TARAS SHEVCHENKO NATIONAL UNIVERSITY OF KYIV
FACULTY OF MECHANICS AND MATHEMATICS
DEPARTMENT OF ALGEBRA

Degree: bachelor

specialization 111 - mathematics
educational program - computer mathematics

"Sign language recognition using deep learning"

3rd year students
Roman Polishchenko
Valentyn Kofanov

Scientific supervisor
Andriy Oliynyk
Professor, Dr Hab,

Scientific consultant
Olga Sinelnikova

Kyiv – 2020

Contents

1 Introduction **3**

2 Analysis of the problem **3**
2.1 Related work 3
2.2 Task setting 4

3 Models and methods **5**
3.1 Convolutional neural networks 5
3.2 Recurrent neural networks 7
3.3 Used technologies 8

4 Experiments **8**
4.1 Implementation Details 10
4.2 Result 11

5 Conclusions **11**

References **13**

Appendices **15**

Abstract

In this work we presented visual-based system for Sign Language Recognition. Our approach uses convolutional neural networks (CNN) to analyze spatial features and recurrent neural networks (RNN) with LSTM cells to analyze temporal features of the video data. Model was trained and evaluated on Argentinean Sign Language dataset and achieved accuracy up to 90% for small part of the dataset (10 signs) and accuracy up to 83% for the whole dataset (64 signs). We also implemented a cross-platform application using Kivy Python framework to demonstrate our results. It can recognize gestures in two modes: from saved video and from camera in real time.

1 Introduction

Gesture recognition task has lots of applications: remote control of computer, smartphone or domestic appliances, interaction with game consoles for better dive into game experience, etc. All of these tasks imply detection and recognition different gestures by photo and video data. And there is one more especially important task, that is slightly different from all the others. It is automated sign language recognition (SLR) and translation.

Modern world is confidently moving towards globalization and simplification all of the barriers between people. Applications for translating text from one language to another have become widely used and irraplacable for majority of the people. They can translate given word, sentence, whole web page or even autonomously recognize and translate the text from a photo. However, there is still an area of human interaction in which the development of translators is not so noticeable, namely – sign language and its translation for people, who do not know it.

Currently, for our best aknowledgement, there is no popular high-quality sign language translation application and live communication is possible only in the presence of a specially trained translator. Similar to oral communication, there is a need in the world for a method or application that can recognize and translate sign language, so that live communication is possible, even if one of the parties does not understand sign language.

In this work we limit the focus on the main principles of deep learning for computer vision and on developing the application for recognition and translation sign language without any additional equipment, but a camera. We consider Argintinean Sign Language dataset with large gesture variants and big amount of raw video samples.

2 Analysis of the problem

2.1 Related work

For the last two decades, the topic of sign language recognition and translation has been actively researched, but external factors such as inhomogeneous image background

or illumination impeded achieving accurate results. Some approaches avoid these obstacles. For example, [3] uses Microsoft Kinect to build a 3D map of hands movements and analyzes this data to recognize gestures. Then, depending on the mode, there is either an independent translation of one gesture, or recognition and translation of an entire sentence, which also includes work with the so-called Sentence Model. The word recognition algorithm is based on a comparison with a set of stored gesture trajectories. The accuracy for the dataset consisting of 239 words of the Chinese sign language (5 records of each word - 1 test, 4 for comparisons) is up to 96.32%. Other approaches use statistical models, such as the hidden Markov model [4] or the Bayes network [5] and various combinations of gethering and processing the data.

On the other hand, in recent decades, deep learning in general and the Computer Vision section in particular have achieved significant results. Therefore, the use of convolutional neural networks (CNN) and recurrent neural networks (RNN) to process photo and video data has become not only possible, but also effective.

For example, [6] demonstrated a successful, though not very accurate, attempt to build a 3D hand model based on a single image. With a larger and better data set this approach could be a powerful basis for many computer vision tasks, including sign language recognition.

Partially, this approach is used in [7]. On the other hand, a combination of "conventional" statistical models and modern image processing technologies is also possible. For example, [8] uses a hybrid of CNN and hidden Markov models (HMM). Finally, [9] uses recurrent neural networks instead of HMM, which takes into account the information from previous inputs for the future predictions.

2.2 Task setting

We set ourselves the following objectives:

- sort out the existing approaches in SLR;
- learn about such neural networks' architectures as convolutional and recurent neural networks (CNN and RNN);
- process the given data and transform it in a way that is applicable for effective training of machine learning models;
- develop architecture for our model and train it to get relevant results;
- create a handy application with our model embedded that will use only a camera to classify gestures.

