# Efficient 3D Video Engine Using Frame Redundancy

Gao Peng Bo Pang Cewu Lu*
Shanghai Jiao Tong University
{penggao, pangbo, lucewu}@sjtu.edu.cn

## Abstract

*Traditional 3d video understanding methods process videos frame by frame. We argue that a lot of computation in this mechanism is redundant based on a key observation - adjacent frames in 3D videos have visually similar geometry structure. To handle the redundancy, we propose the **E**fficient 3D **V**ideo **E**ngine (EVE), aiming to avoid the computation of redundant points. It consists of two modules: 1) redundancy removing module designed to detect redundancy and remove it; 2) residual learning module to extract features on non-redundant points. As a simple plug and play framework, EVE can be easily incorporated in mainstream 3D models. Experiments demonstrate that EVE can significantly reduce computation without performance loss on large scale datasets. On the other hand, with similar computation, EVE outperforms the strong baseline by up to 4.1 mIoU on SemanticKITTI. The code is available on https://github.com/ecr23xx/eve.*

## 1. Introduction

3D video understanding is becoming an industry and academic popular task, and many 3D spatial-temporal datasets have been proposed [1, 19, 24, 44, 45]. Recently, deep convolutional neural networks (CNNs) advanced different tasks of 3D video understanding, such as 3D video segmentation and 3D action recognition [3, 13, 33, 45].

3D videos are mostly represented as consecutive point cloud frames and current state-of-the-art deep learning-based 3D video processing methods are all in frame-by-frame mechanism or using the stacked frames' spatial-temporal features [3, 32, 33]. However, as shown in Fig. 1, there are a lot of geometrically similar points between adjacent frames. We argue that repeatedly processing them will introduce huge redundant computation, making it computationally expensive. For example, to process one clip (5 frames) of 80K points frame by frame, a strong baseline MinkUNet [3] needs $5 \times 119.2$G multiply-add operations (FLOPs) operations, which is around **30**$\times$ larger than a typical image recognition model such as ResNet50 [16].

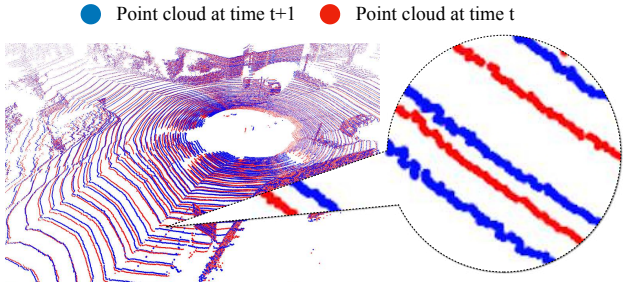


Figure 1: **Motivation**. Adjacent 3D point clouds are visually similar after registration. We argue that the computation of many overlapped or close points can be reused to accelerate 3D video tasks.

To reduce redundant computation, we propose to take advantage of the geometrically close points between adjacent frames in a video clip. Because the semantic information of 3D cloud is encoded in the geometrical structure of points, we naturally hypothesize that the geometrically close points must share similar high-level deep features. Thus finding an efficient method to reuse these features, instead of recalculating them, would be a potential efficient solver for 3D video tasks.

Instead of taking each frame as an independent input, we propose the Efficient 3D Video Engine (EVE) to utilize the correspondence between frames and avoid redundant computation. It contains a *redundancy removing* module to detect the correspondence between two adjacent frames and reuse features for geometrically close points. Remaining points are processed by a *residual learning* module to learn the residual information. Fig. 2 illustrates our ideas. We argue that this match-and-residual learning mechanism should be more efficient than frame-by-frame processing for point cloud videos.

We conduct comprehensive experiments on large-scale point cloud video datasets to show the universal existence of redundancy and evaluate the proposed EVE. Our exper-

*Cewu Lu is corresponding author, member of Qing Yuan Research Institute and MoE Key Lab of Artificial Intelligence, AI Institute, Shanghai Jiao Tong University, China and Shanghai Qi Zhi institute.

iments indicate that frame redundancy in point cloud video is huge, over 60% computation is repetitive using current mainstream methods. By matching and reusing features with previous frames, the redundant computation can be saved. For non-redundant points, it's better to model it as residual features to their nearest points in the previous frame rather than directly process it.

Results on large-scale 3D video datasets justified that the proposed EVE could maintain accuracy while avoiding redundant computation. As shown in Fig. 3, with the proposed EVE, we improve the current strong baseline by **4.1 mIoU** with similar computation on SemanticKITTI [1]. Result on Synthia4D [44] also justifies that EVE is more efficient than the current mainstream method.

Our contributions in this paper are three fold:

- We propose and analyze the frame redundancy problem in 3D video understanding;
- We propose the Efficient Video Engine (EVE) to process 3D videos more efficiently;
- The proposed EVE use similar computation while outperforming current strong baselines (relative improvements of 8%).

## 2. Related Work

**3D Deep Learning.** 3D deep learning has long been researched [5, 21, 23, 25, 26, 35, 40, 41, 49, 54, 61, 64]. Qi *et al*. proposes PointNet [40] which directly performs deep learning on point clouds with the help of symmetric functions. To aggregate neighborhood information, previous works, like PointNet++ [41], defines different convolution kernels on the k-nearest-neighbor or a spherical neighborhood, which greatly improves the capacity of PointNet. Recently, methods that do not process information in the point cloud domain emerged. MinkowskiNet [3] and Submanifold Sparse Convolutional Networks [11] uses sparse convolution to process point cloud, which generates input-output kernel mapping on the fly and skips points that are in ground state.

**Temporal Understanding.** With the availability of large amounts of video datasets [9, 12, 22, 27, 39, 46, 48], 3D convolution based methods have achieved remarkable success on video understanding and action recognition with 3D spatial-temporal features: [2, 7, 28–30, 47, 52, 55, 63, 66], more effective than recursive methods [4, 37, 38, 62]. Recently, several 3D point cloud video datasets have been proposed [1, 24]. Choy *et al*. [3] studies 3D video understanding via 4D Spatial-Temporal neural networks and conditional random fields. Liu *et al*. [33] proposed a meteor module to process 3D sequence information. However, those methods require huge computation for point cloud sequences. Compared with 3D methods, most computation of 4D methods comes from the stacked feature map. But we find there exists redundancy among 4D clips, thus we aim to reduce the redundant computation in it.

