Deakin University
Faculty of Science, Engineering and Built Environment
School of IT



Towards Video Understanding: Isolated Sign Language Recognition by X3D in PyTorchVideo

SIT332 – Robotics, Computer Vision and Speech Processing

Lecturer: Dr. Duc Thanh Nguyen

Student Name: Bao Minh Tran

Student ID: s224236373

Email: s224236373@deakin.edu.au

September 9^{th}, 2026

Abstract

Action recognition models are increasingly deployed on devices whose compute budget is far smaller than the one they were designed on. `X3D` [6] addresses this by expanding a small 2D architecture one axis at a time, trading accuracy against cost explicitly. This report summarises `X3D` and its realisation in `PyTorchVideo` [4], then asks how well its `Kinetics-400` representation transfers to isolated sign language recognition. We fine–tune three `Kinetics-400`–pretrained variants on the `AUTSL` dataset [23] of 226 Turkish signs at six freezing depths, from no frozen block, i.e. the whole network is trained, to five, i.e. only the classifier head is trained. The "best" configuration, in terms of accuracy–trainable parameters trade–off, `X3D_M` with two frozen blocks, reaches 82.28% top–1 on the test split at 4.90 GFLOPs per clip. The implementation can be found in this repository and the data is available in this link.

Contents

1 Introduction **2**

2 Related work **3**

3 X3D: A technical summary **4**
3.1 X2D architecture 4
3.2 Expansion operations and mechanisms 4
3.3 X3D family 6

4 Experiments **8**
4.1 Datasets 8
4.2 Implementation details 9
4.3 Evaluation metrics 10
4.4 Main results 11
4.4.1 Learning behaviour on train/validation sets 11
4.4.2 Final evaluation on test set 13
4.4.3 Generalising to our data 13

5 Conclusion and discussion **15**

1 Introduction

Action Recognition is a dynamic and leading research area in Computer Vision, primarily developing efficient models to detect and understand human actions from video data [10]. Its development is closely tied with several real-world applications. In medical environment, the number of patients usually outnumber the number of skilled staff. Therefore, medical robots are essentially important to reduce the workload, and thus, they must be smoothly integrated and capable of understanding the current states and work without explicit instructions from staff [24]. Another application is early criminal activity recognition from surveillance cameras in real-time manner [16], and other applications can be found in [10, 14]. Due to their prevalence and significant impacts in real-world practices, various models have been proposed, e.g. X3D [6], SlowFast [7], Two-streams CNN–based networks [22], and MViT [5, 15].

Nonetheless, there are non-negligible obstacles associated with Action Recognition applications, and the above-mentioned models cannot overcome all of these. As surveyed in [10, 14], one action is likely performed in various ways by different people, video quality is worsened in outdoor recordings, insufficient annotated data, some actions cannot be well-interpreted and many more. Importantly, Computer Vision applications are mostly deployed to edge devices whose computing capacities are limited while demanding low latency and real–time inference accuracy [13, 29]. Therefore, this report is interested in exploring a model, namely X3D [6], originally implemented using `PyTorchVideo` [4], which offers competitive efficacy in terms of floating point operations (FLOPs) and parameters while maintaining relatively good accuracy, which is installable on edge devices.

Within the realm of action recognition, sign language recognition is particularly intersting due to its practical significance. In daily life, hearing individuals communicate verbally both in–person and online, while deaf people communicate by use of different sign languages, involving plenty of facial expressions, hand and body gestures to express their thoughts. It has posed several difficulties to bridge the communication gap between deaf and hearing people, or even between deaf people using different system of sign languages, especially demanding as fast as we verbally communicate in foreign languages. This topic has been extensively studied, as surveyed in [17]. Similar to real–time speech recognition and translation, the community aims to develop models capable of recognising continuous sign language videos [27]. For sake of simplicity, we shall study isolated sign language recognition tasks [19].

In this work, we are curious about the capacity of pretrained `X3D`, provided by `PyTorchVideo`, can be fine–tuned to understand isolated sign language videos. Also, we are specifically interested in the performance gap between fine–tuning the whole model and partly on the sign language dataset. In brief, the contributions of this report are:

- A technical summary of `X3D`.
- Benchmark different fine-tuning approaches across different `X3D` variants on `AUTSL` dataset.
- A quantitative result on performance of `X3D` on isolated sign language recognition tasks.
- Evaluate `AUTSL`–trained `X3D` on our hand–crafted dataset.

The remainder of this report is structured as follows. In **Sec. 2**, related models are reviewed. We concretely summarise the architecture and key contributions of X3D in **Sec. 3**. The experimental setup is explicitly described in **Sec. 4**. Finally, in **Sec. 5**, we convey certain remarking claims and discuss further on future developments of this work.

2 Related work

A comprehensive survey of proposed methods for different action recognition tasks can be found in [10, 14] and a nicely concise history of development is included in [20]. Hence, we shall confine ourselves to some recent relevant models herein.

SlowFast Networks [7]. This network adopts the Two–stream CNN–based framework [22] for incorporation between spatial and temporal semantics of video data. It was originally inspired by the biological studies in primate visual system, and prematurely mimicking the mechanisms of Parvocellular and Magnocellular to develop the Slow-Fast pathways and lateral connections between them. Evidently, different variants achieved, on Kinetic–400 [11]: 75.6% to 79.8% top–1 accuracy, 92.1% to 93.9% top–5 accuracy, on Kinetic–600 [1]: 78.8% to 81.8%, and on Charades [21]: 39.0 to 45.2 mAP.

EfficientNet [26]. This model relies on the scalability philosophy of DL models, e.g. Residual Networks [9], and platform–aware search of network architecture [25]. Then, by scaling the ConvNets wisely across three dimensions, i.e. depth, width and resolution. Different model variants achieved, on ImageNet dataset [18], 77.1% to 84.3% top–1 accuracy and 93.3% to 97.0% top–5 accuracy, with fewer learnable parameters, from 5.3M to 66M. The **X3D** model was built upon the philosophy in this network type, but offers more computational advantages.

MViT [5, 15]. This model was established by leveraging the *multiscale feature hierarchies* in CNN–based models and the *attention* mechanism in Transformer–based models. By doing this, the model can establish a pyramid of features similar to CNN while globalising the interactions amongst the local regions thanks to self–attention. Therefore, it reduces the locality bias in CNN–based models and simultaneously breaking the permutation invariance in ViT [2], which is semantically wrong in the context of video understanding where temporal relationship must be considered. As a result, it gained, on Kinetic–400 [11], 76.0% to 81.2% top–1 accuracy and 92.1% to 95.1% top–5 accuracy, on Kinetic–600 [1], 82.1% to 83.8% top–1 accuracy and 95.7% to 96.3% top–5 accuracy, and on Charades [21], 43.9 to 47.7 mAP.

