted over a maximum of 30 epochs with a batch size of 32, balancing computational efficiency and the stability of gradient updates. The final model is chosen based on best val_categorical_accuracy. To enhance generalization and mitigate overfitting, several advanced training strategies were applied. An early stopping mechanism with a patience value of seven epochs was utilized, halting the training process once validation accuracy ceased to improve and automatically restoring the best-performing model's weights. Furthermore, the learning rate was adaptively adjusted using a plateau-based scheduling technique. If no improvement in validation accuracy was observed for four consecutive epochs, the learning rate was reduced by a factor of 0.5, with a minimum threshold set at 1e-5. This strategy enabled the model to escape local minima and promoted more refined convergence during the later stages of training.

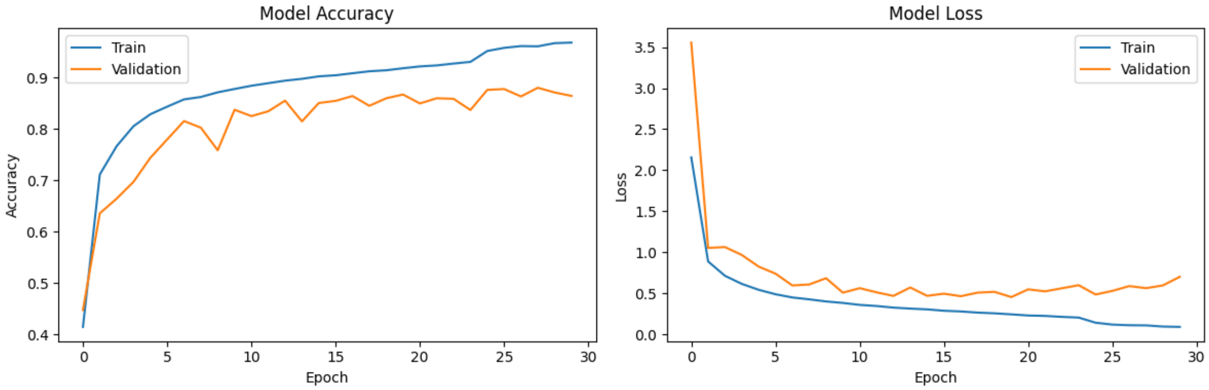


Figure 12: Training and Validation Accuracy/Loss on the Joint Stream Data