**Structural redundancy.** Efforts have been paid to reduce structural redundancy in deep learning. OctNet [42] utilized a more efficient data structure oct-trees to store sparse voxel, which aimed to reduce the geometry redundancy in 3D voxel representation. Graham et al. [11] introduced sub-manifold sparse convolution that eliminated the computation of values in some inactive output positions by recognizing the input cells in the ground state, which aimed to reduce the redundancy computation in 3D voxel convolution. The proposed Recurrent Residual Module (RRM) [36] eliminates computation on overlapping areas between neighboring frames in a 2D video clip.

**Efficient Deep Learning for Videos.** Video understanding is much more computationally expensive than image recognition. To improve the efficiency of video processing, Pan *et al*. [36] utilizes dynamic sparse matrix-vector multiplication techniques and temporal redundancy of videos to realize speedup on ASICs such as EIE [14]. CP-Net [31] aggregates information from potential correspondences in video representation to improve video understanding. Wang *et al*. [57,58] proposed to utilize correspondence in time as a supervision signal.

Besides, A lot of efforts have been paid to the acceleration of general purpose deep learning computing, including pruning [15, 18, 34], weight quantization [56, 65] and hardware-aware neural architecture search [50, 51, 59]. These methods are orthogonal to video-specific efficient deep learning approaches.

However, to the best of our knowledge, all the above methods are tailored for 2D understanding and 2D video processing. None of them takes consideration of the special properties of 3D data and typical 3D operations like sparse convolutions [11], which is the major topic of this paper.

## 3. Method

### 3.1. Task Definition

We first introduce the 3D point cloud video task. A clip of point clouds $\mathbf{X}$ with $T$ frames are given, which is defined as

$$\mathbf{X} = \{X_t \mid t = 0, \ldots, T\} \tag{1}$$

where $X_t = \{x_t^i \mid i = 0, \ldots, N_t\}$ denotes point cloud with $N_t$ points at time $t$ in clip $\mathbf{X}$. For point cloud video tasks , each frame $X_t$ will be processed by a network backbone (*e.g*., MinkUNet [3]), the corresponding output feature of

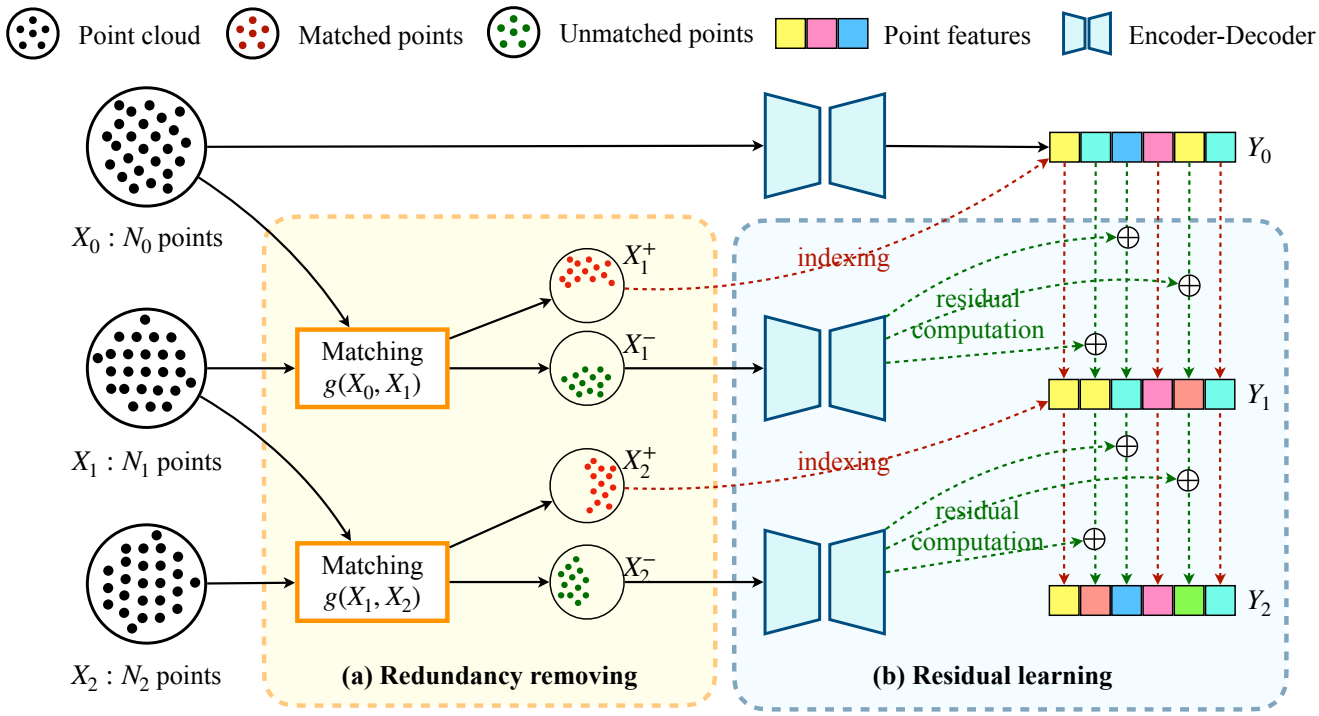


Figure 2: **Model** overview. EVE consists of (a) redundancy removing module and (b) residual learning module. Given a point cloud clip $\mathbf{X} = \{X_t \mid t = 0, \cdots, T\}$, the first frame $X_0$ is processed by an encoder-decoder, getting features $Y_0$ for each point. For the rest frames $X_t (t > 1)$, EVE matches it with previous frame $X_{t-1}$, splitting $X_t$ into $X_t^+$ and $X_t^-$ based on the matched pair's L2 distance (defined in Sec. 3.3.3). For $X_t^+$ (red ones), EVE skipped the computation and use point features with corresponding indices in $Y_{t-1}$. For $X_t^-$ (green ones), EVE computes the residual features. By combining the matched and unmatched parts, EVE outputs features $Y_t$ for each point in $X_t$. Best viewed in color.

each input point is denoted as

$$\mathbf{Y} = \{Y_t \mid t = 0, \ldots, T\} \tag{2}$$

where $Y_t = \{y_t^i \mid i = 0, \ldots, N_t\}$. For each point $x_t^i \in X_t$, its corresponding feature is $y_t^i$. Then for specific tasks (like segmentation), $Y_t$ will be processed by specific heads and output the final labels.

In the rest of paper, we will use two adjacent point cloud frames $X_t = \{x_t^i \mid i = 0, \ldots, N_t\}$ and $X_{t-1} = \{x_{t-1}^j \mid j = 0, \ldots, N_{t-1}\} (t > 1)$ to illustrate the frame redundancy problem and the proposed efficient solver.

