university of groningen / faculty of science and engineering

HandTalk: American sign language recognition by 3D-CNNs

Bachelor's Project Thesis

Julia Walczynska, s3969436, j.i.walczynska@student.rug.nl,
Supervisors: dr. C.P. Lawrence and prof. dr. H. Jaeger

Abstract: The goal of the project is to build a resource-efficient American sign language classification model suitable for potential deployment on mobile devices. This task is a particularly demanding visual recognition problem due to the nature of the sign language and the importance of each of its five parameters: hand shape, orientation, location, movement, and non-manual expression. In order to capture all of those properties, videos are used instead of images, therefore a 3D convolutional neural network needs to be used. On the other hand, such networks tend to be huge and slow. Recently, attempts to introduce resource-efficient 3D-CNNs have been made. This research investigates whether the resource-efficient MobileNetV2 architecture inflated to the 3D version by using the 3D filters as described in Kopuklu et al., 2019 is a suitable model for the video-based sign language classification. The model was trained and tested on the Word-Level American Sign Language dataset and evaluated using accuracy, precision, recall, and F1-score. The top-1 and top-5 accuracy are compared to Pose-GRU, Pose-TGCN, VGG-GRU, and I3D. The model achieved top-1 accuracy of 51.51%, outperforming Pose-GRU and VGG-GRU. Furthermore, the top-5 accuracy was 86.32%, which is higher than achieved by other approaches.

1 Introduction

Sign language is a natural language that is used predominantly by deaf communities. American Sign Language (ASL) is one of its most popular variations, being a first language throughout Anglo-America and West Africa. It is also considered a lingua franca and, as such, is often learned as a second language by deaf people worldwide. It is important to help deaf people by facilitating their everyday lives. This can be done, for example, by creating a translator from sign language to English, so that people not knowing sign language can easily understand deaf ones. However, to allow that to be done, we need to first create a reliable sign language classification system.

This is a difficult task due to the complex nature of sign language. In ASL, most of the signs represent single words, such as 'cat'. Furthermore, there are 24 signs representing the letters of the English alphabet. These are usually used for spelling words that do not have their own gestures, such as names. This is called fingerspelling or dactylology. Moreover, the signs can be either static or dynamic. The first group includes mainly letters and digits, but there are a few words that are also represented by static signs, for example, 'pray'. However, the majority of the words are represented by dynamic gestures.

Furthermore, the phonology of the ASL is complex as well. The signs may be performed with either one or two hands. The latter group is further divided into symmetric signs, that is, those where the same gesture is performed with both hands, and non-symmetric signs, that is, both hands perform a different gesture. Furthermore, each of the signs has five parameters: hand shape, movement, palm orientation, location, and non-manual markers. Even a small difference can completely change the meaning of the word. For example, the words 'read' and 'dance' differ only in the orientation of hands (see Figure 1.1).

The difference may be even more subtle in the case of words similar in meaning: when the word 'drink' is performed using the c-hand shape, it means

Figure 1.1: ASL signs for 'read' (top row) and 'dance' (bottom row), differing only in hand orientation. Taken from Li et al., 2020.

Figure 1.2: The subtle difference between 'drink (non-alcoholic beverage)' and 'drink (liquor)'. Taken from Cartwright, 2016.

Figure 1.3: One ASL sign representing two different words: 'rice' (top) and 'soup' (bottom). Taken from Li et al., 2020.

drinking something non-alcoholic, while when the modified c-hand shape with three fingers is used, it means drinking liquor (see Figure 1.2). Furthermore, multiple words may correspond to the same sign, such as 'rice' and 'soup' (see Figure 1.3) or even be performed differently by different signers (see Figure 1.4). All of these characteristics and subtle differences make sign language classification a difficult and tricky task, not only in terms of developing an appropriate model but also in terms of collecting and annotating datasets.

Figure 1.4: ASL sign for 'scream' performed differently by two signers. Taken from Li et al., 2020.

1.1 Prior approaches

Many different approaches to sign language classification have been studied. Early attempts often required the use of additional devices, such as sensor gloves and magnetic trackers. Those were combined with the use of methods such as rule-based matching (Kadous et al., 1996), artificial neural networks (Fels & Hinton, 1993) and hidden Markov models (Liang & Ouhyoung, 1998).
Research on sign language classification has become more popular recently, especially in the area of visual-based recognition systems. A SqueezeNet model, for example, achieved an accuracy of 83.29 percent after being trained on over 40 000 images representing letters of the ASL alphabet (Kasukurthi et al., 2019). Another approach used a Hough transform combined with a feed-forward back propagation network. The dataset contained the images of ASL letters and a few simple, static signs. An accuracy of 92.3% was achieved with this method (Munib et al., 2007). Such approaches are suitable for fingerspelling, which in sign languages is only used to represent ideas for which there are no official signs, such as places or names. They may also be useful in the recognition of static signs, which rarely appear in ASL. However, it is not scalable to word-level sign language as the movement is very important for almost all signs, and thus, the model should be trained on videos rather than images to capture it properly.
In Starner et al., 1998 two cameras were used. The first one was mounted on the desk and recorded the signing person from the front. The other camera was placed on the hat worn by the signing person so

that the recorded image was similar to that person's perspective. Their recognition system was based on the hidden Markov model and achieved 92% and 97% accuracy for each camera, respectively, for a 40-word lexicon. The issue with the use of HMM was defined as extracting view-invariant features and the use of the homography of hand motions instead was proposed (Wang et al., 2006). In this method, each sign is represented by a template sequence. It is then broken into small units of 3 consecutive frames, such that every unit represents a tiny hand motion. The system is tested by comparing every three frames of video with template units. This method proved to be efficient and performed well. Furthermore, it did not require view-invariant features nor alignment.