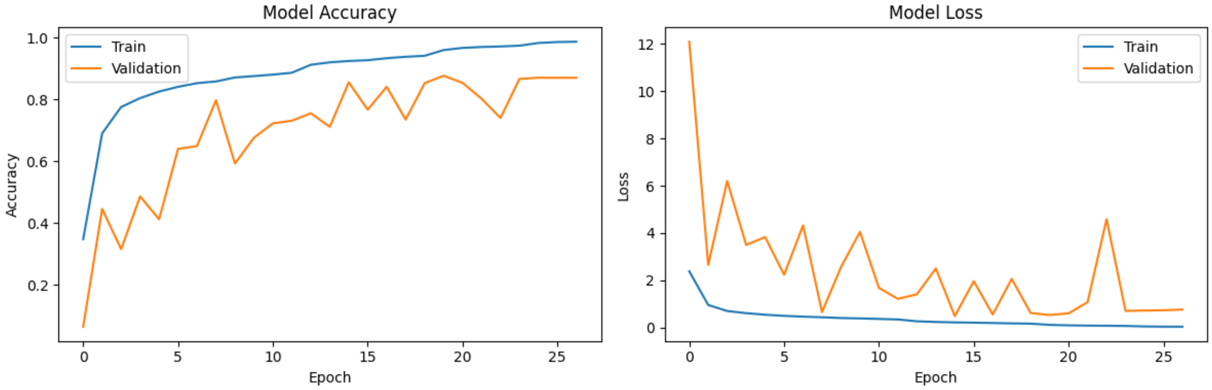


Figure 13: Training and Validation Accuracy/Loss on the Joint-motion Stream Data

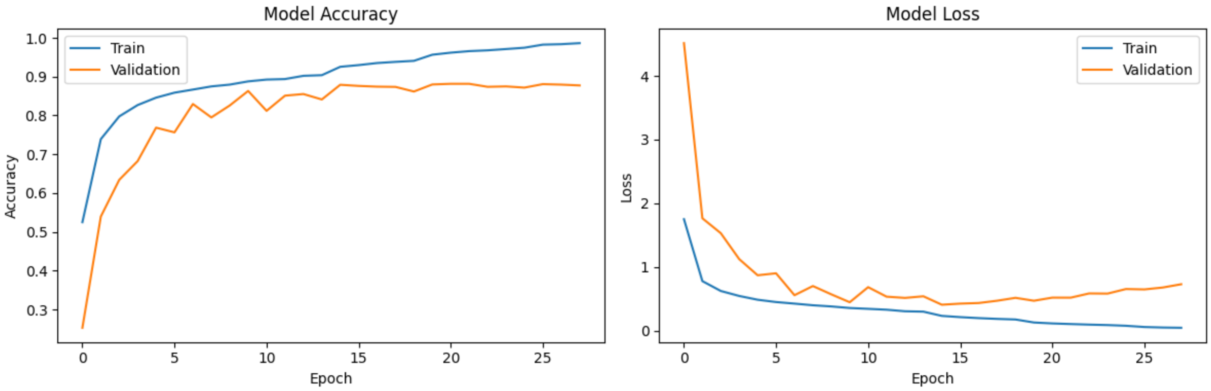


Figure 14: Training and Validation Accuracy/Loss on the Bone Stream Data

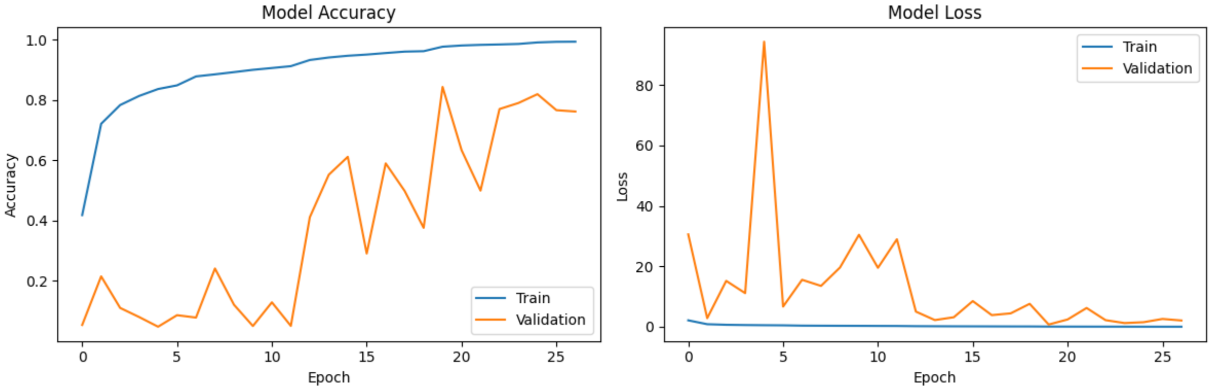


Figure 15: Training and Validation Accuracy/Loss on the Bone-motion Stream Data

Table 4: Performance of each model on validation dataset

Model	Accuracy	Precision	Recall	F1 Score	AUC
Joint stream	0.8906	0.8920	0.8906	0.8904	0.9952
Joint motion stream	0.8877	0.8910	0.8877	0.8879	0.9952
Bone stream	0.8906	0.8913	0.8906	0.8904	0.9952
Bone motion stream	0.8611	0.8633	0.8611	0.8606	0.9947
Ensemble	0.9042	0.9059	0.9042	0.9041	0.9954

4. Result

This section outlines the experimental methodology, including the dataset utilized, the computational environment, and the evaluation metrics. It then presents a comparative analysis of the performance of the evaluated hand gesture recognition models, followed by model-specific observations regarding their training processes and outcomes.