MoViNets [13]. This model substantially reduces the peak memory usage and computational budget while performing competitively in the benchmark datasets above. The authors utilise the neural architecture search to determine a good construction of 3D CNN architectures. Then, they introduce stream buffer component to simultaneously reduce memory consumption and maintain long-range temporal dependencies of sub–clips. Still, it loses some accuracy, by roughly 1% in Kinetic-600 [1]. Thus, they recover the accuracy loss by leveraging the ensembling philosophy [12], specifically, they train two model variants independently and combine them by taking the arithmetic mean on the unweighted logits before their final softmax layer.

Indeed, some models, in the family of MoViNets, are better, in terms of GFLOPs and top-1 accuracy, than certain variants of X3D, as reported in Appendix C of [13]. Nonetheless, it was originally implemented in TensorFlow, not PyTorchVideo, though there is an incomplete unofficial implementation, can be found in `this link`. We shall confine ourselves to **X3D** due to its complete availability and technical ease.

3 X3D: A technical summary

This model adopts the idea of extending 2D CNN models to 3D CNN models by considering temporal axis in the data. More precisely, `X3D` generalises the **compound model scaling** introduced in [26] to more axes and starts scaling from a different baseline 2D model, which they called `X2D`. Moreover, they balance the computation and accuracy trade-offs by the proposal of forward expansion and backward contraction. Interestingly, `X3D` was first implemented in `PySlowFast` [3], and enhanced for adaptability to mobile devices in `PyTorchVideo` [4] thereafter. Next, we concretely introduce the `X2D` backbone and notations used.

3.1 X2D architecture

Briefly, `X2D` is inspired by the ResNet [9] architecture and the Fast pathway in SlowFast networks [7]. Because `X2D` does not consist of temporal axis, hence its temporal stride is set to 1. In particular, the **Table 1** presents the conceptual design of `X2D` if and only if $\gamma_\tau = \gamma_t = \gamma_s = \gamma_w = \gamma_b = \gamma_d = 1$. As reported, `X2D` has 20.67M FLOPS and 1.63M parameters. Interestingly, comparing to the SlowFast networks [7], `X2D` can be viewed as the Slow pathway in the SlowFast networks since it solely uses one single frame at once, while its width is much lighter and similar to the Fast pathway.

As the output sizes column of **Table 1** shows, `X2D` preserves the temporal resolution γ_t at every stage but the last while reducing the spatial one, which resembles the SlowFast pathways [7] and thus retains all temporal signal across features per stage.

The difference between 2D and 3D CNNs are attributed to the number of dimensions of the kernel block, particularly, for 2D CNNs, the kernel is a matrix, whereas for 3D CNNs, the kernel is a 3D tensor. We shall denote the 3D kernel block with a spatiotemporal size of $T \times S^2$, where T is the temporal length/duration and S is the width and height.

3.2 Expansion operations and mechanisms

To extend a tiny spatial network `X2D` to a spatiotemporal network `X3D`, [6] recommends six possible expansion operations, defined to scale four dimensions, i.e. temporal, spatial, width and depth as follows.

- **X–Fast**: expands γ_t by increasing the frame rate $1/\gamma_\tau$.
- **X–Temporal**: expands γ_t by increasing the frame rate $1/\gamma_\tau$ and sampling longer subclips.
- **X–Spatial**: expands γ_s by increasing the spatial sampling resolution of input frames.
- **X–Depth**: expands γ_d by increasing the number of layers per residual block res_i.
- **X–Width**: expands γ_w to uniformly increase the number of channels across all layers per residual block res_i.
- **X–Bottleneck**: expands the inner channel width, γ_b, of first two layers in per residual block res_i.

These axes are illustrated in **Figure 1**.

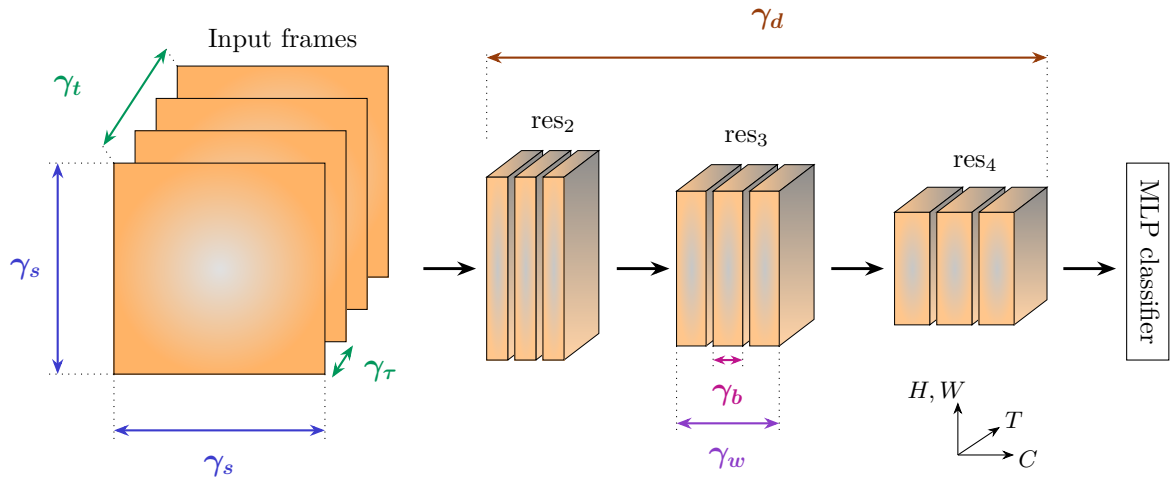


Figure 1: **X3D** networks progressively expand a 2D network across the following axes: temporal duration γ_t, frame rate γ_τ, spatial resolution γ_s, width γ_w, bottleneck width γ_b, and depth γ_d. Redrawn after [6].

Having introduced possible scaling dimensions suggested in the paper, we show two key expansion regimes underpinning their discovery of network architectures. It is noteworthy that, there are different sizes of `X3D` depending how many times `X2D` is progressively scaled using the aforementioned expansion operations. Remarkably, the computation/accuracy trade–off is controlled by utilising a feature selection method, i.e. forward selection and backward elimination [8].