Recently, the use of convolutional neural networks has been popularized in the domain of sign language classification. For example, a cascaded model including Single Shot Detector (SSD), Convolutional Neural Network (CNN), and Long Short Term Memory (LSTM) was proposed (Rastgoo et al., 2020). The dataset included 100 000 videos of 100 Persian signs. About 96% accuracy was achieved using only SSD, pre-trained ResNet-50 model, and LSTM. The best system achieved over 98% and included additional feature extraction methods. Furthermore, the use of three-dimensional convolutional neural networks, cascaded for different viewpoints, was proposed (Sharma & Kumar, 2021). It scored 96% precision on the ASL-LVD dataset, which consists of about 3300 ASL words, recorded with four synchronized cameras showing a side view, a head close-up, a half-speed high resolution front view, and a full resolution front view (Neidle et al., 2012).

The 3D-CNN-based methods achieve high accuracies in sign language classification problems. On the other hand, they are often slow and not suitable for use in real-time, especially on mobile devices. Recently, five resource-efficient 3D convolutional neural networks were introduced: 3D-SqueezeNet, 3D-MobileNetV1, 3D-MobileNetV2, 3D-ShuffleNetV1 and 3D-ShuffleNetV2 (Kopuklu et al., 2019). The models are based on popular, lightweight 2D-CNNs often used in mobile applications and inflated to 3D-CNNs versions.

1.2 Outline

This research will focus on investigating the use of 3D-MobileNetV2 for the American Sign Language classification. The MobileNetV2 model was built and inflated to 3D-CNN as described in Kopuklu et al., 2019. It was trained and tested on the Word-Level American Sign Language Dataset (WLASL), which is the largest available video dataset for ASL. The videos had to be first processed in terms of rescaling and extracting frames. Furthermore, data augmentation was performed to ensure more diversity. This is covered in section 2.1 and 2.2. Section 2.3 describes the architecture of the 3D-MobileNetV2. The model is trained on 32 consecutive video frames of size 112×112px for over 400 epochs. Then, it is evaluated using metrics such as accuracy, precision, recall, and F1-score. Furthermore, the top-1 and top-5 accuracy are compared between this model and other approaches on the same dataset: Pose-GRU, Pose-TGCN, VGG-GRU, and I3D.

2 Method

This section begins by introducing a dataset. It is followed by a brief description of the other approaches on which it was tested, and the accuracies achieved by them are presented. The next subsection covers preprocessing and data augmentation. Then, the architecture of 3D-MobileNetV2 is described in detail, and the most important building blocks and layers are covered. In the last two subsections, the training setup and the metrics used to evaluate the model are given.

2.1 Dataset

Li et al., 2020 introduced a Word-Level American Sign Language Dataset (WLASL). It is one of the largest publicly available datasets, containing 21 083 videos of 2 000 signs performed by 119 signers. The length of the videos varies between 0.26 and 8.12 seconds, with an average length of 2.41 seconds. Moreover, the width and height vary between videos. All the data is collected from educational websites and YouTube tutorials about ASL. Due to the limited resources available, as well as to shorten training time, only a subset of the WLASL dataset was used. The chosen subset is

called WLASL100 and contains over 2 000 videos belonging to 100 classes and performed by 97 signers. The dataset is unbalanced, as the number of samples per class varies between as low as 18 and as high as 40.
Apart from the videos, the JSON file is provided by the authors. It contains information such as: video id, split (training or validation), label, start frame, and end frame. Furthermore, the Li et al., 2020 tested the dataset on four different methods: Pose-GRU, Pose-TGCN, VGG-GRU, and I3D. Those approaches are briefly described below. Furthermore, Table 2.1 shows the top-1 and top-5 accuracy achieved by each of the models on the WLASL100 subset.

VGG-GRU uses a 2D-CNN model called VGG16, pre-trained on ImageNet to extract spatial features. Those features are then fed to a stacked Gated Recurrent Unit (GRU) network.

Pose-GRU first extracts 13 upper-body keypoints and 21 joints for each hand. Then, 2D coordinates are concatenated and fed to a stacked GRU. GRU's role is to model the movement's temporal sequential information. The main issue with this approach is that, for this reason, it may not be able to fully capture the spatial relationship between key points. Pose-TGCN was proposed to tackle this issue.

Pose-TGCN similarly to Pose-GRU, utilizes the body and hand keypoints. However, a novel temporal graph convolutional network was proposed. This network was designed to capture both temporal and spatial features. It shows a significant improvement over Pose-GRU, achieving 55.43% accuracy, as opposed to only 46.51% scored by Pose-GRU.

I3D is the Inception network with the filters inflated to 3D. It is pre-trained on both ImageNet and Kinetics-400. Furthermore, the I3D network uses a two-stream configuration, meaning that two models are trained: one on RGB frames and the other one on optical flow. The prediction is averaged over both of them at test time (Carreira & Zisserman, 2017).