4.1. Experimental Setup

Training Environment:

The training for all models was conducted on the Kaggle platform, utilizing a consistent environment equipped with 2x NVIDIA T4 GPUs. This standardized setup ensures fair comparison of training efficiency and model performance across the different architectures.

4.2. Comparative Performance

The results of the evaluation are as follows:

Table 5: Performance of all models on validation dataset

Model	Accuracy	Precision	Recall	F1 Score	AUC
STMEM + TSM-Resnet	0.9098	0.9132	0.9098	0.9095	0.9966
DSCNet Ensemble	0.9095	0.9108	0.9095	0.9094	0.9939
MSSTNET (Ensemble)	0.9042	0.9059	0.9042	0.9041	0.9954
3D ResNet-50	0.7713	0.8003	0.7398	0.7687	0.9723

4.3. Model-Speci⁄c Observations

3D ResNet-50: This model processes video data directly using 3D convolutions to learn spatio-temporal features. On the Jester validation set, it achieved an accuracy of 0.7713 and an AUC of 0.9723.

MSSTNET (Ensemble): This approach also uses skeleton data and utilizes four MSSTNet models (for joint, joint-motion, bone, and bone-motion streams). Each of these models required 4 hours and 37 minutes for training. The ensemble of these streams has an accuracy of 0.9042 and an AUC of 0.9954.

STMEM + TSM-ResNet: This RGB-based model integrates a Short-Term Motion Extract Module (STMEM) with a Temporal Shift Module (TSM) applied to a ResNet backbone. The training was about 200 minutes. It demonstrated strong performance, achieving the highest accuracy of 0.9098 and an AUC of 0.9966 among the models.

DSCNet Ensemble: This dual-stream architecture leverages both RGB (via STMEM + TSM-ResNet) and skeleton (via MSSTNet) modalities. By fusing features from both streams, it achieved a high accuracy of 0.9095 and an AUC of 0.9939. Its training time is shown by its ensembled models.

5. Application

5.1. Image enhancement

To improve the visual quality of input images and enhance the performance of subsequent computer vision tasks, particularly keypoint extraction, we implement two image enhancement techniques: Contrast Limited Adaptive Histogram Equalization (CLAHE) and Gamma Correction. These methods are specifically chosen to produce images with balanced histograms and optimal brightness, which are crucial for accurately identifying and extracting keypoints.

Contrast Limited Adaptive Histogram Equalization (CLAHE) is a refined variant of the traditional histogram equalization method. Unlike standard histogram equalization, which enhances contrast across the entire image, CLAHE works on small regions known as tiles (in our case, 8×8 tiles). Each tile is individually processed to enhance local contrast, and the neighboring tiles are then combined using bilinear interpolation to eliminate artificially induced boundaries.

Since the input images are in color, it is necessary to convert them into an appropriate color space before applying CLAHE. We use the LAB color space, where 'L' denotes lightness and 'A' and 'B' represent color-opponent dimensions. The CLAHE technique is applied specifically to the L-channel (Lightness), thereby enhancing the luminance component of the image while preserving the color integrity. After the contrast enhancement is performed on the L-channel, the image is recombined and converted back to the original color space to yield the final enhanced image.

Gamma Correction is another widely used technique to adjust the overall brightness of an image. It involves applying a nonlinear transformation to each pixel value using the following formula:

$$f(I) = \left(\frac{I}{255}\right)^{1/\gamma} \times 255$$

In this equation, I represents the original intensity value of a pixel (ranging from 0 to 255), and γ is the gamma value that controls the transformation. If $\gamma < 1$, the image becomes darker, while $\gamma > 1$ results in a brighter image. The γ is calculated by the formula:

$$\gamma = \frac{log(255 \times 0.5)}{log\, mean}$$

Where the mean refers to the average brightness, which is calculated by converting the image to the HSV color space and then computing the mean of the V (value) channel across all pixels.

This transformation is particularly useful for correcting images that appear either too dark or too bright due to improper exposure or lighting conditions.

