Fact sheet: CVPR 2021 ChaLearn Looking at People Large Scale Signer Independent Isolated SLR Challenge

This is the fact sheet's template for the CVPR 2021 ChaLearn Looking at People Large Scale Signer Independent Isolated SLR Challenge [1]. Please fill out the following sections carefully in a scientific writing style. Then, send the compressed project (in `.zip` format), i.e., the generated PDF, `.tex`, `.bib` and any additional files to `juliojj@gmail.com`, and put in the Subject of the email "CVPR 2021 SLR Challenge / Fact Sheets", following the schedule and instructions provided in the Challenge webpage [1] "*Wining solutions (post-challenge), Fact Sheets*". Note, if you participated in both track, you will need to send one fact sheet per track.

I. Team details

- **Challenge Track** (RGB or RGB+D): RGB
- Team leader name: Mathieu De Coster
- Username on Codalab: m-decoster
- Team leader affiliation: IDLab-AIRO - Ghent University - imec
- Team leader address: Technologiepark Zwijnaarde 126, 9052 Zwijnaarde, Belgium
- Team leader phone number: 003292643366
- Team leader email: mathieu.decoster@ugent.be
- Name of other team members (and affiliation): NA
- Team website URL (if any): NA

II. Contribution details

A. Video Transformer Networks with hand cropping and pose flow

Our goal in this competition is to apply the Video Transformer Network on a publicly available dataset for isolated sign language recognition. We specifically choose the Video Transformer Network because transformers are used in sign language translation with state of the art performance. The Video Transformer Network has already been evaluated on an isolated sign language recognition task before; we now verify its performance on a different, public, dataset. We implement the model and improve its performance by cropping out the hands and encoding movement information in a low-dimensional representation which we call pose flow. As part of our entrance to this competition, we have applied our model for the RGB track with an added depth branch to the problem for the RGB+D track. As our only modification to our model and approach is this added depth branch, the majority of this fact sheet is identical to our fact sheet for the RGB+D track. We achieve a recognition rate of 0.908300 on the validation set and of 0.929200 on the test set of AUTSL.

B. Introduction and Motivation

Our method is based on the Video Transformer Network (VTN). This network architecture was originally introduced by Kozlov *et al.* for action recognition [2]. It consists of a 2D Convolutional Neural Network (CNN) as feature extractor and a multi-head self-attention sequence classifier. De Coster *et al.* applied this method to isolated sign language recognition on the corpus of Flemish sign language [3] and obtained state of the art results on that dataset [4]. Our goal in this competition is to apply and improve the VTN for sign language recognition using the AUTSL dataset [5]. We propose several improvements, mostly related to pre-processing and data augmentation, that increase the classification accuracy of the model. These improvements include additional augmentation (random horizontal flipping and changes in brightness, contrast, and saturation), as well as the cropping of the hands and the introduction of pose flow features. The details are explained in section II-D.

One powerful architecture for sign language recognition is based on 3D CNNs. Specifically, the I3D network [6] has been used to obtain state of the art accuracy on several isolated sign language recognition datasets [7], [8]. This method has the advantage of combining spatial and temporal feature extraction, while the VTN model performs these in separate stages. However, the VTN model allows for the processing of additional features in the temporal stage, without requiring that these features be of the same modality as the RGB features. In fact, this property of the VTN allows us to include pose flow information in the same temporal processing layers as the features extracted from the RGB input frames. In I3D, a separate temporal CNN or other architectural modifications would be required.

The fact that the VTN architecture separates spatial and temporal processing, additionally allows individual tweaking of the spatial processing network (CNN) and temporal processing network (self-attention decoder). For example, if the VTN is applied on a small dataset, one might wish to use ResNet-18 instead of ResNet-34 to reduce the number of trainable parameters. One can also replace the ResNet with a different architecture, for example VGG-16 [9], or tweak the number of heads and layers in the self-attention decoder (or use a different decoder architecture altogether). In I3D, the spatial and temporal processing are intertwined and it is more difficult to alter the model significantly while still using pre-trained weight initialization. Therefore, we chose to work with the VTN as we deemed it allowed more flexibility in our experiments during the competition.

Finally, transformers obtain state of the art results in sign language *translation* [10], [11]. The VTN is therefore an interesting candidate to evaluate for isolated sign language *recognition*.