Forward expansion. At each iteration, the algorithm loops through all possibilities by changing a factor in $\Gamma \triangleq \{\gamma_\tau, \gamma_t, \gamma_s, \gamma_w, \gamma_b, \gamma_d\}$ and keeping others unchanged. Two important factors driving the choice of scaling dimensions are. First, the goodness criterion function $J(\Gamma)$, which is chosen to be the accuracy gained by expanding the current model by Γ, and thus, the higher the better. Second, computing resources are of paramount importance, and hence, deserved to be considered during the expansion. Let us denote $C(\Gamma)$ be a complexity criterion function, specifically FLOPs measurement in [6]. Ultimately, each iteration, the algorithm opts out the best balance between accuracy and complexity. Formally, $\Gamma = \underset{\mathcal{Z},\ C(\mathcal{Z})=c}{\arg\max}\ J(\mathcal{Z})$, where $\mathcal{Z}$ are possibilities of varying Γ and c is the target complexity budget.

Remarkably, one may notice, there can be at most 2^6 ways to change, each of which exhausts the computational capacity by varying $c \in \mathbb{R}$. Therefore, they rely on the idea of coordinate descent algorithms [28] and fixing the expansion rate at $c \approx 2$. In order to ensure this expansion rate, [6] provides details how much each scaling dimension can be expanded coordinate–wise.

Backward contraction. Simply speaking, this step is introduced to the algorithm to reduce $C(\mathcal{Z})$ when its target complexity is surpassed in the prior forward expansion, so that $C(\mathcal{Z}) < c$ is guaranteed at each iteration.

3.3 X3D family

stage	filters	output sizes $T \times S^2$	
data layer	stride γ_τ, 1^2	$1\gamma_t \times (112\gamma_s)^2$	
conv_1	1×3^2, 3×1, $24\gamma_w$	$1\gamma_t \times (56\gamma_s)^2$	`blocks[0]`
res_2	$\begin{bmatrix} 1 \times 1^2, & 24\gamma_b\gamma_w \\ 3 \times 3^2, & 24\gamma_b\gamma_w \\ 1 \times 1^2, & 24\gamma_w \end{bmatrix} \times \gamma_d$	$1\gamma_t \times (28\gamma_s)^2$	`blocks[1]`
res_3	$\begin{bmatrix} 1 \times 1^2, & 48\gamma_b\gamma_w \\ 3 \times 3^2, & 48\gamma_b\gamma_w \\ 1 \times 1^2, & 48\gamma_w \end{bmatrix} \times 2\gamma_d$	$1\gamma_t \times (14\gamma_s)^2$	`blocks[2]`
res_4	$\begin{bmatrix} 1 \times 1^2, & 96\gamma_b\gamma_w \\ 3 \times 3^2, & 96\gamma_b\gamma_w \\ 1 \times 1^2, & 96\gamma_w \end{bmatrix} \times 5\gamma_d$	$1\gamma_t \times (7\gamma_s)^2$	`blocks[3]`
res_5	$\begin{bmatrix} 1 \times 1^2, & 192\gamma_b\gamma_w \\ 3 \times 3^2, & 192\gamma_b\gamma_w \\ 1 \times 1^2, & 192\gamma_w \end{bmatrix} \times 3\gamma_d$	$1\gamma_t \times (4\gamma_s)^2$	`blocks[4]`
conv_5	1×1^2, $192\gamma_b\gamma_w$	$1\gamma_t \times (4\gamma_s)^2$	`blocks[5]`
pool_5	$1\gamma_t \times (4\gamma_s)^2$	$1 \times 1 \times 1$	
fc_1	1×1^2, 2048	$1 \times 1 \times 1$	
fc_2	1×1^2, #classes	$1 \times 1 \times 1$	

Table 1: **X3D architecture** [6]. The dimensions of kernels are denoted by $\{T \times S^2, C\}$ for temporal, spatial, and channel sizes. Strides are denoted as {temporal stride, spatial stride^2}. This network is expanded using factors $\{\gamma_\tau, \gamma_t, \gamma_s, \gamma_w, \gamma_b, \gamma_d\}$ to form `X3D`. The braces on the right give the corresponding entry of the `blocks` module list of the `PyTorchVideo` implementation [4], i.e. the stem `blocks[0]`, the four residual stages `blocks[1]` to `blocks[4]`, and the head `blocks[5]`. The data layer is a sampling step so no trainable parameters required.

Concretely, `X3D` is a feature extractor of a video. Conceptually, this can be represented as in **Figure 2** below:

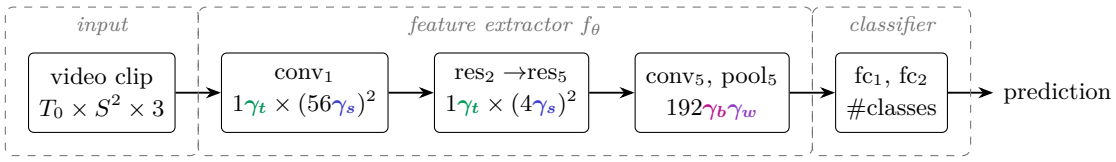


Figure 2: `X3D` as a video feature extractor followed by a linear classifier. The shape below each stage is its output size, following **Table 1**. In the input stage, T_0 is the total frames in the raw video input and 3 implies each video's frame is in RGB color space.

The 2D CNN processes the whole image by learning local information stage by stage, until the final blocks are sufficiently dense, and hence, a classifier, typically MLP, is used to yield final

prediction. The 3D CNN generalises this concept by considering multiple frames at once, thereby allowing the model to understand the underlying temporal dependence inherently in video data, and keeps other aspects of 2D CNN as is. Put simply, both 2D and 3D CNNs are simply feature extractors, said differently, it learns to transform raw data into numerically meaningful vectors.

Due to limited computing resources, we shall skip the expansion of `X2D` to `X3D`, and directly leverage different variants of `X3D` models, supported by `PyTorchVideo`, to conduct the experiments in the upcoming sections.

4 Experiments

4.1 Datasets

In short, their motivation for creation of the `AUTSL` dataset is to replicate the real–world scenarios when using sign languages. The dataset contains frequently used vocabularies in spoken language, and keep the dataset balanced in terms of motion characteristics and time taken to complete a word. **Table 2** below shows an overview of the dataset.