Together, CLAHE and gamma correction work to produce enhanced images with balanced contrast and appropriate brightness, which are critical conditions for effective keypoint extraction and reliable feature matching in subsequent stages of image analysis or computer vision tasks.

5.2. Keyboard control

The program incorporates keyboard control functionality to allow for intuitive and efficient user interactions. This feature is implemented using the *keyboard* module, which enables the detection and handling of various keyboard events. Through this module, users can trigger specific functions or navigate the interface seamlessly by pressing predefined keys.

5.3. UI Integration

The graphical user interface (GUI) of the program is developed using Dear PyGui, a fast and user-friendly Python GUI framework. Dear PyGui enables the creation of modern and interactive user interfaces with minimal overhead. It provides a range of built-in widgets and tools that facilitate the integration of image processing functions, control buttons, sliders, and display panels. The UI design ensures that users can easily interact with the application, adjust parameters such as gamma values or enhancement settings, and view the processed images in real-time.

6. Discussion and Conclusion

6.1. Discussion

6.1.1. Comparative Analysis

The evaluation shows different performance levels and efficiency among the models. The STMEM + TSM-ResNet rised as the top-performing model (Accuracy: 0.9098, AUC: 0.9966), outperforming the standard 3D ResNet-50 (Accuracy: 0.7713). This shows the effectiveness of STMEM in creating motion-focused representations and TSM's efficiency in capturing temporal dynamics within a 2D CNN framework, while also being more efficient in training (~200 minutes).

For skeleton-based data, MSSTNET's multi-stream, multi-scale architecture, with a significant training time of 4.6 hours per model effectively captured complex skeletal relationships.

The multi-modal DSCNet Ensemble (Accuracy: 0.9095) ssed both STMEM + TSM-ResNet (RGB) and MSSTNET (skeleton) streams. While the accuracy is high, the performance still falls short over the best single-modality model (STMEM + TSM-ResNet). This suggests that for this specific dataset and model combination, the RGB stream already captured a good portion of the distinctive information. The score-level fusion utilized might also be a factor. Better fusion techniques could potentially unlock greater benefits. However, the combined training time from its models makes the training efficiency worse than the leading single-stream RGB model.

Overall, the results indicate that specialized modules for motion extraction (STMEM) and efficient temporal modeling (TSM) in RGB models can be highly effective. Multi-modal fusion, in this case, did not elevate performance beyond the best individual stream, suggesting that the choice of fusion strategy and the relative strengths of the individual streams are critical considerations.

6.1.2. Comparison with State-of-the-Art (SOTA)

The top-performing models in this project, STMEM + TSM-ResNet, achieved accuracies around 90.9%. While this is a solid result, they fall short of the current SOTA on the Jester dataset. For example, DirecFormer (CVPR 2022), an RGB-based Transformer model, reported a Top-1 accuracy of 98.15%. This significant performance difference can be due to the Transformer architecture, whose attention mechanisms are really good at capturing long-range temporal dependencies and complex relationships within video data, often surpassing traditional CNN in such tasks.

Another notable SOTA model, Motion Fused Frames (MFFs) / DRX3D (CVPRW 2018), which is CNN but includes both RGB and clear motion information through Optical Flow, achieved 96.3% accuracy. This highlights the benefit of integrating dense motion cues like optical flow, which provides more detailed inter-frame change information than the motion maps generated by STMEM alone.

6.2. Conclusion

This report looked at how well different methods can recognize hand gestures using the Jester dataset. We tested three main types: methods using normal videos (RGB-based), methods using skeletal data, and methods combining both (multi-modal).

Our results show that for recognizing gestures from videos, adding special components to better understand movement and how it changes over time works well. For example, the STMEM + TSM-ResNet performed very well, gaining an accuracy of 0.9098, and it was quite efficient in training.

For skeleton-based models, the MSSTNET model, which combines several approaches, receeive a good accuracy (0.9042). However, it requires a lot of computing power, which last up to a day to finish.

When we tried to ensemble video and skeleton data models with the DSCNet model, it also performed well (0.9095 accuracy). But, there was no improvement over the best single method we tested. This suggests that how the data are combined and how good each individual method is really matters.