Table 2.1: Top-1 and top-5 accuracy achieved by each model on the WLASL-100 subset

	top-1	top-5
Pose-GRU	46.51	76.74
Pose-TGCN	55.43	78.68
VGG-GRU	25.97	55.04
I3D	65.89	84.11

Due to the fact that a significant portion of the videos are not available anymore and only part of the original dataset could be downloaded using the code provided by the authors, the dataset used was downloaded from Kaggle instead (Gazquez, 2022). It included all the videos from the original WLASL dataset. The only difference between this and the original dataset was that the videos were already resized, which will be further described in section 2.2.

2.2 Data preprocessing and augmentation

All of the videos in the dataset have been resized to 256px at their longest dimension. Then, it was padded with the pixel value of 0 in the other dimension to form a square video of size 256×256px. It is different from the implementation of Li et al., 2020. The original rescaling was performed as follows: if either the width or the height of the frame is lower than 226px, the frames are scaled by the factor s as in the equation 2.1, where w is the width and h is the height of the frame. When the width or height exceeds 256px, both dimensions are resized to 256px.

$$s = (1 + \frac{226 - \min(w, h)}{\min(w, h)}) \tag{2.1}$$

The custom PyTorch data loader reads the information from the JSON file mentioned in section 2.1 and converts it to a dataset. Therefore, the dataset contains information such as video ID, label, start frame, and the number of frames. The actual videos are not loaded beforehand. Instead, the IDs saved in the dataset correspond to the filenames, and thus videos are only loaded when they are needed.This approach saves memory and allows for more variation in a dataset due to the fact that each time

a different portion of the video is loaded. This is done by randomly choosing a value equal to or bigger than the start frame, but lower than the end frame, minus 32. This value will be the starting point of the video for the given step. For example, let's take the video, which has 73 frames. The start frame is 0 and the end frame is 72. The starting frame for the given step of the training is a randomly chosen value between 0 and $72 - 32 = 40$. Let's say this value is 25. Then, the subsequent 32 frames, starting from frame number 25, are used during this step of the training. This is only relevant for videos consisting of more than 32 frames and it assures that the model will be trained on different portions of the video.
The next step is extracting the frames from the video, starting from the frame chosen before. There are 32 frames to be extracted. If the video is shorter than that, the remaining frames will be padded with repeated frames, either the first or last one, chosen with equal probability.
Subsequently, data augmentation is performed. This is different for training and validation datasets. During training, a random patch of 112×112px is cropped first from the video, in the same location for every frame. Then, the brightness is adjusted by randomly choosing a value between -111 and 111. Similarly, the contrast value is adjusted to be between -31 and 31. Those ranges were chosen by experimenting with the videos present in the dataset. The same brightness and contrast values are used across all frames in a video. The last step in data augmentation is a horizontal flip with a probability of 0.5. This is necessary as left-handed people usually perform the non-symmetric signs in a mirrored way. During the validation phase, only the center crop is performed; that is, the patch of size 112×112px is cropped from the center of the video. After the data augmentation, for each frame of the video, the values of all pixels are normalized to be between 0 and 1.

2.3 Model

Kopuklu et al., 2019 introduced five resource-efficient 3D-CNN models: 3D-MobileNetV1, 3D-MobileNetV2, 3D-ShuffleNetV1, 3D-ShuffleNetV2 and 3D-Squeezenet. All of those models are based on popular architectures for image classification. Those architectures are fast and lightweight, which was the main reason for choosing one of them for this research. Following a comparison of those five networks, the 3D-MobileNetV2 was selected for several reasons. First of all, its 2D version was specifically tailored for use on mobile devices. Moreover, it achieved 94.59% accuracy for one of the largest available hand-gesture datasets called Jester (Sandler et al., 2018). This result is comparable to that of the ResNetXT-101 model (Sandler et al., 2018), which has over 15 times more parameters. Furthermore, 3D-MobileNetV2 outperformed other resource-efficient 3D-CNN models with a width multiplier of 1, which is a scaling parameter further described in section 2.3.5. On the other hand, resource-efficient 3D-CNN architectures tend to achieve low accuracy on Kinetics-600 (Sandler et al., 2018), with 56.84% achieved by 3D-ShuffleNetV1 with a width multiplier of 2, being the best result, and 3D-MobileNetV2 scoring only 50.65% . The I3D network used as one of the methods of comparison for the WLASL dataset achieved 71.70% accuracy on Kinetics-600 (Carreira et al., 2018). This may suggest that MobileNetV2 will not be able to outperform it, but should achieve higher accuracy than other efficient 3D-CNNs would.
The architecture of this model is based on a 2D-CNN called MobileNetV2 introduced by Sandler et al., 2018. Due to the use of videos as an input, the model was modified by using the 3D kernels, as described in Kopuklu et al., 2019 to allow the network to extract spatio-temporal features within video frames. The code was written in Python and the PyTorch framework was used. It is based on an existing, official Torchvision implementation of MobileNetV2 and the implementation of 3D-MobileNetV2 by Kopuklu et al., 2019.
In this subsection, I will give the reasoning behind choosing this architecture, describe the key building blocks of MobileNetV2 and provide details of its architecture.