Property	Description
Number of signs	226
Number of signers	43
Total samples	38,336
Number of different backgrounds	20
Mean sample per sign	169.6
Modalities	RGB, depth, skeleton
RGB and depth resolution	512×512
FPS	30

Table 2: Statistics of AUTSL dataset [23].

Regarding challenges, the videos span a wide variety of static and dynamic backgrounds, the latter containing objects moving behind the signer. The authors further selected signs that are very similar in motion and separated only by subtle gestures, included compound signs built from simpler ones, and recruited signers of differing backgrounds, so that personal signing style varies across the dataset.

In this experiment, we leverage the available `AUTSL` dataset given in Kaggle, accessible via this link. This dataset only comprises of RGB-colored videos, for a comprehensive dataset of `AUTSL`, including different video formats of the same data, refer to this link. Therefore, by utilisation of Kaggle dataset, we limit ourselves to benchmark `X3D` performance on RGB–colored videos. Lastly, the downloaded data was already partitioned, and thus, we shall not split the dataset by our own.

Set	Number of samples	Percentage
Train	28,142	77.5%
Validation	4,418	12.2%
Test	3,742	10.3%
Total	36,302	100%

Table 3: Number of samples inspected directly from the dataset.

Apparently, **Table 2** and **Table 3** differ in the total number of samples, suggesting a data shift between the Kaggle release and the original. We therefore re–inspect the dataset from its filenames in **Figure 3** and **Figure 4**, which still count every clip. Of these, 11 are truncated and cannot be decoded, i.e. 10 in train and 1 in test, and are excluded from both training and testing, leaving the 28,132 and 3,741 clips actually used.

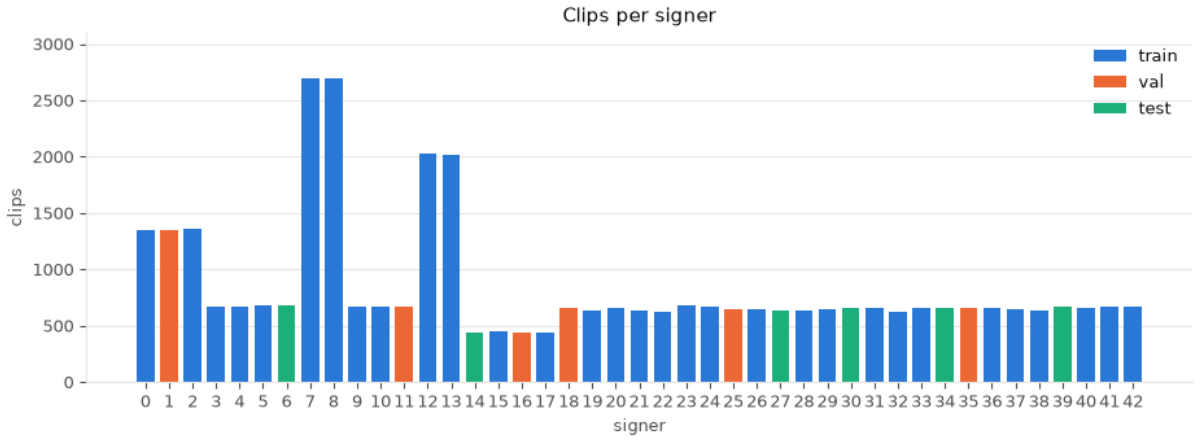


Figure 3: Inspection of number of clips produced by each signer.

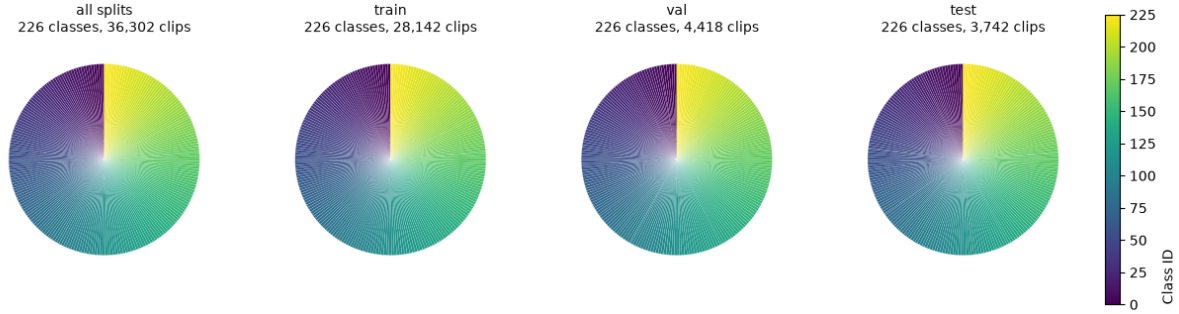


Figure 4: Distribution of sign classes across the dataset splits. Every split covers all 226 classes. **All splits:** 36,302 clips, with min 121, median 162, mean 160.6 and max 164 clips per class. **train:** 28,142 clips (min 90, median 126, mean 124.5, max 127) **validation:** 4,418 clips (min 12, median 20, mean 19.5, max 20) **test:** 3,742 clips (min 10, median 17, mean 16.6, max 17).

Dataset building. According to the models' sizes in this code, we adopt the clip geometry summarised in **Table 4**.

Model size	num_frames	crop_size
X3D_XS	4	160
X3D_S	13	160
X3D_M	16	224

Table 4: Number of frames and crop size for each video.

AUTSL dataset must be built with the matching num_frames and crop_size, otherwise the weights load but the model sees inputs it was never trained for.

4.2 Implementation details

In this experiment, we do not benchmark the model against other methods. Alternatively, we benchmark different fine-tuning approaches of different models' sizes.

In the original codebase, X3D was pretrained on Kinetics-400 dataset [11], and thus the final softmax layer does not match the number of classes in AUTSL dataset. Therefore, we replace

the final linear projection layer, to reduce the number of classes. Formally, let $g(\mathbf{x})$ is the final fully connected layer of pretrained `X3D`, given by:

$$g : \mathbb{R}^k \mapsto \mathbb{R}^{400} \tag{1}$$

We shall replace it by:

$$g : \mathbb{R}^k \mapsto \mathbb{R}^{226} \tag{2}$$

The remaining settings are as follows.