### 3.2. Frame Redundancy Problem

Based on universal awareness, adjacent frames in one video are usually visually similar. As shown in Fig. 1, two adjacent frames from one point cloud video, $X_t$ (in blue) and $X_{t-1}$ (in red), have visual similar geometry structure. Current point-based methods and voxel-based methods all process each point cloud frame in full size, thus we assume that a lot of computation is redundant.

Formally, we define the frame redundancy as, features of points in current frame can be represented by features of points in the previous frame, which is,

$$y_t^i = y_{t-1}^j, \text{ where } y_t^i \in Y_t, y_{t-1}^j \in Y_{t-1}. \tag{3}$$

To have a more specific understanding of the redundancy problem, we make a statistics on SemanticKITTI [1] dataset's clips. We iterate through the point cloud clips in train split, and match each frame with previous frame using the Nearest Neighbor (NN) algorithm. If the nearest point pair has the same label, we take it as a redundant pair. For its train split, the redundant ratio is **86.0%** $\pm$ 0.5, which indicates approximately 86.0% points have the same label as previous frame. Intuitively, if we can avoid processing these points by reusing features computed before, a lot computation can be saved.

### 3.3. Efficient 3D Video Engine

To reuse the redundant features, computation of the proposed method should be correlated to the number of points (linear positive correlation) so that computation can be reduced by removing redundant points. Traditional grid-defined convolution network do not meet this criteria be-

cause it requires the feature map to be a dense tensor. Therefore, we opt to use sparse neural network [3, 10, 11] as our network backbone. The sparse tensor representation, even some voxels are removed, can still be processed by sparse convolution with sparsity preserved. The preserved sparsity enables a linear or sub-linear relationship between input number of points with the computation needed.

Based on the sparse tensor representation, we propose the **E**fficient 3D **V**ideo **E**ngine (EVE), which contains a *redundancy removing* module to match adjacent point cloud frames, and a *residual learning* module to learn the residual information. Fig. 2 illustrates our concept. Algo. 1 provides the PyTorch style pseudo-code of EVE. We will first give an overview of EVE, and then describe each module in details.

#### 3.3.1 Overview

For the first frame, because there is no previous frame to match with, EVE will process it solely in full size as the base frame for the next frame. Formally, EVE processes the first frame $X_0$ with encoder-decoder $f(\cdot)$, getting its feature set $Y_0$,

$$Y_0 = f(X_0). \tag{4}$$

For consequent frame $X_t (t > 1)$, EVE transforms it and matches it with previous base frame $X_{t-1}$ to find the redundancy. The matching function $g(\cdot)$ will be discussed in details in Sec. 3.3.2. All points will be matched with their nearest neighbourhood after transformation, and point pairs with distance below a given threshold are accepted as successfully matched pairs, $\mathcal{P}$, and others are unsuccessfully matched pairs, $\mathcal{Q}$:

$$\mathcal{P}, \mathcal{Q} = g(X_t, X_{t-1}). \tag{5}$$

Points in $\mathcal{P}$ will inherit the features of their corresponding points with a negligible computational complexity. For points in $\mathcal{Q}$, a designed residual learning method will compute their features, which will be discussed in Sec. 3.3.3. Combining two parts together, we get features $Y_t$ as,

$$Y_t = Y_t^- \cup Y_t^+, \tag{6}$$

where $Y_t^+$ is features of the successfully matched points and $Y_t^-$ is features of the unsuccessfully matched points in $X_t$.

#### 3.3.2 Redundancy Removing module

Next we describe the instantiation for the matching function $g(\cdot)$. Although correspondence in time has long been researched [31, 36, 57, 58], it is still challenging to utilize it to improve efficiency due to the irregular matching structure. Because traditional convolution is defined on grids of voxels, if the matched voxels are removed, the sparse feature map cannot be directly convoluted.

Algorithm 1: Pseudocode of EVE in a PyTorch-like style

```
# f: sparse conv based encoder-decoder network
# g: matching function
# alpha: ratio of successful matched points
# T: number of frames in a clip

y = []

for t in range(T):
  N = x[t].size(0)

  if t == 0:
    yt = f(x[t])  # directly process 1st frame

  else:
    # matching indices sorted by distance: N x C
    cur_match, prev_match = g(x[t], x[t-1])

    # propogate feature from last frame:
    # match_num x C
    match_num = int(N * alpha)
    yt_plus = y[t-1][prev_match[:match_num]]

    # residual feature of left points
    # res_num x C
    res_num = int(N * (1 - alpha))
    left_behind_idx = cur_match[-res_num:]
    tr = f(x[t-1][left_behind_idx])
    yt_minus = tr + y[t-1][prev_match[-res_num:]]

    # union
    yt = zeros(N, dim_out)  # init empty tensor
    yt[cur_match[:match_num]] = yt_plus
    yt[cur_match[-res_num:]] = yt_minus

  y.append(yt)
```

Instead of directly subtracting frames, we opt to use geometry distance based matching function to detect the redundant points between adjacent frames. In general, we will find the semantically closest point $x_{t-1}^{ji}$ in $X_{t-1}$ for every $x_t^i$ in $X_t$. If the geometry distance (L2 distance) after registration between $x_t^i$ and $x_{t-1}^{ji}$ is below a threshold $\delta$, we take it a successful match. Otherwise, we do not consider it a successful match. Formally, it can be represented as $g(X_t, X_{t-1})$ that:

$$\begin{aligned} &\mathcal{P}, \mathcal{Q} = g(X_t, X_{t-1}), \\ &\mathcal{P} = \{(x_t^i, x_{t-1}^{ji}) \mid ||x_t^i - x_{t-1}^{ji}||^2 \leq \delta\}, \\ &\mathcal{Q} = \{(x_t^i, x_{t-1}^{ji}) \mid ||x_t^i - x_{t-1}^{ji}||^2 > \delta\}, \end{aligned} \tag{7}$$

where $\mathcal{P}$ and $\mathcal{Q}$ are successfully and unsuccessfully matched points pairs set based on point L2 distance. In practice, we manually set the value of $\delta$ so that a fix ratio of points, $\alpha$, (`alpha` in Algo. 1) are in $\mathcal{P}$ and others are in $\mathcal{Q}$.