2.3.1 Depthwise separable convolutions

The main layers of MobileNetV2 are depthwise separable convolutions. They have been used in MobileNet since its first version, introduced in Howard et al., 2017, due to the drastic reduction in the model size and computations needed. This

is achieved by replacing standard convolution with two separate layers: depthwise and pointwise convolutions. The main difference is that, in contrast to standard convolution, which is applied across all channels at each step, depthwise separable convolution is applied along one channel at a time and then combined using pointwise convolution. The depthwise layer applies a single kernel of size 3 to each input channel, leading to the number of input channels being equal to the number of output channels. The pointwise convolution is a regular convolution layer with a kernel of size 1. For example, if we take the input image of volume $10 \times 10 \times 3$ (width $\times$ height $\times$ number of channels) and apply a standard convolution with a 33 kernel, the output volume will be $8 \times 8 \times 1$. In depthwise separable convolution, the depthwise layer first applies 33 on the same image and outputs $8 \times 8 \times 3$ volume. Then, the pointwise layer combines information across all channels and outputs the result of $8 \times 8 \times 1$ volume. For both layers, batch normalization and the rectified linear activation function are applied. The computational cost of standard convolutions is as presented in equation 2.2, where C_S is the computational cost, M is the number of input channels, N is the number of output channels, D_K is the kernel size and D_F is the size of the feature map.

$$C_S = D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F \tag{2.2}$$

The depthwise separable convolution's cost C_D is the sum of the costs of the depthwise and pointwise convolutions, and it can be expressed as in equation 2.3.

$$C_D = D_K \cdot D_K \cdot M \cdot D_F \cdot D_F + M \cdot N \cdot D_F \cdot D_F \tag{2.3}$$

The cost reduction R of using the depthwise separable convolutions instead of standard convolutions can be calculated by dividing equation 2.3 by equation 2.2, as calculated in equation 2.4.

$$\begin{aligned} R &= \frac{D_K \cdot D_K \cdot M \cdot D_F \cdot D_F + M \cdot N \cdot D_F \cdot D_F}{D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F} \\ &= \frac{1}{N} + \frac{1}{D_K^2} \end{aligned} \tag{2.4}$$

Since in 3D-MobileNetV2, 3D kernels are used, the cost reduction is $\frac{1}{N} + \frac{1}{D_K^3}$ as calculated in equation 2.5.

$$\begin{aligned} R &= \frac{D_K^3 \cdot M \cdot D_F^3 + M \cdot N \cdot D_F^3}{D_K^3 \cdot M \cdot N \cdot D_F^3} \\ &= \frac{1}{N} + \frac{1}{D_K^3} \end{aligned} \tag{2.5}$$

The MobileNetV2, compared to its previous version, introduced two new elements: bottlenecks between layers and shortcut connections between those bottlenecks.

2.3.2 Bottleneck residuals

Residual bottleneck blocks were introduced by He et al., 2016 as a part of the ResNet architecture. The main idea behind them is to make the number of parameters and matrix multiplications lower by stacking three convolution layers. The first and third layers have a filter size of 1. They are responsible for reducing the dimensions and restoring them afterward. The second layer has a filter size of 3, but its input and output dimensions are lower than they normally would be due to the use of the other two layers. The Rectified Linear Unit (ReLu) activation is used after every layer. Furthermore, the ratio between the size of the input bottleneck and its inner size is called the expansion ratio.

2.3.3 Linear bottlenecks

The linear bottlenecks are very similar to the residual bottlenecks. However, it was found that ReLu activation discards values lower than 0, which may result in hurting the performance by destroying too much information. Therefore, in the linear bottlenecks used in MobileNetV2, ReLu is used after the first two layers only. There is no activation function after the last convolution layer in a bottleneck block.

2.3.4 Inverted residuals

This block is the modified version of the linear bottleneck and, as such, it is also built from three stacked convolution layers. As mentioned in previous sections, in a linear block, the dimensions are first reduced, then expanded, and then reduced again. Inverted residuals are called so because those three operations are inverted. The dimensions are

first expanded using the convolution layer with a filter of size 1. This layer is followed by a depth-wise convolution with filter size 3. The third layer, with filter size 1, is responsible for reducing the number of channels. Then, the input and output can be added. Inverted residual blocks yield slightly better results and allow for more memory-efficient implementation than non-inverted versions. This is due to the fact that the amount of memory required is dominated by the size of the smaller bottleneck tensors instead of larger tensors internal to the bottleneck (Sandler et al., 2018).

2.3.5 Width multiplier

The width multiplier is a parameter α allowing the uniform scaling of the MobileNet at each layer. The width multiplier's default value is $alpha = 1$. Changing the value of the parameter to $\alpha \in (0, 1)$, typically 0.25, 0.5, or 0.75, allows the model to be even smaller and faster by reducing the number of parameters by roughly α^2. For multipliers other than 1, it is applied to all the convolution layers except for the last one. On the other hand, the width multiplier may be used to increase the size of the network.

2.3.6 Architecture

The architecture of 3D-MobileNetV2 is shown in Table 2.2. The first layer is 3D convolution with 32 output channels and a stride of $1 \times 2 \times 2$. It is followed by 17 inverted residual blocks with either stride 1 or stride 2, meaning spatiotemporal $2\times$ downsampling (see Figure 2.1). Each of the inverted residual bottleneck blocks, except for the first one, has an expansion factor $t = 6$. Furthermore, every block is repeated up to four times, denoted by n in the table. The network architecture concludes with convolution with a filter size of one and a stride size of one, followed by average pooling and a linear layer with an output size equal to the number of classes k.