- **Data.** The `AUTSL` train split, i.e. $28{,}142 - 10 = 28{,}132$ clips over 226 classes, unless a subset is configured.
- **Preprocessing.** Uniform temporal subsampling to the variant's clip length, resizing to its crop size, random horizontal flip, scaling to $[0, 1]$ and per–channel normalisation with mean 0.45 and standard deviation 0.225.
- **Initialisation.** `Kinetics-400` weights for the backbone, and a freshly initialised linear head.
- **Optimisation.** Cross–entropy loss with `Adam`. About fine–tuning different components, frozen blocks are excluded from the optimiser and held in evaluation mode, so that their `BatchNorm` running statistics do not drift.
- **Hyperparameters.** Epochs = 100, batch size = 16 and learning rate = 10^{-4}.
- **Hardware.** AMD EPYC 7402P (24 cores - 48 threads), 256 GB DDR4 RAM and 4x NVIDIA A100 (40GB HBM2 each).

4.3 Evaluation metrics

Since `X3D` is designed around an accuracy–complexity trade-off rather than accuracy alone [6], we report metrics along two axes, i.e. how well a model recognises signs, and how expensive it is to store and to evaluate.

Top–k accuracy. Let the test split be $\mathcal{D} = \{(\mathbf{V}_i, y_i)\}_{i=1}^{N}$, where $\mathbf{V}_i$ is a clip and $y_i \in \{1, \ldots, C\}$ is its sign label, $C = 226$ being the number of classes. For each clip the network, i.e. the final `softmax` layer, produces a score vector $\mathbf{z}_i = (z_{i1}, \ldots, z_{iC}) \in \mathbb{R}^C$, and we write $\mathcal{R}_k(\mathbf{V}_i) \subseteq \{1, \ldots, C\}$ to denote the subset of $k \leq C$ classes receiving the largest scores. The top–k accuracy:

$$\text{top–}k = \frac{1}{N} \sum_{i=1}^{N} \mathbf{1}\left\{ y_i \in \mathcal{R}_k(\mathbf{V}_i) \right\}, \tag{3}$$

where $\mathbf{1}\{\cdot\}$ denotes the indicator function.

We evaluate $k \in \{1, 3, 5\}$, and note that $\mathcal{R}_1 \subseteq \mathcal{R}_3 \subseteq \mathcal{R}_5 \Rightarrow$ Top–1 $\leq$ Top–3 $\leq$ Top–5. Top–1 is the strict accuracy, i.e. the fraction of clips whose most probable class is correct, and is the quantity of practical interest for a recognition system. Top–5 is the conventional companion metric on `Kinetics-400` [11] and `ImageNet` [18]. Top–3 is included to give balance between strict metric top–1 and easy metric top–5. The margin between Top–1 and Top–3 therefore diagnoses the nature of the residual errors, i.e. a small margin indicates genuine failures, whereas a large one indicates near misses within a small group of mutually confusable signs.

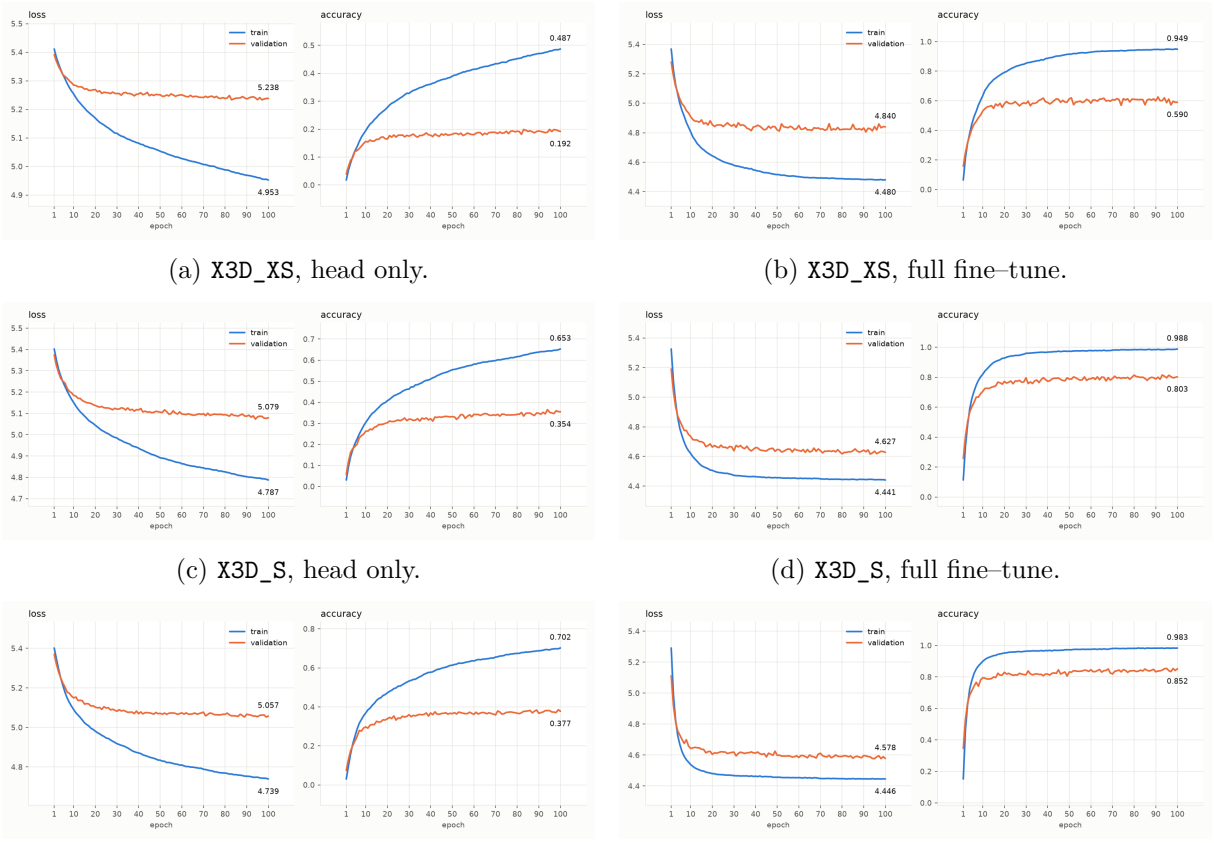
Total and trainable parameters. Let θ be the full set of parameters of a model and $\theta_{\text{tr}} \subseteq \theta$ the subset left unfrozen during training. We report the cardinalities $|\theta|$ and $|\theta_{\text{tr}}|$. Because our experiments compare fine–tuning strategies rather than competing architectures, this pair separates two costs that a single count would conflate. $|\theta|$ governs the memory required to store and deploy the network, whereas $|\theta_{\text{tr}}|$ governs the gradients and optimiser state held during training, and thus the cost of adaptation.