After matching, we reuse the features in every pair in $\mathcal{P}$ and computation of these points can be saved,

$$Y_t^+ = \{y_{t-1}^{ji} \mid \text{where } (x_t^i, x_{t-1}^{ji}) \in \mathcal{P}\}. \tag{8}$$

One natural choice to find the closest point $x_{t-1}^{ji}$ is the Nearest Neighbor search (NN) algorithm. It has been

widely adopted in 3D deep learning [20, 41]. NN algorithm directly matches each point with its nearest neighbor point, such that for each point $x_t^i$,

$$x_{t-1}^{ji} = \arg\min_j ||x_t^i - x_{t-1}^j||^2 \text{ for } x_{t-1}^j \in X_{t-1}. \quad (9)$$

Although NN matching is better than directly subtracting scheme, the matching is still sensitive to camera movement and sampling deviation. Thus we need a more accurate instantiation of matching function. Here we describe the Iterative Closest Point (ICP) algorithm, which iteratively finds an optimal transformation $[R, \mathbf{t}] : X_t \to X_{t-1}$ that minimize the distance between given sets till convergence. Formally, in each iteration, ICP minimizes the error

$$E(R, \mathbf{t}) = \sum_i^N ||(Rx_t^i + \mathbf{t}) - x_{t-1}^{ji}||, \quad (10)$$

where $x_{t-1}^{ji}$ is $x_t^i$'s nearest point in $X_{t-1}$ calculated by NN matching. And $X_t$ will be updated by $R$ and $\mathbf{t}$ for new iteration until the average distance between $X_t$ and $X_{t-1}$ is below a given threshold or the number of iteration exceeds the maximum number.

Through these iteratively affine transformation, $X_t$ is transformed to align with $X_{t-1}$, so that the problem of camera movement and sampling deviation is alleviated and the matching is more accurate.

#### 3.3.3 Residual Learning Module

Points in $\mathcal{Q}$ will be processed by a residual learning module $f(\cdot)$. The output of residual learning module is $Y_t^-$, which will be union with $Y_t^+$ to get the full feature set $Y_t$.

One straightforward method to process the partial feature map is directly processing it with $f$ (Eq. 4),

$$Y_t^- = f(X_t^-) \quad (11)$$

where $X_t^- = \{x_t^i \mid (x_t^i, x_{t-1}^{ji}) \in \mathcal{Q}\}$, is the unsuccessfully matched points set.

The drawback of this simple method is obvious that it reduces point cloud resolution, which has been proved to impair CNN's performance [6, 17]. And more importantly, the partial point cloud $X_t^-$ does not preserve the full geometry information as the full-size point cloud, thus it's difficult to maintain accuracy by solely processing $X_t^-$.

To deal with the problems above, we draw inspiration from recent works on residual connection [16, 30, 31, 31] and propose a residual learning module to learn the residual information of the unsuccessfully matched points in the previous frame. Specifically, we utilize the corresponding point $x_{t-1}^{ji}$ in $\mathcal{Q}$ as base feature for $x_t^i$ and calculate the residual information as Eq. 12 shows.

$$Y_t^- = f(X_t^-) + f(X_{t-1}^-), \quad (12)$$

where $X_t^-$ is the unsuccessfully matched points, $X_{t-1}^- = \{x_{t-1}^{ji} \mid (x_t^i, x_{t-1}^{ji}) \in \mathcal{Q}\}$ are corresponding points at time $t-1$.

$Y_t^-$ do not lose much context information thanks to the base features in the previous frame, which is calculated with complete context information. And the computed residual features introduce new information from current frame. This mechanism enables efficiently extracting features from the residual point features while maintaining performance.

## 4. Experiments

### 4.1. Experiments setup

All the experiments are on semantic segmentation task, and use the mean Intersection over Union (mIoU) as evaluation protocol. Our running environment is GTX 2080Ti GPU and Intel Xeon E5-2678 v3 CPU.

We select MinkUNet [3] as our baseline method. Based on our experiments, we manually set $T$ as 5 without specification. The default choice for fusion type in residual learning module is residual connection, and the default matching algorithm in redundancy removing module we use is Iterative Closest Point (ICP). Without specification, the matching threshold is set to ensure 50% points are successfully matched. We measure FLOPs of Sparse Convolution [3] which is used in both MinkUNet and EVE by multiplying the size of kernel map with the number of input channels and output channels. The number of points of each scene is fixed to ensure the comparison is fair.

During training, we use Stochastic Gradient Descent (SGD) [43] as the optimizer. For model with Efficient 3D Video Engine (EVE), we import the weights of encoder in model without EVE as pre-trained weights for EVE, and freeze the encoder-decoder for the first frame.

### 4.2. Datasets

**SemanticKITTI** We use SemanticKITTI [1] dataset single scan task [1] to evaluate the performance of Efficient Video Engine's (EVE). SemanticKITTI is based on the odometry dataset of the KITTI Vision Benchmark [8]. It consists of 22 sequences, splitting sequences 0 to 10 as training set, and 11 to 21 as test set. Overall, it contains 23201 full 3D scans for training and 20351 for testing.

We use voxel size 0.05 as default to voxelize the point cloud scene without specification. We randomly choose 50000 points from each scene so that one GPU contains one video clip in training. We use cosine annealing scheduler as the learning rate scheduler, and the base learning rate is set as 0.24 for baseline training, and 0.12 for EVE training. And in evaluation, we process the full point cloud without random sampling.

| method | domain | size | convolution | val mIoU | test mIoU | FLOPs(G) | memory(M) |
|---|---|---|---|---|---|---|---|
| PointNet [40] | 3D | 50K pts | point-based | - | 14.6 | | |
| PointNet++ [41] | | | | - | 20.1 | | |
| TangentConv [53] | | | | - | 40.9 | | |
| RandLA-Net [20] | | | | - | 50.3 | | |
| SqueezeSeg | 2D | 64 × 2048 pixels | pixel-based | - | 29.5 | | |
| SqueezeSegV2 [60] | | | | - | 39.7 | 13.6 | |
| DarkNet21 [1] | | | | - | 47.4 | 213.1 | |
| DarkNet53 [1] | | | | - | 49.9 | 377.1 | |
| MinkNet [3] | 3D | 80K pts | voxel-based | 57.0 | 52.1 | 119.2 | ∼665 |
| **EVE** ($\alpha$=50%) | | 80K pts | | 56.0 | 51.5 | 89.6 | ∼393 |
| MinkNet [3] | | 50K pts | | 51.7 | - | 86.8 | ∼631 |
| **EVE** ($\alpha$=60%) | | 50K pts | | 52.2 | - | 53.9 | ∼375 |

Table 1: Semantic segmentation result for **SemanticKITTI**. The proposed EVE can maintain segmentation accuracy while reduce computation. Compared with the same baseline with lower resolution input, EVE only needs similar FLOPs but achieve much higher accuracy. EVE is also more computation friendly and more accurate than 2D counterpart.

| Method | test mIoU | GFLOPs |
|---|---|---|
| MinkNet [3] | 75.9 | 84.7 |
| EVE ($\alpha$=40%) | **75.2** | 58.7 |
| EVE ($\alpha$=50%) | 74.2 | 51.9 |
| EVE ($\alpha$=60%) | 74.3 | **44.8** |

Table 2: Semantic segmentation result for **Synthia4D**.