Even though our models performed not bad, they are not yet as good as the SOTA models, like DirecFormer (98.15% accuracy) and MFFs/DRX3D (around 96.3%). This difference shows that newer designs, such as Transformer models which are good at noticing important details in long sequences, and methods that use detailed motion information (like optical flow), have made big progress. Still, the models we studied, especially STMEM + TSM-ResNet give us useful ideas for designing effective systems using common techniques like CNNs and GCNs, if we need them to be efficient.

For future work, we need to find ways to make our models perform as well as the top SOTA models. We could try adding attention mechanisms into our current models, or even switch to using Transformer models. It would also be good to find better ways to include detailed motion information, perhaps something like optical flow but easier to compute. Additionally, we should explore more advanced ways to combine video and skeleton data, instead of just adding their scores together, to get more benefits.

Finally, a key goal is to use what we've learned to build a faster, more accurate, more useful practical computer control system, other than just controlling Youtube, that uses hand tracking in real-time. This will mean making our models even faster for quick responses, with better integrations.

7. References

[1] J. Materzynska, G. Berger, I. Bax and R. Memisevic, "The Jester Dataset: A Large-Scale Video Dataset of Human Gestures," 2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW), Seoul, Korea (South), 2019, pp. 2874-2882, doi: 10.1109/ICCVW.2019.00349.

[2] Zhang, F., Bazarevsky, V., Vakunov, A., Tkachenka, A., Sung, G., Chang, C., & Grundmann, M. (2020, June 18). *MediaPipe Hands: On-device real-time hand tracking*. arXiv.org. https://arxiv.org/abs/2006.10214

[3] Cheng, Q., Cheng, J., Ren, Z. *et al.* Multi-scale spatial–temporal convolutional neural network for skeleton-based action recognition. *Pattern Anal Applic* 26, 1303–1315 (2023). https://doi.org/10.1007/s10044-023-01156-w

[4] Q. Cheng, J. Cheng, Z. Liu, Z. Ren and J. Liu, "A Dense-Sparse Complementary Network for Human Action Recognition based on RGB and Skeleton Modalities," *Expert Systems with Applications*, vol. 222, 2023, Art. no. 123061, doi: 10.1016/j.eswa.2023.123061.

[5] Lin, J., Gan, C., & Han, S. (2018, November 20). *TSM: Temporal Shift Module for Efficient video Understanding*. arXiv.org. https://arxiv.org/abs/1811.08383

[6] Wang, X., Girshick, R., Gupta, A., & He, K. (2017b, November 21). *Non-local neural networks*. arXiv.org. https://arxiv.org/abs/1711.07971
[7] He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep Residual Learning for Image Recognition. arXiv.org. https://arxiv.org/abs/1512.03385
[8] Ju, J. (n.d.). keras-resnet3d. GitHub. https://github.com/JihongJu/keras-resnet3d.git
[9] K. Cheng, Y. Zhang, X. He, W. Chen, J. Cheng and H. Lu, "Skeleton-Based Action Recognition With Shift Graph Convolutional Network," 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Seattle, WA, USA, 2020, pp. 1106-1115,
https://openaccess.thecvf.com/content_CVPR_2020/papers/Cheng_Skeleton-Based_Action_Recognition_With_Shift_Graph_Convolutional_Network_CVPR_2020_paper.pdf
[10] Wu, C., Liu, Y., Zaheer, M., Dai, H., Avanesov, A., & Yu, F. (2022). DirecFormer: A Directed Attention in Transformer for 3D Human Pose Estimation. 2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), New Orleans, LA, USA, 2022, pp. 20929-20938, doi: 10.1109/CVPR52688.2022.02029.
[11] Li, Y., Felsen, P., & Yuan, J. (2018). DRX3D: Dynamic Resolution eXchange for 3D Human Pose Estimation in Videos. 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), Salt Lake City, UT, USA, 2018, pp. 1049-10499, doi: 10.1109/CVPRW.2018.00140.