GFLOPs. We quantify inference cost by the number of floating–point operations required for one forward pass, expressed in units of 10^9. Following the convention adopted in [6, 26], we count multiply–adds and quote the cost of a *single* clip. Importantly, GFLOPs is an analytic quantity, independent of hardware, and is therefore a fair proxy for arithmetic effort but not for wall–clock latency.

4.4 Main results

4.4.1 Learning behaviour on train/validation sets

We train every combination of the three variants `X3D_XS`, `X3D_S` and `X3D_M` with 0 to 5 of the six blocks held frozen. **Figure 5** shows the two extremes of this sweep, and **Table 5** collects the best validation top–1 of every run.



(a) `X3D_XS`, head only.

(b) `X3D_XS`, full fine–tune.

(c) `X3D_S`, head only.

(d) `X3D_S`, full fine–tune.

(e) `X3D_M`, head only.

(f) `X3D_M`, full fine–tune.

Figure 5: Per–epoch cross–entropy loss and top–1 accuracy on the train and validation splits, for the two extreme fine–tuning depths of each variant. Freezing five blocks leaves only the classifier trainable, whereas freezing none updates the whole network.

Frozen	X3D_XS		X3D_S		X3D_M	
blocks	train	val	train	val	train	val
0	95.4	63.9	98.3	**81.5**	98.4	85.4
1	95.4	**64.7**	98.4	80.4	98.3	85.5
2	95.5	64.2	98.4	81.0	98.9	**86.1**
3	93.7	58.7	98.5	75.8	98.7	83.4
4	92.4	49.2	97.7	71.1	97.8	77.1
5	48.0	19.9	64.3	36.5	69.8	38.5

Table 5: Top–1 accuracy (%) on the train and validation splits, for each variant and freezing depth. The variants differ only in the clip they sample, i.e. `X3D_XS` takes 4 frames at 160^2, `X3D_S` 13 frames at 160^2 and `X3D_M` 16 frames at 224^2. The best validation entry per variant is bolded.

Underfitting. Training the classifier head only amount to underfitting, evidently 48.0%, 64.3% and 69.8% accuracy for `X3D_XS`, `X3D_S` and `X3D_M`, respectively. This is indicative of the `Kinetics-400` pretrained weights may not contribute vastly to models' performance on `AUTSL`, and mostly rely on the learning capacity of the classifier block.

Pretrained weights may not help. Looking at learning curves in **Figure 5**, at the first few epochs, they accuracy is extremely low. Semantically, this is true, since `Kinetics-400` and `AUTSL` are constructed for two completely different tasks. Nonetheless, we cannot strongly claim that this does not help, since the pretrained weights may yield a relatively good initial starting point to optimise the models, in terms of local optimality and convergence speed. Further ablation studies must be conducted to compare between train–from–scratch vs. train–from–pretrained.

Overfitting. Interestingly, enabling more blocks in the backbone model trainable flips the situation. It immediately overfits to the training data for the first block adjacent to the classifier head, evidently by the huge gap between train and validation accuracy. Nevertheless, the gap is converging as we allows more parameters learnable, and astonishingly, such train–validation gap is narrower in `X3D_M` than in `X3D_XS`.

Clip sampling outweighs adaptation depth. The ordering `X3D_M` > `X3D_S` > `X3D_XS` holds at every freezing depth, and the margin is large. More specifically, on validation accuracy, `X3D_M` with three blocks frozen (83.4%), fully fine–tuned "best" `X3D_S` (81.5%), and one block frozen "best" `X3D_XS` (64.7%). From **Table 6**, three variants differ only in the clip drawn from each video and the cropping size each frame. Therefore, the gain is attribute to temporal and spatial sampling rather than to capacity, and it is traded off with GFLOPs.

Recommendation. From the above–mentioned results and analysis, we shall pick `X3D_M` with 2 frozen blocks as our "best" model, due to its performance competency and good trade–off with trainable parameters.

4.4.2 Final evaluation on test set

Model	Frozen	Top–1 (%)	Top–3 (%)	Top–5 (%)	Params (M)	Trainable (M)	GFLOPs
X3D_XS	0	54.08	72.36	78.40	3.438	3.438	0.63
X3D_XS	1	**56.91**	74.58	80.78	3.438	3.437	0.63
X3D_XS	2	53.49	73.08	79.76	3.438	3.422	0.63
X3D_XS	3	49.67	69.31	76.90	3.438	3.348	0.63
X3D_XS	4	38.89	58.49	66.61	3.438	2.779	0.63
X3D_XS	5	14.68	26.01	31.36	3.438	1.432	0.63
X3D_S	0	75.67	89.74	93.42	3.438	3.438	2.03
X3D_S	1	**76.56**	89.76	93.26	3.438	3.437	2.03
X3D_S	2	75.67	89.60	93.50	3.438	3.422	2.03
X3D_S	3	69.85	85.89	90.35	3.438	3.348	2.03
X3D_S	4	63.46	81.05	86.93	3.438	2.779	2.03
X3D_S	5	29.35	43.89	50.12	3.438	1.432	2.03
X3D_M	0	**82.38**	93.50	95.94	3.438	3.438	4.90
X3D_M	1	80.83	92.70	95.64	3.438	3.437	4.90
X3D_M	2	82.28	94.49	96.52	3.438	3.422	4.90
X3D_M	3	77.17	90.56	94.04	3.438	3.348	4.90
X3D_M	4	73.62	89.20	93.40	3.438	2.779	4.90
X3D_M	5	34.86	50.98	57.31	3.438	1.432	4.90

Table 6: Performance on the AUTSL test split (3,741 clips evaluated over 226 classes). The best top–1 per variant is bolded.

Freezing depth. Training the whole network rather than the head alone adds 39.4, 46.3 and 47.5 top–1 points for X3D_XS, X3D_S and X3D_M, at 2.4× the trainable parameters. Interestingly, by unfreezing only one block adjacent to the head block, i.e. $\texttt{res}_5$, lifts X3D_M from 34.86% to 73.62% for 1.35M extra trainable parameters, likewise for X3D_XS and X3D_S. Thereafter, unfreezing other blocks gain more accuracy, though not a big jump as before.