**Synthia4D** We also experiment on Synthia4D [44] to evaluate EVE's performance. Synthia4D is a large synthetic dataset designed to facilitate the training of deep neural networks for visual inference in driving scenarios. Photo-realistic renderings are generated from a virtual city, allowing dense and precise annotations of 13 semantic classes, together with pixel-accurate depth. We follow the train/-val/test split as prescribed by [3].

We use voxel size 0.15 as default to voxelize the point cloud scene. We randomly choose 50000 points from each scene so that one GPU contains one video clip in training. And we follow pre-processing in [3], which removes the outlier points to reduce computation and ignores reserved and void points. We use cosine annealing scheduler as the learning rate scheduler, and the base learning rate is set as 0.12 for EVE training. During evaluation, we process the full point cloud without random sampling.

### 4.3. Results on Semantic Segmentation

Tab. 1 compares the MinkUNet based EVE method with current state-of-the-art methods and Fig. 4 shows some qualitative results of EVE on the validation split. Given these results, EVE's performance is superior to strong pixel-based and point-based baselines. Compared with voxel-based method, we achieve higher performance (51.7% *v.s.* 52.2%) but need much less computation (86.8 GFLOPs *v.s.* 53.9 GFLOPs) and memory usage (631M *v.s.* 375M). Category-wise (Fig. 5), EVE also keeps up with the baseline method. The largest absolute decrease are observed for *"parking"* (-0.051), *"truck"* (-0.048) and *"terrain"* (-0.029). These categories are all physically close to the road, the matching accuracy of which we analyze might be prone to the LiDAR scanning wave. Even though, the absolute drop is still less than 0.05. On the other hand, categories like *"bicyclist"* (+0.146) and *"person"* (+0.059) which have distance with road, obtain a higher performance compared with baseline. This demonstrates that EVE has the ability to maintain or improve the accuracy while reduce redundant computation under different circumstances.

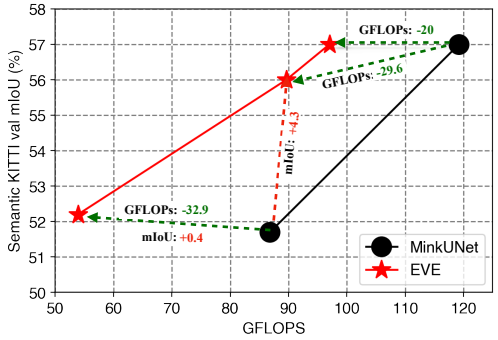


Figure 3: **Mean IoU / computation** comparisons on SemanticKITTI. EVE has a better performance / computation balance. With similar computation, EVE achieve much higher mIoU (+4.1) than baseline method (the red dotted line). With similar performance, EVE reduces up to 32.9 GFLOPs (the green dotted line).

Fig. 3 compares EVE with MinkUNet [3] on both mIoU and needed computation resource. With similar computation overhead, EVE can achieves higher mIoU (the red dot-

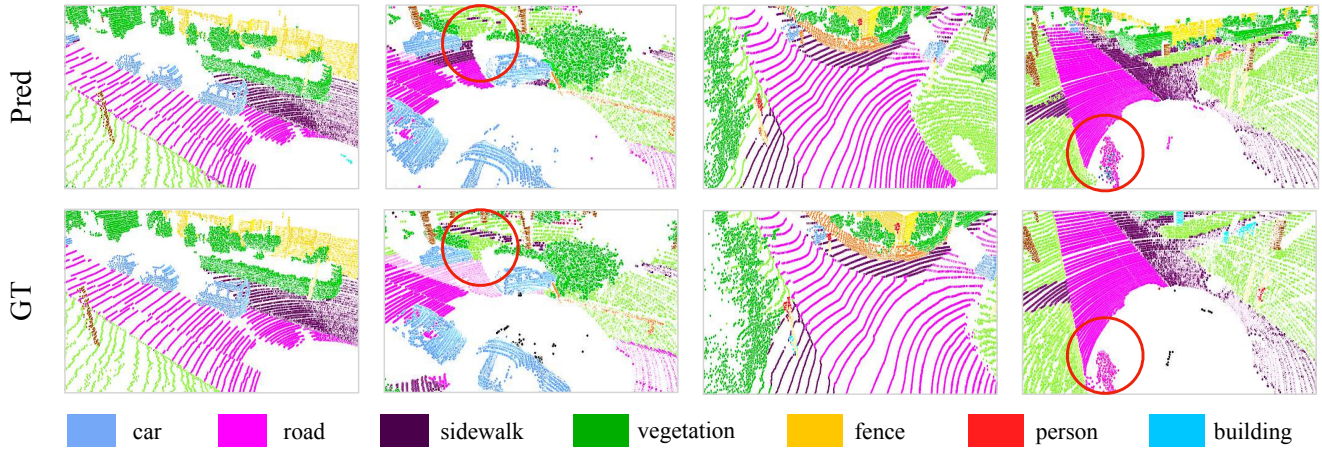


Figure 4: **Qualitative results** of EVE on the validation set of SemanticKITTI. Images in the first come from EVE's prediction, and images in the second row is the ground truth result. Red circles show failure cases.

ted line) compared with baseline method. And to achieve the same mIoU, EVE needs less GFLOPs (the green dotted line), which demonstrates EVE is more efficient.

For point-based state-of-the-art methods RandLA-Net [20], we use latency instead of FLOPs to measure because it does not process one scene at one pass. Its average latency for one frame is **2.01 s**. In comparison, EVE needs **0.22 s** to processes one frame in average, which is **almost an order of magnitude faster** than point-based state-of-the-art method RandLA-Net which is tailored for efficient inference. These results demonstrate that EVE can reduce a lot computation while preserve high mIoU. EVE also outperforms its pixel-based / point-based counterparts with much better mIoU and less computation.

Tab. 2 evaluates EVE on Synthia4D. The results also demonstrate that EVE outperforms strong baseline method with a better mIoU / computation balance.