2.4 Training

The model accepts a series of 32 frames, each of size 112×112px. The batch size was 8. 70% of the dataset was used for training, 15% for validation, and 15% for testing. The model was trained for

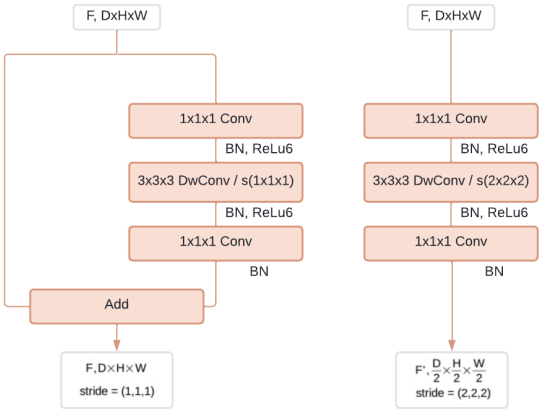

Figure 2.1: Difference between 3D-MobileNetV2 block with stride 1 (left) and stride 2, meaning spatiotemporal $2\times$ downsampling (right). Based on Kopuklu et al., 2019.

Table 2.2: The architecture of 3D-MobileNetV2 model with expansion factor (t), number of output channels (c), number of times the block is repeated (n) and stride (s). k denotes the number of classes. Based on Kopuklu et al., 2019 and Sandler et al., 2018.

Layer	t	c	n	s
Conv3D	-	32	1	$1 \times 2 \times 2$
Bottleneck	1	16	1	$1 \times 1 \times 1$
Bottleneck	6	24	2	$2 \times 2 \times 2$
Bottleneck	6	32	3	$2 \times 2 \times 2$
Bottleneck	6	64	4	$2 \times 2 \times 2$
Bottleneck	6	96	3	$1 \times 1 \times 1$
Bottleneck	6	160	3	$2 \times 2 \times 2$
Bottleneck	6	320	1	$1 \times 1 \times 1$
Conv3D	-	1280	1	$1 \times 1 \times 1$
AvgPool	-	-	1	-
Linear	-	k	-	-

about 400 epochs.
At each step, the cross-entropy loss was calculated between the input and the target. Furthermore, the AdamW optimizer was used. It is a version of the Adam optimizer that implements decoupled weight decay (Loshchilov & Hutter, 2017). The weight decay was found experimentally, and 0.001 was found to work the best. All other parameters were left at their default values. Furthermore, I found the learning rate of 0.0001 to work the best for this model. Moreover, the ReduceLROnPlateau learning rate scheduler was used. If there was no change in loss observed throughout 10 epochs, it decreased the learning rate by a factor of 0.1.
For training of the model, the Peregrine HPC cluster of the University of Groningen was used. Specifically, the node with 6 cores @ 2.7 GHz (12 cores with hyperthreading), 128 GB of memory, and an Nvidia V100 GPU accelerator card were used. Training with this setup took about 7 hours. This was as expected, especially after changes in the number of frames (32) and video size (112×112px). The training time in the initial trials, where those values matched the setup used by Li et al., 2020 (64 frames of size 224×224px) was about 18 hours per 100 epochs.

2.5 Evaluation

To evaluate the model, the test set was used. The main metrics used for the evaluation and further comparison to other models are top-1 and top-5 accuracy. Top-1 accuracy measures the percentage of correct predictions over all predictions, as in the equation 2.6. The top-5 accuracy is similar, but the prediction is classified as correct when the correct label is among the best 5 predictions.

$$accuracy = \frac{correct\_classifications}{all\_classifications} \cdot 100\% \quad (2.6)$$

Due to the unbalanced dataset, three additional metrics will be used for the evaluation of the model. These are: precision, recall, and F1 score. A macro-average is used for all of those metrics, as it is sensitive to data imbalance. It is computed as the arithmetic mean of all per-class scores for the given metrics.
Precision measures true positives, that is, the number of correct positive predictions. For each class, it is calculated as in equation 2.7, where TP means true positive and FP means false positives. Then, the average of all classes is taken.

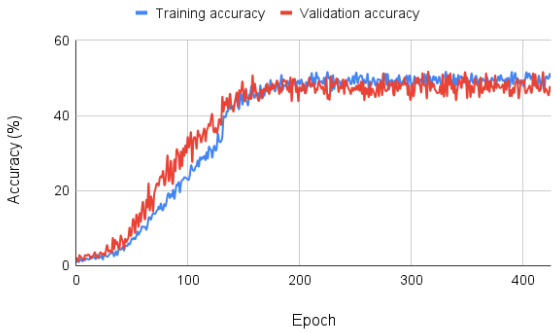

Figure 3.1: Training and validation accuracy of MobileNetV2 trained on WLASL100 dataset.

$$precision = \frac{TP}{TP + FP} \quad (2.7)$$

The recall is the number of correct positive predictions over all positive labels in the data. Similarly to the precision, it is calculated for each class separately and then averaged. Equation 2.8 shows how the recall is calculated for each class.

$$recall = \frac{TP}{TP + FN} \quad (2.8)$$

The F1 score is a harmonic mean of precision and recall. It helps to balance those two metrics. In this case, it is calculated as the average of the F1 scores for each class. Equation 2.9 shows how the F1 score is calculated for each class.

$$F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall} \quad (2.9)$$

Furthermore, the cross-entropy loss for both the training and validation for each epoch will be plotted.