C. Representative image / workflow diagram of the method

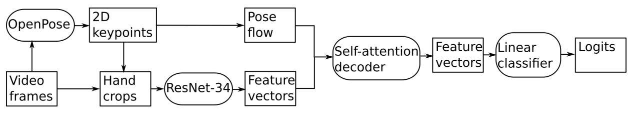

Fig. 1. The pipeline of our approach (not depicted: data augmentation).

D. Detailed method description

For all our experiments, we use PyTorch 1.7.1 [12] and Torchvision 0.8.2 [13]. Furthermore, our implementation is based on PyTorch Lightning 1.1.1 [14].

Figure 1 shows the architecture of our model. First OpenPose is used to extract 2D keypoints from the RGB video frames, specifically using the BODY-135 model [15]. Then, the keypoints of the wrists, elbows and shoulders are used to extract hand crops. The keypoints are also used to extract pose flow information. The hand crop images are processed in a mini-batch by a pre-trained ResNet-34 and the resulting feature vectors are concatenated with the pose flow features. These features are processed using a self-attention decoder and its outputs are classified by a linear classifier to obtain the final output.

1) Dataset preparation: We make a custom validation set for the development stage. We do this by taking a stratified grouped split with 80% of the original training set used as training data and the remaining 20% as validation data. We use person identifiers as groups to make our split signer independent. The resulting label file, provided in the code repository, can be used to reproduce our results on the development set. However, we recommend using the validation set provided by the competition organizers, as this allows for a larger training set.

2) Keypoint extraction: We use OpenPose [16] with the BODY-135 model [15][1] to extract 2D keypoints for all samples. We do this in an offline fashion by using the OpenPose demo binary to extract the keypoints per video file. We store the resulting keypoints as JSON files along with the original MP4 files. Each sample has a corresponding directory of JSON files: one JSON file per frame.

3) Hand cropping: Our hand cropping procedure is based on the work of Simon *et al.* [17]. We extract image crops around the hands using the OpenPose keypoints. We place the crop along the extension of the elbow and wrist. Unlike Simon *et al.*, we compute the (square) crop size based on the distance between the shoulders, such that the size is always linked to the specific camera settings and physical proportions of the person.

The center of the bounding box c is derived from the wrist w and elbow e:

$$c = w + 0.15(w - e). \tag{1}$$

The size s of the bounding box is derived from the distance between the left and right shoulder (l and r):

$$s = 1.2 \left\| l - r \right\|_2 . \tag{2}$$

The factor 0.15 is proposed by Simon *et al.*. The factor 1.2 is empirically obtained by us.

4) Pose flow: We select $K = 53$ keypoints for the computation of pose flow. We only compute the pose flow for keypoints of the hands and the upper body. The face keypoints are not included, as we empirically found no benefit to doing so. We remove following keypoints: $k \in [11, 16]$, $k \in [19, 24]$ and $k \in [65, 134]$.

For each of these keypoints, we compute the angle and magnitude of the motion vector between two subsequent frames. Before computing the pose flow, we pre-process the keypoints of each sample to reduce the impact of keypoint estimation errors and to be signer independent. When OpenPose is unable to estimate a keypoint, it maps it to the origin by predicting $(x, y) = (0, 0)$. We first replace any such keypoint by an estimated keypoint in neighbouring frames. We look for such a replacement keypoint in both previous and subsequent frames. While doing so, we minimize the distance between the original frame and the replacement frame. Afterwards, we normalize each pose, by dividing all keypoints by the length of the neck to account for differences in camera setups and physical properties of the person in the sample.

After these pre-processing steps, we can compute the pose flow as follows. Let P be the keypoints of the entire sample, $P^{(i)}$ the keypoints of frame i and $P_k^{(i)}$ the keypoint k in frame i. Then the motion vector $\mu_k^{(i)}$ for keypoint k in frame i, for $i > 0$, is defined as

$$\mu_k^{(i)} = P_k^{(i)} - P_k^{(i-1)}. \tag{3}$$

The angle $\theta_k^{(i)}$ is then

$$\theta_k^{(i)} = \frac{\mathit{arctan2}(y, x)}{\pi} \tag{4}$$

for $(x, y) = \mu_k^{(i)}$. The normalization factor $\frac{1}{\pi}$ is used to obtain a value in the range $(-1, 1]$. The magnitude $\rho_k^{(i)}$ is defined as the 2-norm of the motion vector:

$$\rho_k^{(i)} = \left\| \mu_k^{(i)} \right\|_2 . \tag{5}$$

We obtain a vector $\left(\theta_k^{(i)}, \rho_k^{(i)}\right)$ for each of the remaining 53 keypoints per frame. For the first frame, $i = 0$, we initialize the pose flow to zero.