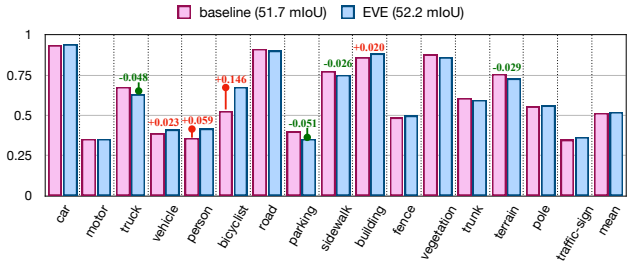


Figure 5: **Per-category IoU on SemanticKITTI**: EVE *v.s.* MinkUNet baseline. The highlighted categories are the 4 highest absolute drop (green digits) and increase (red digits). The largest absolute decrease are observed for *"parking"* (-0.051), *"truck"* (-0.048), *"terrain"* (-0.029) and *"sidewalk"* (-0.026). In general, EVE maintains similar performance as baseline for all categories. (Best viewed in color)

### 4.4. Ablation Study

**Matching accuracy.** We compare different matching function introduced in Sec. 3.3.2 on several metrics. The matching accuracy is defined as follow. For each match pair in $\mathcal{P}$ and $\mathcal{Q}$ (Eq. 7), if the corresponding label is the same, we take the matching as correct. Otherwise the matching is wrong.

Tab. 3 (a) and (b) show that the high matching accuracy is important for EVE. With the same match ratio (50%), EVE with Iterative Closest Point (ICP) performs better (+5.1 mIoU) than that with Nearest Neighbor (NN). Although NN matching accuracy is high enough to ensure point pairs in $\mathcal{P}$ are correctly matched, the performance is also hurt. This result indicates that matching accuracy for point pairs in in $\mathcal{Q}$ is also important.

To further validate this idea, we compare EVE's performance with different ICP hyper-parameters settings on SemanticKITTI. We change the ICP convergence threshold and maximum matching iteration to control the matching accuracy. The result is shown in Tab. 3 (b). We could see a clear positive relationship between matching accuracy and mIoU, which aligns with our hypothesis above.

**Matching ratio.** We experiment with using different matching ratio $\alpha$ (defined in Sec. 3.3.2). Tab. 3 (c) shows validation mIoU and GFLOPs of EVE with different matching ratio. In general, higher $\alpha$ means more computation is reused from previous frame thus less GFLOPs. But when $\alpha$ is too high ($\geq 60\%$ in our case), residual learning module cannot get enough information from the residual point cloud, thus the validation mIoU starts to drop.

We desire reuse as many point features as possible, but

| match algo. | size | match acc. | val mIoU |
|---|---|---|---|
| NN | 80K pts | ∼89.6% | 50.90 |
| ICP | | ∼96.0% | **56.0** |
| NN | 50K pts | ∼85.7% | 46.70 |
| ICP | | ∼95.9% | **51.6** |

(a) Matching function comparison

| match acc. | val mIoU |
|---|---|
| ∼88.3% | 47.6% |
| ∼90.1% | 49.1% |
| ∼92.3% | 51.0% |
| ∼95.9% | 51.6% |

(b) Matching accuracy comparison

| $\alpha$ | GFLOPs | val mIoU |
|---|---|---|
| 0% | 86.8 | 51.7% |
| 25% | 80.3 | 51.5% |
| 50% | 62.7 | 51.6% |
| 60% | 53.9 | **52.2**% |
| 75% | 39.9 | 50.1% |

(c) Matching ratio comparison

Table 3: **Ablation experiments** on redundancy matching module. **(a), (b)**: Under different matching setting, the matching accuracy has the key impact on EVE's performance with different input point cloud size. **(c)**: higher matching ratio $\alpha$ leads to less GFLOPs, but when $\alpha$ exceeds a threshold (60%), val mIoU drops.

| fusion function | size | val mIoU |
|---|---|---|
| in-place | 80K pts | 51.2 |
| residual | | **56.0** |
| in-place | 50K pts | 51.0 |
| residual | | **51.6** |

Table 4: **Fusion function comparison**. Residual connection performs better than in-place operation with different input size.

| $T$ | 1 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|
| FLOPs(G) | 86.8 | 75.4 | 64.5 | 62.7 | 55.0 |
| val mIoU (%) | 51.7 | **52.0** | 51.8 | 51.6 | 51.7 |

Table 5: **Comparison on number of frames**. We fix matching ratio $\alpha$ as 50% and number of points as 50k. Higher $T$ leads to more computation saving. EVE successfully maintains the validation mIoU with different $T$.

we also need to leave enough points so the residual learning module gets enough information. Based on this result, we choose 50% and 60% as the default value for $\alpha$.

**Fusion function.** We compare different fusion algorithms introduced in Sec. 3.3.3. Tab. 4 shows performance of in-place fusion and residual fusion with different size of point cloud as input. EVE with residual fusion consistently outperforms EVE with in-place fusion, especially the point cloud size is huge (+4.8 mIoU). This result indicates that residual fusion is one key factor to EVE, which aligns with our assumption that, directly processing the residual point cloud will fail as the loss of information.

**Number of frames in a clip.** Tab. 5 compares EVE's performance with different number of frames, $T$. In general, higher $T$ leads to more computation saving. EVE successfully maintains the validation mIoU with different $T$, and it even outperforms baseline method by 0.3 mIoU when $T = 3$. This result demonstrates that EVE is more efficient than the baseline method. And EVE has the potential in bringing in more temporal information to increase baseline performance.

We also notice that as $T$ increases, the relative saved computation decreases. That's because the relationship between average computation and number of frames in a clip, $T$, is $\frac{\alpha(T-1)+1}{T} = \frac{1-\alpha}{T} + \alpha$ (here we assume the saving computation is linear to the matching ratio $\alpha$). When $\alpha$ is fixed, the saved computation would be less with increasing $T$ as the derivative of $T$ is negative. Therefore, we do not need to increase $T$ infinitely, and we select 5 as $T$'s default value due to analysis above.

**Failure cases.** Red circles in Fig. 4 shows some failure cases. We observe that failure cases appear most between *"road"* and other categories (like *"sidewalk"* in the provided case). We speculate this is due to the redundancy removing failure as the irregular point pattern and LiDAR scanning wave of the road. Improving the matching accuracy around the road would be a possible direction for future research.

## 5. Conclusion

In this paper, we propose the Efficient 3D Video Engine (EVE) based on the frame redundancy observation in 3D video understanding tasks. Our experiments reveal that 1) Frame redundancy in point cloud video is huge, over 60% computation is repetitive using current mainstream methods. By matching and reusing features with previous frames, the redundant computation can be saved. 2) For non-redundant points, it's better to model it as residual features to their nearest points in the previous frame rather than directly process it. 3) Matching accuracy is a key factor to the whole EVE pipeline, more accurate matching will improve EVE performance. We hope that EVE can foster future works in 3D video tasks.