3 Results

The results of the 3D-MobileNetV2 for WLASL100 are presented in this section. First, the training and validation accuracy for all epochs is plotted (see Figure 3.1). Similarly, the loss was also plotted for both the training and validation for each epoch

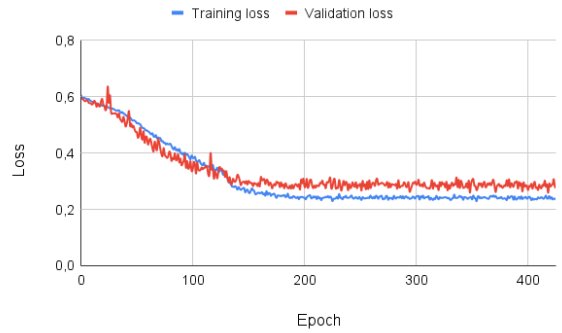

Figure 3.2: Training and validation loss of MobileNetV2 trained on WLASL100 dataset.

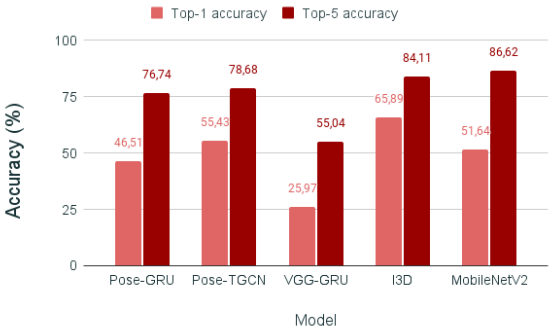

Figure 3.3: The comparison of top-1 and top-5 performance of Pose-GRU, Pose-TGCN, VGG-GRU, I3D and MobileNetV2 on WLASL100 dataset.

(see Figure 3.2).

The presented graphs show very little change in both accuracies during the first 30 epochs. Furthermore, we can clearly see that the model tends to underfit during about 130 first epochs. This happens when the model struggles to fit the training data. In this case, it is most likely caused by the amount of data augmentation during the training, which includes mainly random crop, horizontal flip with 0.5 probability, and randomly adjusted brightness and contrast. In contrast, for the validation step, only a center crop is performed. This causes the training data to be significantly harder to predict than the validation data. Over time, this difference becomes less significant, and both the accuracies and losses converge. The training loss then varies between 0.23 and 0.25 for the rest of the training, and the validation loss varies between 0.26 and 0.31.

The testing set was used to determine all the other metrics for the model. The accuracy on the testing set was 51.64%. Therefore, the MobileNetV2 performed worse than the I3D model trained on the same dataset by Li et al., 2020. However, it is worth noting that the I3D model was pre-trained on both ImageNet and Kinetics. Furthermore, it is significantly larger than MobileNetV2. It also employs the two-stream configuration described in Section 2.1. The action classification model may significantly benefit from such an approach due to the fact that optical flow captures the motion.

The model performed slightly worse than Pose-TGCN, based on extracting body and hand keypoints and the TGCN network. This model achieved a top-1 accuracy of 55.43%. On the other hand, the Pose-GRU is based on a similar idea but uses GRU instead of TGCN. This model was outperformed by 3D-MobileNetV2, as it scored 46.51%. The worst performance was demonstrated by the VGG-GRU, which is a 2D-CNN network VGG16 combined with GRU. Only 25.94% accuracy was achieved by this model.

Surprisingly, the MobileNetV2 outperformed all other models in terms of top-5 accuracy, achieving 86.62%. This may suggest that the model is overall learning well, but often faces issues when signs are similar to each other. It probably fails to recognize the difference between some of the features of sign language.

Table 3.1 shows the values of the remaining metrics. All of them have rather low values. The precision was calculated to be 54.02%. Therefore, only 54.02% of the videos classified actually belonged to the class they were predicted to belong in. Furthermore, the recall was 43.10%. This is the percentage of the videos classified correctly over all of the examples for the class they belong to. The F1 score, being the harmonic mean of the two, is only 47.94%.

4 Discussion

Overall, MobileNetV2 outperformed two popular approaches: 2D-CNN combined with RNN and a pose-based approach with RNN. Both the

Table 3.1: F1 score, precision and recall for MobileNetV2 model trained on WLASL100 dataset.

Metrics	Result
precision	54.02%
recall	43.10%
F1 score	47.94%

Pose-TGCN and I3D models are better than MobileNetV2. On the other hand, all of the methods introduced for the WLASL database are performing worse in terms of top-5 accuracy. Therefore, as for now, the Convolutional Neural Networks with 3D kernels yield the best results for the task of sign language recognition based on video input.

Figures 3.1 and 3.2 showed that the validation accuracy was higher than the training accuracy and the validation loss was lower than the training loss during the first stages of the training. However, the values converge, and eventually the training accuracy exceeds the validation accuracy, and the training loss becomes lower than the validation loss. All the values begin to stabilize after about 150 epochs. The underfitting during the first phase of the training is most likely caused by a heavy difference in the data augmentation. For the training dataset, significantly more data augmentation is performed: random crop, random horizontal flip, and randomly adjusted brightness and contrast. With the validation set only being cropped in the center, it is undoubtedly much easier to fit, causing the model to underfit in the first phase. On the other hand, a lot of data augmentation ensures that the training data is more diverse and therefore allows the model to better learn the relevant features.