Model size. Moving from X3D_XS to X3D_S costs 3.2× the GFLOPs and returns 19.7 top–1 points; the further step to X3D_M costs 2.4× again for only 5.8. The longer clip is therefore not wasted on isolated signs, but its value saturates.

Error type. The top–1 to top–3 margin is widest for X3D_XS and narrowest for X3D_M. We shall claim, though insignificantly, that the small variant's errors are largely near misses among confusable signs, whereas the residual errors of X3D_M are more often outright failures.

4.4.3 Generalising to our data

We separate this section from the testing above, since the test split still shares underlying patterns with the train split, e.g. signer identity, background and experimental bias. To build the dataset we watched the public clips, replicated their motions and annotated with the same numeric id, as the model learnt indexed rather than text–based classes. Also, it poses more challenges, as we cannot mimic the mouth movements that some signs involve, our lighting is poor and introduces scattering, and AUTSL records the full body whereas we recorded only the upper half.

Setup. We recorded ten clips on a laptop camera at 1620 × 1080, each imitating one AUTSL sign and named after the test clip whose label it borrows. In other words, the footage itself is

our own and only the labelling convention is shared. They were converted to `mp4` and passed through the pipeline of **Sec. 4.2**. We run the inference on this hand–crafted data using CPU on MacBook M3 Pro.

Clip	True class	Predicted (top–1)	Confidence	Inference time (s)
0	161	161	1.000	0.369
1	77	77	0.667	0.331
2	22	18	0.974	0.343
3	157	213	0.860	0.310
4	112	32	0.680	0.328
5	128	54	0.703	0.313
6	14	14	1.000	0.316
7	5	5	1.000	0.340
8	23	23	0.987	0.332
9	155	155	0.999	0.404

Table 7: Predictions of `X3D_M` with two frozen blocks on ten self–recorded clips. Confidence is the `softmax` probability of the predicted class.

Accuracy. As collected in **Table 7**, the model recognises six of the ten clips, i.e. 60% top–1, 60% top–3 and 70% top–5, yet incomparable with the test results in **Table 6** due to insufficient samples.

Confidence. The wrong predictions are as confident as the right ones, i.e. 0.68 to 0.97 against 0.67 to 1.00. Softmax confidence therefore offers no usable signal for rejecting an out–of–distribution clip.

Latency. Inference took 0.310 to 0.404 seconds per clip, 0.339 on average, on a CPU alone. It suggests the model is deployable for real–time inference.

5 Conclusion and discussion

Conclusion. This report summarised `X3D` [6], the family of video networks obtained by expanding a small 2D architecture one axis at a time, and examined how its `PyTorchVideo` implementation [4] transfers to isolated sign language recognition on `AUTSL` [23]. Our best configuration, `X3D_M` with two frozen blocks, reaches 82.28% top–1, 94.49% top–3 and 96.52% top–5 on the `AUTSL` test split, at 4.90 GFLOPs per clip and 3.42M trainable parameters. Importantly, the emprical results highlight that, it is unnecessary to retrain the whole backbone model.

Regarding fine–tuning approaches, considering each model size (`XS, S, M`), on test accuracy, full fine–tuning is worth between 39.4 and 47.5 top–1 points over training only classifier head, but the gain is unevenly distributed. Unfreezing one further block substantially improves model's prediction accuracy, and freezing the first one or two blocks costs nothing.

Regarding model sizes, considering each fine–tuning strategy, the larger variant repays its cost only in part. On test accuracy, from `X3D_XS` to `X3D_S` returns 14.67 - 24.57 points for 3.2× the GFLOPs, whereas from `X3D_S` to `X3D_M` returns $4.27 - 10.16$ for a further 2.4×. Since all three variants share one parameter count, this is the price of sampling a longer clip rather than of a larger model, suggesting the models are capable of longer video contexts and supporting its generalisation.

Limitations. First, due to resources availability, each configuration was trained once, i.e. no confidence interval was conducted, and thus the reported result is unreliable in terms of seed variance. Second, only the RGB modality was used, leaving the depth and skeleton streams of `AUTSL` unexploited, which may improve models' generalisation capacity. Third, GFLOPs measures arithmetic rather than latency on an actual edge device, and hence we are unable to claim its practical significance on IoT applications. Fourth, the model can only predicts 226 Turkish isolated signs, any thus, fails to predict out–of–distribution data, e.g. other signs in Turkish language system and generalising to other sign language systems.

Future work. `PyTorchVideo`'s `X3D` is faster than the original implementation [6, 3], yet still short of what edge deployment demands. `PyTorchVideo` also supports `EfficientX3D` for that purpose, so future migration to this model may facilitate more inference speed. More importantly, as mentioned earlier, this is an isolated sign language dataset, and thus, inapplicable to real–world demands for real-time translation and continuous video processing. Therefore, this current work may be adopted as pretrained weights, due to its inherent semantic meaning, unlike `Kinetics-400` pretrained models.

References

[1] Carreira, J., Noland, E., Banki-Horvath, A., Hillier, C., Zisserman, A.: A short note about kinetics-600 (2018), https://arxiv.org/abs/1808.01340

[2] Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., Houlsby, N.: An image is worth 16x16 words: Transformers for image recognition at scale. In: 9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021. OpenReview.net (2021), https://openreview.net/forum?id=YicbFdNTTy

[3] Fan, H., Li, Y., Xiong, B., Lo, W.Y., Feichtenhofer, C.: PySlowFast. https://github.com/facebookresearch/SlowFast (2020)

[4] Fan, H., Murrell, T., Wang, H., Alwala, K.V., Li, Y., Li, Y., Xiong, B., Ravi, N., Li, M., Yang, H., Malik, J., Girshick, R., Feiszli, M., Adcock, A., Lo, W.Y., Feichtenhofer, C.: PyTorchVideo: A deep learning library for video understanding. In: Proc. ACM International Conference on Multimedia (ACM MM) (2021), arXiv:2111.09887

[5] Fan, H., Xiong, B., Mangalam, K., Li, Y., Yan, Z., Malik, J., Feichtenhofer, C.: Multiscale vision transformers. In: Proc. IEEE/CVF International Conference on Computer Vision (ICCV). pp. 6824–6835 (2021)

[6] Feichtenhofer, C.: X3D: Expanding architectures for efficient video recognition. In: Proc. IEEE/CVF Conf. Computer Vision and Pattern Recognition (CVPR). pp. 203–213 (2020)