[1]The keypoints and their corresponding indices in the pose arrays are given in the source code of OpenPose: `https://github.com/CMU-Perceptual-Computing-Lab/openpose/blob/41d8c087459fae844e477dda50a6f732e70f2cb8/src/openpose/pose/poseParameters.cpp#L149`.

5) Data augmentation: We perform spatial data augmentation on a per clip basis. This means that we fix the parameters of our augmentation pipeline per clip such that all frames in the sample are augmented in the same way.

We first scale the hand crop images to 256 by 256 pixels. Then, we perform a multi-scale and corner crop [18] to 224 by 224 pixels. In a multi-scale corner crop, a crop size is randomly selected, followed by a cropping position: either a centered crop or a crop positioned in one of the four corners of the image. We first randomly select a crop size for the image of 256 by 256 pixels to have a width and height of 256, 215, 181, or 152 pixels. These values are obtained following the implementation of the VTN of Kozlov *et al.* [2]: the initial size of 256 is multiplied with several scales s_i to obtain the resulting crop sizes. These scales are:

$$s_0 = 1, \tag{6}$$

$$s_i = 2^{-\frac{1}{4}} s_{i-1}, \qquad i \in [1, 3]. \tag{7}$$

Then we randomly select a position from the four corners and the center. Finally, the cropping is performed at the selected location and the cropped image is resized to the desired dimensions of 224 by 224 pixels.

We additionally randomly flip the frames horizontally with probability $p = 0.5$ and introduce color jitter to alter the brightness, contrast and saturation. For each of these, the factor by which they are altered is uniformly sampled between 0.5 and 1.5.

For validation and testing of the model, we scale to 256 by 256 pixels and perform a centered crop to 224 by 224 pixels, but do not perform any data augmentation.

The pose flow is not augmented.

6) RGB feature extraction: We use ResNet-34 [19] pre-trained on ImageNet [20] as feature extractor for individual RGB frames. The weights are obtained from the Torchvision model zoo [13]. The RGB channels of image inputs are normalized using the mean μ and standard deviation σ of the ImageNet dataset, given by $\mu = (0.485, 0.456, 0.406)$ and $\sigma = (0.229, 0.224, 0.225)$, respectively. We obtain a 512-dimensional feature vector for each hand by pooling the feature map obtained from the final convolutional block in the spatial dimensions. We concatenate these vectors (one per hand) to obtain our 1024-dimensional feature vector for each sequence element.

7) Self-attention decoder: The self-attention decoder is based on the work of Kozlov *et al.* [2]. In this section, we explain how the self-attention decoder works.

We obtain a feature vector x per frame as output of the ResNet-34 feature extractor. The pose flow of the corresponding frame is a feature vector p. We concatenate x and p and normalize the resulting vector q to have zero mean and unit variance by subtracting the mean and dividing by the standard deviation across all frames in the mini-batch. Then, we use a non-linear transformation to obtain the self-attention decoder inputs r:

$$r = \max(0, qW + b). \tag{8}$$

The dimensionality of q is $1024 + 106$; the dimensionality of r is 1024.

a) Positional encoding: First, the positional encoding is added to the inputs r. The positional encoding is defined as in the seminal work on transformers [21]:

$$PE(p, 2i) = \sin(p/10000^{2i/d_{model}}), \tag{9}$$

$$PE(p, 2i+1) = \cos(p/10000^{2i/d_{model}}), \tag{10}$$

with p the position in the sequence and i the dimension. There are $d_{model} = 1024$ dimensions in our case.

b) Decoder layers: This is followed by four decoder layers. Each layer computes eight-head attention followed by layer normalization [22] and a position wise feed forward network.