The 3D-MobileNetV2 achieved lower testing accuracy than the I3D and Pose-TGCN. Unfortunately, none of those are directly comparable in terms of architecture. Pose-TGCN utilizes a completely different method, based on extracting the body and hand keypoints. I3D is the only architecture different from 3D-MobileNetV2, which is based on CNN with 3D filters. On the other hand, it utilizes a two-stream configuration, where two I3D networks are trained. One accepts RGB frames as input, while the other uses optical flow. The predictions are averaged between the two networks. Optical flow is the pattern of an apparent motion between two consecutive frames. Thus, using such a configuration, the model's performance can be improved due to capturing motion better than when using RGB frames only. Another notable difference is that the I3D was pre-trained on both the ImageNet and the Kinetics-400. Firstly, the 2D-CNN Inception was trained on ImageNet. Then, the 3D filters were bootstrapped from the 2D filters, and the network itself was inflated to use 3D filters. It was then fine-tuned on Kinetics-400. The authors of the WLASL dataset used this network to fine-tune it on their own, sign language dataset. There are few resources on 3D-MobileNetV2 because it is still relatively new and not as widely used as I3D. Therefore, no such method has been used on it yet. This may be an idea for future research. It would be interesting to test how much the pretraining on a classic MobileNetV2 based on 2D kernels can influence the performance of the 3D-MobileNetV2 of that network. Another idea for future research would be to check how well 3D-MobileNetV2 deals with the optical flow and whether it would benefit from using two streams.

Furthermore, the 3D-MobileNetV2 proved to achieve high top-5 accuracy, outperforming all of the other models. This may suggest that it fails to capture small differences between the signs and often confuses those that are very similar to each other. This may be due to another difference between the used model and the methods utilized by the authors of the WLASL dataset: the input size. They used 224×224px frames for training. In my case, the videos were already resized to 256×256px. The portion cropped from them and input to the model was only 112×112px. Due to the limited resources available, it was necessary to reduce the training time significantly. On the other hand, this could lead to some details being lost. Furthermore, due to the size of the data being only 256×256px, cropping the random 224×224px patch from it would reduce the number of different data points for training when compared to the smaller size of the patch. Moreover, only 32 frames were used for the training, compared to 64 in the original study. On the one hand, it reduces

the training time due to the smaller input size. Furthermore, it provides more variability in the dataset due to the fact that at each step, different, randomly chosen portions of videos will be used. On the other hand, it can hurt the performance, as often only part of the sign is visible. This could be another reason for the confusion caused by similar signs. For example, if there is part of the motion for different signs that is almost the same, it is possible that the model classified it as one of them, but in fact, it should be the other one. Thus, the top-1 accuracy is low, but the correct prediction was among the top-5, hence the high top-5 accuracy. With more resources and time available, it may be interesting to see how the I3D model would compare to 3D-MobileNetV2 when both are trained with the same training configurations.

A further issue with the American Sign Language classification lies in the data itself. Even though the Word-Level American Sign Language dataset is one of the largest ones available, it is still very limited. For most of the classes in the WLASL100 subset, there are only 20 or even fewer samples. In the whole dataset, there are often fewer than 10 samples per class. To compare, Kinetics-400 has as many as 400 videos per class (Kay et al., 2017) and its newest version, Kinetics-700, has at least 600 samples per class (Carreira et al., 2019). Furthermore, the Jester (hand gesture dataset), on which 3D-MobileNetV2 achieved an accuracy of 94.59%, consists of 148 092 videos belonging to 27 classes (Materzynska et al., 2019). Moreover, the WLASL dataset sometimes contains videos of the compound signs. For example, in the 'book' class, there are signs for 'geography book', 'history book', 'law book', 'math book', etc. While undeniably all of those means 'book', each of those videos in fact contains two signs: 'geography' and 'book'; 'history' and 'book'; 'law' and 'book', 'math' and 'book'. Thus, it would be more useful to divide those videos into two, such that in a 'book' class there are only signs meaning exactly the 'book'. Those compound signs may often confuse the classifier, as when the video is cropped, it may only contain one of the signs (so, for example, 'chemistry' from a 'book' class, while the full video would mean 'chemistry book'. Moreover, there are sometimes a few different signs with the same meaning. This is also an issue, especially when considering an already limited number of samples. All of the issues mentioned above make the training of the classifier an even harder task. There is a need for a bigger and cleaner dataset of American sign language or a model able to learn to generalize sufficiently well after training on such limited resources.

5 Conclusion

Recently, convolutional neural networks with 3D kernels have been gaining more and more interest from the research community. For example, the new efficient network called X3D has been introduced (Feichtenhofer, 2020). It is also based on a 2D-CNN network but expands it in different dimensions rather than just inflating the kernels. The basic set of expansion operations includes X-Fast expanding the temporal activation size; X-Temporal expanding the temporal size; X-Spatial expanding the spatial resolution; X-Depth expanding network depth; X-Width expanding the channel number; X-Bottleneck expanding the inner channel width. The smallest version of the network has only 3.8 M parameters and achieved 76% accuracy on the Kinetics-400 dataset. It is a great result, especially when compared to state-of-the-art I3D pre-trained on ImageNet, achieving only 71.1%. It may be worthwhile to investigate the X3D network further, and it will be interesting to see how it performs in such a difficult task as sign language classification.