## 6. Acknowledgement

This work is supported in part by the National Key R&D Program of China, No. 2017YFA0700800, National Natural Science Foundation of China under Grants 61772332 and Shanghai Qi Zhi Institute, SHEITC(2018-RGZN-02046). Thanks Haotian Tang, Sucheng Qian, Chang Ge in Shanghai Jiao Tong University for valuable discussions.

## References


[1] Jens Behley, Martin Garbade, Andres Milioto, Jan Quenzel, Sven Behnke, Cyrill Stachniss, and Juergen Gall. Semantickitti: A dataset for semantic scene understanding of lidar sequences. In *ICCV*, 2019.

[2] Joao Carreira and Andrew Zisserman. Quo vadis, action recognition? a new model and the kinetics dataset. In *CVPR*, pages 6299–6308, 2017.

[3] Christopher Choy, JunYoung Gwak, and Silvio Savarese. 4d spatio-temporal convnets: Minkowski convolutional neural networks. In *CVPR*, pages 3075–3084, 2019.

[4] Jeffrey Donahue, Lisa Anne Hendricks, Sergio Guadarrama, Marcus Rohrbach, Subhashini Venugopalan, Kate Saenko, and Trevor Darrell. Long-term recurrent convolutional networks for visual recognition and description. In *CVPR*, pages 2625–2634, 2015.

[5] Hao-Shu Fang, Chenxi Wang, Minghao Gou, and Cewu Lu. Graspnet-1billion: A large-scale benchmark for general object grasping. In *CVPR*, pages 11444–11453, 2020.

[6] Christoph Feichtenhofer. X3d: Expanding architectures for efficient video recognition. In *ICCV*, 2020.

[7] Christoph Feichtenhofer, Haoqi Fan, Jitendra Malik, and Kaiming He. Slowfast networks for video recognition. *arXiv preprint arXiv:1812.03982*, 2018.

[8] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3354–3361. IEEE, 2012.

[9] Raghav Goyal, Samira Ebrahimi Kahou, Vincent Michalski, Joanna Materzynska, Susanne Westphal, Heuna Kim, Valentin Haenel, Ingo Fruend, Peter Yianilos, Moritz Mueller-Freitag, et al. The" something something" video database for learning and evaluating visual common sense. In *ICCV*, 2017.

[10] Ben Graham. Sparse 3d convolutional neural networks. In *BMVC*, pages 150.1–150.9, September 2015.

[11] Benjamin Graham, Martin Engelcke, and Laurens van der Maaten. 3d semantic segmentation with submanifold sparse convolutional networks. In *CVPR*, pages 9224–9232, 2018.

[12] Chunhui Gu, Chen Sun, David A Ross, Carl Vondrick, Caroline Pantofaru, Yeqing Li, Sudheendra Vijayanarasimhan, George Toderici, Susanna Ricco, Rahul Sukthankar, et al. Ava: A video dataset of spatio-temporally localized atomic visual actions. In *CVPR*, pages 6047–6056, 2018.

[13] Michelle Guo, Edward Chou, De-An Huang, Shuran Song, Serena Yeung, and Li Fei-Fei. Neural graph matching networks for fewshot 3d action recognition. In *ECCV*, pages 653–669, 2018.

[14] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. Eie: efficient inference engine on compressed deep neural network. In *ISCA*, pages 243–254. IEEE, 2016.

[15] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CVPR*, pages 770–778, 2016.

[17] Tong He, Zhi Zhang, Hang Zhang, Zhongyue Zhang, Junyuan Xie, and Mu Li. Bag of tricks to train convolutional neural networks for image classification. In *CVPR*, 2018.

[18] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. Amc: Automl for model compression and acceleration on mobile devices. In *ECCV*, pages 784–800, 2018.

[19] JF Hu, WS Zheng, J Lai, and J Zhang. Jointly learning heterogeneous features for rgb-d activity recognition. *IEEE transactions on pattern analysis and machine intelligence*, 39(11):2186–2200, 2017.

[20] Qingyong Hu, Bo Yang, Linhai Xie, Stefano Rosa, Yulan Guo, Zhihua Wang, Niki Trigoni, and Andrew Markham. Randla-net: Efficient semantic segmentation of large-scale point clouds. *CVPR*, 2020.

[21] Mingyang Jiang, Yiran Wu, Tianqi Zhao, Zelin Zhao, and Cewu Lu. Pointsift: A sift-like network module for 3d point cloud semantic segmentation. *arXiv preprint arXiv:1807.00652*, 2018.

[22] Will Kay, Joao Carreira, Karen Simonyan, Brian Zhang, Chloe Hillier, Sudheendra Vijayanarasimhan, Fabio Viola, Tim Green, Trevor Back, Paul Natsev, et al. The kinetics human action video dataset. *arXiv preprint arXiv:1705.06950*, 2017.

[23] Jiefeng Li, Can Wang, Wentao Liu, Chen Qian, and Cewu Lu. Hmor: Hierarchical multi-person ordinal relations for monocular multi-person 3d pose estimation. *arXiv preprint arXiv:2008.00206*, 2020.

[24] Wanqing Li, Zhengyou Zhang, and Zicheng Liu. Action recognition based on a bag of 3d points. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition-Workshops*, pages 9–14. IEEE, 2010.

[25] Yangyan Li, Rui Bu, Mingchao Sun, Wei Wu, Xinhan Di, and Baoquan Chen. Pointcnn: Convolution on x-transformed points. In *NIPS*, pages 820–830, 2018.

[26] Yong-Lu Li, Xinpeng Liu, Han Lu, Shiyi Wang, Junqi Liu, Jiefeng Li, and Cewu Lu. Detailed 2d-3d joint representation for human-object interaction. In *CVPR*, 2020.

[27] Yong-Lu Li, Liang Xu, Xijie Huang, Xinpeng Liu, Ze Ma, Mingyang Chen, Shiyi Wang, Hao-Shu Fang, and Cewu Lu. Hake: Human activity knowledge engine. *arXiv preprint arXiv:1904.06539*, 2019.

[28] Yong-Lu Li, Liang Xu, Xinpeng Liu, Xijie Huang, Yue Xu, Shiyi Wang, Hao-Shu Fang, Ze Ma, Mingyang Chen, and Cewu Lu. Pastanet: Toward human activity knowledge engine. In *CVPR*, pages 382–391, 2020.

[29] Yong-Lu Li, Siyuan Zhou, Xijie Huang, Liang Xu, Ze Ma, Hao-Shu Fang, Yanfeng Wang, and Cewu Lu. Transferable interactiveness knowledge for human-object interaction detection. In *CVPR*, 2019.

[30] Ji Lin, Chuang Gan, and Song Han. Temporal shift module for efficient video understanding. *arXiv preprint arXiv:1811.08383*, 2018.