The multi-head attention input X (composed of vectors $r + PE(r)$) is first transformed into a query Q, key K and value V using a trained linear transformation. Each head i (of the $n = 8$ heads) computes attention on a subset of the query, key and value,

$$Q^i = QW_Q^i, \tag{11}$$

$$K^i = KW_K^i, \tag{12}$$

$$V^i = VW_V^i, \tag{13}$$

$$A(Q^i, K^i, V^i) = softmax\left(\frac{Q^i K^{i\top}}{\sqrt{1/d_k}}\right) V^i, \tag{14}$$

where weight matrices W_Q^i, W_K^i and W_V^i are used to transform the query, key and value to a lower-dimensional subspace, namely $d_k = d_{model}/n = 1024/8 = 128$. The results are concatenated, added to the residual X, and normalized with layer normalization.

The position wise feed forward network computes

$$f(x) = \max(0, xW_1 + b_1)W_2 + b_2, \tag{15}$$

as per the original transformer [21]. It is also followed by an addition with the residual and subsequently layer normalization.

In both the multi-head attention and position wise feed forward network, dropout ($p = 0.1$) is applied before the addition of the residual signal.

8) Training approach: Per sample, we take a temporally centered segment of 16 frames with a stride of 2, to obtain a temporal receptive field of 32 frames. If the source sample is too short, we repeat the final frame until we obtain 16 frames.

We use mini-batches of 32 samples. Each sample consists of two times 16 frames of RGB inputs (one 224 by 224 pixel image per hand) and pose flow, a 106-dimensional vector. To process the RGB inputs, we construct a mini-batch of size 1024 by concatenating the batch, time, and hand axes. After processing by the ResNet, we perform the inverse to obtain 512 features per hand, per frame, per sample.

The model is trained using early stopping: when the validation loss (categorical cross-entropy) does not decrease for 10 epochs, training is stopped and the model state at the epoch with the lowest loss is saved. We begin training with a learning rate of 1e$-$4, and decrease the learning rate with a factor 0.1 every five epochs. The Adam optimizer [23] is used with $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 1\text{e}{-8}$. Any gradients that are larger than 1 are clipped.

Our models are trained on a single NVIDIA TITAN X (with 12GB VRAM). Because of memory constraints, we use a batch size of 4 and accumulate gradients over 8 batches for an effective batch size of 32.

E. Challenge results and final remarks

TABLE I
LEADERBOARD: RESULTS OBTAINED BY THE PROPOSED APPROACH.

Phase	Track	Rank position	Rec. Rate
Development	RGB	20	0.908300
Test	RGB	14	0.929200

III. ADDITIONAL METHOD DETAILS

- **Did you use any kind of depth information (directly, such as RGBD data, or indirectly such as 3D pose estimation trained on RGBD data), either if during training or testing stage?** ( ) Yes, (X) No
  If yes, please detail:
- **Did you use pre-trained models?** (X) Yes, ( ) No
  If yes, please detail:
  A pre-trained 2D CNN, specifically ResNet-34 [19], was used. This network was pre-trained on ImageNet [20] and the weights were retrieved from the Torchvision model zoo [13]. Moreover, the OpenPose BODY-135 model [16], [15] was used to extract 2D pose keypoints. The ResNet was fine-tuned on the sign language recognition task, while the OpenPose model was frozen and used for offline feature extraction only.
- **Did you use external data?** ( ) Yes, (X) No
  If yes, please detail:
- **Did you use other regularization strategies/terms?** ( ) Yes, (X) No
  If yes, please detail:
- **Did you use handcrafted features?** (X) Yes, ( ) No
  If yes, please detail:
  We crafted features based on OpenPose keypoints. Based on the keypoints extracted by OpenPose, we calculated a representation of movement which we call pose flow. Specifically, per frame and per keypoint, the pose flow consists of the angle and magnitude of the motion vector between this frame and the previous frame. Details are explained in Section II-D.
- **Did you use any face / hand / body detection, alignment or segmentation strategy?** (X) Yes, ( ) No
  If yes, please detail:
  We extracted square image crops around the hands using the OpenPose keypoints. Details are explained in Section II-D.
- **Did you use any pose estimation method?** (X) Yes, ( ) No
  If yes, please detail:
  We used OpenPose (BODY-135 model) for full-body pose estimation. However, the face keypoints were not used and several irrelevant keypoints (such as the lower body) were also dropped. Details are explained in Section II-D.
- **Did you use any fusion strategy of modalities?** (X) Yes, ( ) No
  If yes, please detail:
  We perform early fusion (i.e., before our sequence processing module) of pose flow and features extracted from the RGB frames. This fusion is performed using a non-linear transformation (dense layer with ReLU activation). Details are explained in Section II-D.
- **Did you use ensemble models?** ( ) Yes, (X) No
  If yes, please detail:
- **Did you use any spatio-temporal feature extraction strategy?** (X) Yes, ( ) No
  If yes, please detail:
  Pose flow is extracted from (spatial) keypoints over time. This is detailed in Section II-D.
- **Did you explicitly classify any attribute (e.g. gender)?** ( ) Yes, (X) No
  If yes, please detail:
- **Did you use any bias mitigation technique (e.g. rebalancing training data)?**
  ( ) Yes, (X) No
  If yes, please detail:

IV. CODE REPOSITORY

Code repository: `https://github.com/m-decoster/ChaLearn-2021-LAP`

REFERENCES

[1] ChaLearnLAP. CVPR 2021 ChaLearn Looking at People Large Scale Signer Independent Isolated SLR Challenge. [Online]. Available: http://chalearnlap.cvc.uab.es/challenge/43/description/

[2] A. Kozlov, V. Andronov, and Y. Gritsenko, "Lightweight network architecture for real-time action recognition," in *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, 2020, pp. 2074–2080.

[3] M. Van Herreweghe, M. Vermeerbergen, E. Demey, H. De Durpel, and S. Verstraete, "Het Corpus VGT. Een digitaal open access corpus van videos and annotaties van Vlaamse Gebarentaal, ontwikkeld aan de Universiteit Gent i.s.m. KU Leuven." www.corpusvgt.be, 2015.

[4] M. De Coster, M. Van Herreweghe, and J. Dambre, "Sign language recognition with transformer networks," in *12th International Conference on Language Resources and Evaluation*, 2020.

[5] O. M. Sincan and H. Y. Keles, "AUTSL: A Large Scale Multi-Modal Turkish Sign Language Dataset and Baseline Methods," *IEEE Access*, vol. 8, pp. 181 340–181 355, 2020.

[6] J. Carreira and A. Zisserman, "Quo vadis, action recognition? A new model and the kinetics dataset," in *proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 6299–6308.

[7] D. Li, C. Rodriguez, X. Yu, and H. Li, "Word-level Deep Sign Language Recognition from Video: A New Large-scale Dataset and Methods Comparison," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, March 2020.

[8] S. Albanie, G. Varol, L. Momeni, T. Afouras, J. S. Chung, N. Fox, and A. Zisserman, "BSL-1K: Scaling up co-articulated sign language recognition using mouthing cues," in *European Conference on Computer Vision*, 2020.

[9] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[10] N. C. Camgoz, O. Koller, S. Hadfield, and R. Bowden, "Sign language transformers: Joint end-to-end sign language recognition and translation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 10 023–10 033.

[11] ——, "Multi-channel transformers for multi-articulatory sign language translation," in *European Conference on Computer Vision*. Springer, 2020, pp. 301–319.

[12] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[13] PyTorch. Torchvision. [Online]. Available: https://github.com/pytorch/vision

[14] W. Falcon *et al.*, "PyTorch Lightning," *GitHub. Note: https://github.com/PyTorchLightning/pytorch-lightning*, vol. 3, 2019.

[15] G. Hidalgo, Y. Raaj, H. Idrees, D. Xiang, H. Joo, T. Simon, and Y. Sheikh, "Single-network whole-body pose estimation," in *ICCV*, 2019.

[16] Z. Cao, G. Hidalgo, T. Simon, S.-E. Wei, and Y. Sheikh, "OpenPose: realtime multi-person 2D pose estimation using Part Affinity Fields," in *arXiv preprint arXiv:1812.08008*, 2018.

[17] T. Simon, H. Joo, I. Matthews, and Y. Sheikh, "Hand keypoint detection in single images using multiview bootstrapping," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2017, pp. 1145–1153.

[18] L. Wang, Y. Xiong, Z. Wang, and Y. Qiao, "Towards good practices for very deep two-stream ConvNets," *arXiv preprint arXiv:1507.02159*, 2015.

[19] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[20] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[21] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *arXiv preprint arXiv:1706.03762*, 2017.

[22] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *arXiv preprint arXiv:1607.06450*, 2016.

[23] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.