To conclude, the 3D-MobileNetV2 requires further testing to determine whether it is suitable for sign language classification. It outperformed the significantly larger I3D architecture in terms of top-5 accuracy, while it was only slightly worse in terms of top-1 accuracy. This, together with the great results on the Jester dataset, may suggest that 3D-MobileNetV2 is able to capture the spatio-temporal features of hand gestures sufficiently well. It would be interesting to see how it performs when trained on a bigger and cleaner dataset.

The area of CNN with 3D kernels to be used for video classification tasks is still fairly new. There is not much research on it. Furthermore, most of the current networks are built by inflating the

kernels of well-known 2D-CNN architectures, such as Inception or MobileNet. As for now, there are not many alternative approaches, nor architectures tailored specifically for video classification.
Furthermore, the research on sign language classification in the past has been largely focused on learning from images. This is not a suitable approach for something as complex as sign language, where motion is very important. Furthermore, the video-based approaches are often tested on small datasets or only for a limited vocabulary. This is due to the large dataset not being available. The other reason is that 3D-CNN, even in its efficient versions such as 3D-MobileNetV2, is expensive to train.

The code for this project is available on GitHub: `github.com/JuliaWalczynska/3D-MobileNetV2-ASL`.

References

Carreira, J., Noland, E., Banki-Horvath, A., Hillier, C., & Zisserman, A. (2018). A short note about kinetics-600. *arXiv preprint arXiv:1808.01340*.

Carreira, J., Noland, E., Hillier, C., & Zisserman, A. (2019). A short note on the kinetics-700 human action dataset. *arXiv preprint arXiv:1907.06987*.

Carreira, J., & Zisserman, A. (2017). Quo vadis, action recognition? a new model and the kinetics dataset. In *proceedings of the ieee conference on computer vision and pattern recognition* (pp. 6299–6308).

Cartwright, B. (2016). *Signs that are close... but not the same - set 1.* `https://www.signingsavvy.com/article/173/Signs+That+Are+Close...+But+Not+the+Same+-+Set+1`.

Feichtenhofer, C. (2020). X3d: Expanding architectures for efficient video recognition. In *Proceedings of the ieee/cvf conference on computer vision and pattern recognition* (pp. 203–213).

Fels, S. S., & Hinton, G. E. (1993). Glove-talk: A neural network interface between a data-glove and a speech synthesizer. *IEEE transactions on Neural Networks*, *4*(1), 2–8.

Gazquez. (2022). *Wlasl (world level american sign language) videos.* `https://www.kaggle.com/datasets/gazquez/wlasl-processed`. (Version 2)

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 770–778).

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., ... Adam, H. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*.

Kadous, M. W., et al. (1996). Machine recognition of auslan signs using powergloves: Towards large-lexicon recognition of sign language. In *Proceedings of the workshop on the integration of gesture in language and speech* (Vol. 165, pp. 165–174).

Kasukurthi, N., Rokad, B., Bidani, S., Dennisan, D., et al. (2019). American sign language alphabet recognition using deep learning. *arXiv preprint arXiv:1905.05487*.

Kay, W., Carreira, J., Simonyan, K., Zhang, B., Hillier, C., Vijayanarasimhan, S., ... others (2017). The kinetics human action video dataset. *arXiv preprint arXiv:1705.06950*.

Kopuklu, O., Kose, N., Gunduz, A., & Rigoll, G. (2019). Resource efficient 3d convolutional neural networks. In *Proceedings of the ieee/cvf international conference on computer vision workshops* (pp. 0–0).

Li, D., Rodriguez, C., Yu, X., & Li, H. (2020). 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* (pp. 1459–1469).

Liang, R.-H., & Ouhyoung, M. (1998). A real-time continuous gesture recognition system for sign language. In *Proceedings third ieee international conference on automatic face and gesture recognition* (pp. 558–567).

Loshchilov, I., & Hutter, F. (2017). Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*.

Materzynska, J., Berger, G., Bax, I., & Memisevic, R. (2019). The jester dataset: A large-scale video dataset of human gestures. In *Proceedings of the ieee/cvf international conference on computer vision workshops* (pp. 0–0).

Munib, Q., Habeeb, M., Takruri, B., & Al-Malik, H. A. (2007). American sign language (asl) recognition based on hough transform and neural networks. *Expert systems with Applications*, *32*(1), 24–37.

Neidle, C., Thangali, A., & Sclaroff, S. (2012). Challenges in development of the american sign language lexicon video dataset (asllvd) corpus. In *5th workshop on the representation and processing of sign languages: interactions between corpus and lexicon, lrec.*

Rastgoo, R., Kiani, K., & Escalera, S. (2020). Video-based isolated hand sign language recognition using a deep cascaded model. *Multimedia Tools and Applications*, *79*(31), 22965–22987.

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L.-C. (2018). Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 4510–4520).

Sharma, S., & Kumar, K. (2021). Asl-3dcnn: American sign language recognition technique using 3-d convolutional neural networks. *Multimedia Tools and Applications*, *80*(17), 26319–26331.

Starner, T., Weaver, J., & Pentland, A. (1998). Real-time american sign language recognition using desk and wearable computer based video. *IEEE Transactions on pattern analysis and machine intelligence*, *20*(12), 1371–1375.

Wang, Q., Chen, X., Wang, C., & Gao, W. (2006). Sign language recognition from homography. In *2006 ieee international conference on multimedia and expo* (pp. 429–432).