[7] Feichtenhofer, C., Fan, H., Malik, J., He, K.: SlowFast networks for video recognition. In: Proc. IEEE/CVF International Conference on Computer Vision (ICCV). pp. 6202–6211 (2019)

[8] Guyon, I., Elisseeff, A.: An introduction of variable and feature selection. J. Machine Learning Research Special Issue on Variable and Feature Selection **3**, 1157 – 1182 (01 2003). https://doi.org/10.1162/153244303322753616

[9] He, K., Zhang, X., Ren, S., Sun, J.: Deep Residual Learning for Image Recognition . In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 770–778. IEEE Computer Society, Los Alamitos, CA, USA (Jun 2016). https://doi.org/10.1109/CVPR.2016.90, https://doi.ieeecomputersociety.org/10.1109/CVPR.2016.90

[10] Heo, S., Moon, J., Jung, S.K.: Action recognition: A comprehensive survey of tasks, methods, and challenges. ICT Express **12**(1), 32–49 (2026). https://doi.org/https://doi.org/10.1016/j.icte.2025.11.015, https://www.sciencedirect.com/science/article/pii/S2405959525001869

[11] Kay, W., Carreira, J., Simonyan, K., Zhang, B., Hillier, C., Vijayanarasimhan, S., Viola, F., Green, T., Back, T., Natsev, P., Suleyman, M., Zisserman, A.: The Kinetics human action video dataset. arXiv preprint arXiv:1705.06950 (2017)

[12] Kondratyuk, D., Tan, M., Brown, M., Gong, B.: When ensembling smaller models is more efficient than single large models. arXiv preprint arXiv:2005.00570 (2020)

[13] Kondratyuk, D., Yuan, L., Li, Y., Zhang, L., Tan, M., Brown, M., Gong, B.: Movinets: Mobile video networks for efficient video recognition. In: 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 16015–16025 (2021). https://doi.org/10.1109/CVPR46437.2021.01576

[14] Kong, Y., Fu, Y.: Human action recognition and prediction: A survey. International Journal of Computer Vision **130**, 1366–1401 (2022). https://doi.org/10.1007/s11263-022-01594-9

[15] Li, Y., Wu, C.Y., Fan, H., Mangalam, K., Xiong, B., Malik, J., Feichtenhofer, C.: MViTv2: Improved multiscale vision transformers for classification and detection. In: Proc. IEEE/CVF Conf. Computer Vision and Pattern Recognition (CVPR). pp. 4804–4814 (2022)

[16] LIM, J.Y.: Real-time Aggressive Action Recognition using Deep Learning in a Multifarious Setting (2 2023). https://doi.org/10.26180/22083245.v1, https://bridges.monash.edu/articles/thesis/Real-time_Aggressive_Action_Recognition_using_Deep_Learning_in_a_Multifarious_Setting/22083245

[17] Rastgoo, R., Kiani, K., Escalera, S.: Sign language recognition: A deep survey. Expert Systems with Applications **164**, 113794 (2021). https://doi.org/https://doi.org/10.1016/j.eswa.2020.113794, https://www.sciencedirect.com/science/article/pii/S095741742030614X

[18] Russakovsky, O., Deng, J., Su, H., et al.: Imagenet large scale visual recognition challenge. International Journal of Computer Vision **115**, 211–252 (2015). https://doi.org/10.1007/s11263-015-0816-y

[19] Sarhan, N., Frintrop, S.: Unraveling a decade: A comprehensive survey on isolated sign language recognition. In: 2023 IEEE/CVF International Conference on Computer Vision Workshops (ICCVW). pp. 3202–3211 (2023). https://doi.org/10.1109/ICCVW60793.2023.00345

[20] Shuchang, Z.: A survey on human action recognition. arXiv preprint arXiv:2301.06082 (2022)

[21] Sigurdsson, G.A., Varol, G., Wang, X., Farhadi, A., Laptev, I., Gupta, A.: Hollywood in homes: Crowdsourcing data collection for activity understanding. In: Leibe, B., Matas, J., Sebe, N., Welling, M. (eds.) Computer Vision – ECCV 2016. pp. 510–526. Springer International Publishing, Cham (2016)

[22] Simonyan, K., Zisserman, A.: Two-stream convolutional networks for action recognition in videos. In: Advances in Neural Information Processing Systems (NeurIPS) (2014)

[23] Sincan, O.M., Keles, H.Y.: AUTSL: A large scale multi-modal Turkish sign language dataset and baseline methods. IEEE Access **8**, 181340–181355 (2020)

[24] Stabenow, S., Wagner, L., Knoll, A., Bengler, K., Wilhelm, D.: Action recognition in medical environments for robotic assistance. International Journal of Computer Assisted Radiology and Surgery **21**(2), 241–252 (2026). https://doi.org/10.1007/s11548-025-03551-6

[25] Tan, M., Chen, B., Pang, R., Vasudevan, V., Sandler, M., Howard, A., Le, Q.V.: MnasNet: Platform-Aware Neural Architecture Search for Mobile . In: 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 2815–2823. IEEE Computer Society, Los Alamitos, CA, USA (Jun 2019). https://doi.org/10.1109/CVPR.2019.00293, https://doi.ieeecomputersociety.org/10.1109/CVPR.2019.00293

[26] Tan, M., Le, Q.V.: EfficientNet: Rethinking model scaling for convolutional neural networks. In: Proc. International Conference on Machine Learning (ICML). pp. 6105–6114 (2019)

[27] Wei, F., Chen, Y.: Improving Continuous Sign Language Recognition with Cross-Lingual Signs . In: 2023 IEEE/CVF International Conference on Computer Vision (ICCV). pp. 23555–23564. IEEE Computer Society, Los Alamitos,

CA, USA (Oct 2023). https://doi.org/10.1109/ICCV51070.2023.02158, https://doi.ieeecomputersociety.org/10.1109/ICCV51070.2023.02158

[28] Wright, S.J.: Coordinate descent algorithms. Mathematical Programming **151**, 3 – 34 (2015), https://api.semanticscholar.org/CorpusID:15284973

[29] Yang, Z., Nahrstedt, K., Guo, H., Zhou, Q.: Deeprt: A soft real time scheduler for computer vision applications on the edge. In: 2021 IEEE/ACM Symposium on Edge Computing (SEC). pp. 271–284 (2021). https://doi.org/10.1145/3453142.3491278