[31] Xingyu Liu, Joon-Young Lee, and Hailin Jin. Learning video representations from correspondence proposals. In *CVPR*, pages 4273–4281, 2019.

[32] Xingyu Liu, Charles R Qi, and Leonidas J Guibas. Flownet3d: Learning scene flow in 3d point clouds. In *CVPR*, pages 529–537, 2019.

[33] Xingyu Liu, Mengyuan Yan, and Jeannette Bohg. Meteornet: Deep learning on dynamic 3d point cloud sequences. In *ICCV*, pages 9246–9255, 2019.

[34] Zechun Liu, Haoyuan Mu, Xiangyu Zhang, Zichao Guo, Xin Yang, Tim Kwang-Ting Cheng, and Jian Sun. Metapruning: Meta learning for automatic neural network channel pruning. *arXiv preprint arXiv:1903.10258*, 2019.

[35] Jiageng Mao, Xiaogang Wang, and Hongsheng Li. Interpolated convolutional networks for 3d point cloud understanding. *arXiv preprint arXiv:1908.04512*, 2019.

[36] Bowen Pan, Wuwei Lin, Xiaolin Fang, Chaoqin Huang, Bolei Zhou, and Cewu Lu. Recurrent residual module for fast inference in videos. In *CVPR*, pages 1536–1545, 2018.

[37] Bo Pang, Kaiwen Zha, Hanwen Cao, Chen Shi, and Cewu Lu. Deep rnn framework for visual sequential applications. In *CVPR*, pages 423–432, 2019.

[38] Bo Pang, Kaiwen Zha, Hanwen Cao, Jiajun Tang, Minghui Yu, and Cewu Lu. Complex sequential understanding through the awareness of spatial and temporal concepts. *Nature Machine Intelligence*, 2(5):245–253, 2020.

[39] Bo Pang, Kaiwen Zha, Yifan Zhang, and Cewu Lu. Further understanding videos through adverbs: A new video task. In *AAAI*, pages 11823–11830, 2020.

[40] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *CVPR*, pages 652–660, 2017.

[41] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *NIPS*, pages 5099–5108, 2017.

[42] Gernot Riegler, Ali Osman Ulusoy, and Andreas Geiger. Octnet: Learning deep 3d representations at high resolutions. In *CVPR*, pages 3577–3586, 2017.

[43] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

[44] German Ros, Laura Sellart, Joanna Materzynska, David Vazquez, and Antonio M Lopez. The synthia dataset: A large collection of synthetic images for semantic segmentation of urban scenes. In *CVPR*, pages 3234–3243, 2016.

[45] Amir Shahroudy, Jun Liu, Tian-Tsong Ng, and Gang Wang. Ntu rgb+ d: A large scale dataset for 3d human activity analysis. In *CVPR*, pages 1010–1019, 2016.

[46] Gunnar A Sigurdsson, Gül Varol, Xiaolong Wang, Ali Farhadi, Ivan Laptev, and Abhinav Gupta. Hollywood in homes: Crowdsourcing data collection for activity understanding. In *ECCV*, pages 510–526. Springer, 2016.

[47] Karen Simonyan and Andrew Zisserman. Two-stream convolutional networks for action recognition in videos. In *NIPS*, 2014.

[48] Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. Ucf101: A dataset of 101 human actions classes from videos in the wild. *arXiv preprint arXiv:1212.0402*, 2012.

[49] Hang Su, Varun Jampani, Deqing Sun, Subhransu Maji, Evangelos Kalogerakis, Ming-Hsuan Yang, and Jan Kautz. Splatnet: Sparse lattice networks for point cloud processing. In *CVPR*, pages 2530–2539, 2018.

[50] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. In *CVPR*, pages 2820–2828, 2019.

[51] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, pages 6105–6114, 2019.

[52] Jiajun Tang, Jin Xia, Xinzhi Mu, Bo Pang, and Cewu Lu. Asynchronous interaction aggregation for action detection. *arXiv preprint arXiv:2004.07485*, 2020.

[53] Maxim Tatarchenko, Jaesik Park, Vladlen Koltun, and Qian-Yi Zhou. Tangent convolutions for dense prediction in 3d. In *CVPR*, pages 3887–3896, 2018.

[54] Hugues Thomas, Charles R Qi, Jean-Emmanuel Deschaud, Beatriz Marcotegui, François Goulette, and Leonidas J Guibas. Kpconv: Flexible and deformable convolution for point clouds. *arXiv preprint arXiv:1904.08889*, 2019.

[55] Du Tran, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. Learning spatiotemporal features with 3d convolutional networks. In *ICCV*, pages 4489–4497, 2015.

[56] Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. Haq: Hardware-aware automated quantization with mixed precision. In *CVPR*, pages 8612–8620, 2019.

[57] Xiaolong Wang and Abhinav Gupta. Unsupervised learning of visual representations using videos. In *ICCV*, 2015.

[58] Xiaolong Wang, Allan Jabri, and Alexei A. Efros. Learning correspondence from the cycle-consistency of time. In *CVPR*, 2019.

[59] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *CVPR*, pages 10734–10742, 2019.

[60] Bichen Wu, Xuanyu Zhou, Sicheng Zhao, Xiangyu Yue, and Kurt Keutzer. Squeezesegv2: Improved model structure and unsupervised domain adaptation for road-object segmentation from a lidar point cloud. In *ICRA*, pages 4376–4382. IEEE, 2019.

[61] Wenxuan Wu, Zhongang Qi, and Li Fuxin. Pointconv: Deep convolutional networks on 3d point clouds. In *CVPR*, pages 9621–9630, 2019.

[62] Zuxuan Wu, Xi Wang, Yu-Gang Jiang, Hao Ye, and Xiangyang Xue. Modeling spatial-temporal clues in a hybrid deep learning framework for video classification. In *ACMMM*, pages 461–470, 2015.

[63] Jin Xia, Jiajun Tang, and Cewu Lu. Three branches: Detecting actions with richer features. *arXiv preprint arXiv:1908.04519*, 2019.

[64] Yifan Xu, Tianqi Fan, Mingye Xu, Long Zeng, and Yu Qiao. Spidercnn: Deep learning on point sets with parameterized convolutional filters. In *ECCV*, pages 87–102, 2018.

[65] Aojun Zhou, Anbang Yao, Yiwen Guo, Lin Xu, and Yurong Chen. Incremental network quantization: Towards lossless cnns with low-precision weights. *arXiv preprint arXiv:1702.03044*, 2017.

[66] Bolei Zhou, Alex Andonian, Aude Oliva, and Antonio Torralba. Temporal relational reasoning in videos. In *ECCV*, pages 803–818, 2018.