3 Models and methods

Sign language recognition implies analyzing both spatial and temporal features of the data. In our work we used convolutional neural network to extract spatial features from the frames. After that to learn the temporal dependencies of frames we have used recurrent neural network.

3.1 Convolutional neural networks

Main idea of convolutional neural network [2], why it is so good for image processing, is to perform analysis of the data by small windows to prevent the result be dependent on relative position of the object on the picture. Of course, object position plays an important role, but first it must be recognized in any case, and this recognition is local and independent of the specific position of the area with the object inside the large picture. Therefore, the CNN makes this assumption explicitly: let's cover the input with small windows (say, 5x5 pixels) and extract the features in each such window with a small neural network. This operation is called convolution and it is just a linear transformation and the windows are called kernels. If x^l is l'th feature map of the layer, than the result of two dimensional convolution with kernel size $2d+1$ and weights matrix W of size $(2d+1) \times (2d+1)$ is formally defined as:

$$y^l_{i,j} = \sum_{-d \le a,b \le d} W_{a,b} x^l_{i+a,j+b},$$

where $y^l_{i,j}$ – result of the convolution on the level l, $x^l_{i,j}$ - its input, i.e. the output of the whole previous layer. In other words, to get (i,j)-th component of the next layer we apply scalar product to the pixels in kernel and weights matrix. Consider an example of convolution (Figure 1) with weights matrix W of size 3×3 applied to the matrix X of size 5×5:

$$X * W = \begin{pmatrix} 0 & 1 & 2 & 1 & 0 \\ 4 & 1 & 0 & 1 & 0 \\ 2 & 0 & 1 & 1 & 1 \\ 1 & 2 & 3 & 1 & 0 \\ 0 & 4 & 3 & 2 & 0 \end{pmatrix} * \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 2 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 9 & 5 & 4 \\ 8 & 8 & 10 \\ 2 & 1 & 0 \end{pmatrix}.$$

This transformation has following properties:

- convolution preserves the input structure (the order of the one-dimensional case, the relative position of pixels in two-dimensional, etc.), as it is applied to each section of the input data separately;
- the convolution operation has the sparse property, since the value of each neuron of

the next layer depends only on a small fraction of the input neurons (for example, in a densely connected neural network, each neuron would depend on all neurons of the previous layer);

- convolution repeatedly uses the same weights, since they are re-applied to different sections of the input.

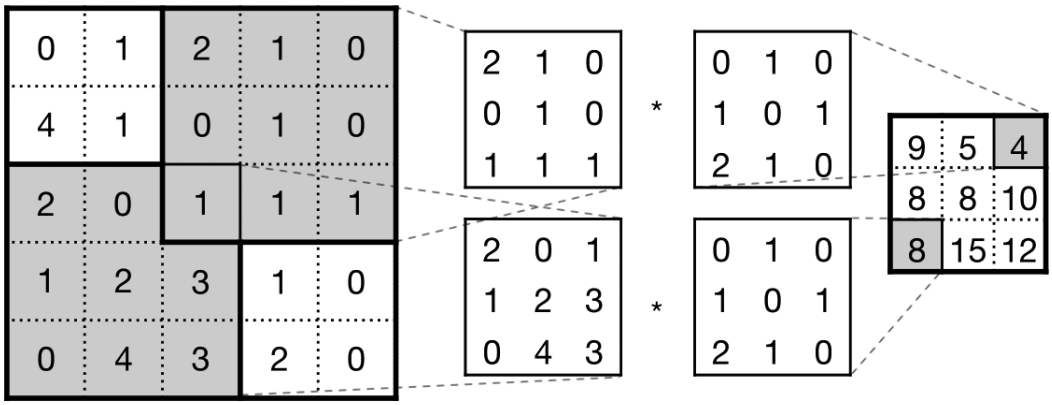


Figure 1: Example of convolution: two submatrices examples and overall result

Almost always, after convolution the non-linear function applied, which we can write as follows:

$$z_{i,j}^{l} = h(y_{i,j}^{l}).$$

ReLU (rectified linear unit) is often used as function h, especially in deep networks, but traditional σ and tanh are also used sometimes:

$$h_{ReLU}(x) = \max(0, x).$$

After the nonlinerity we need to perform a downsampling operation along the spatial dimensions (width, height) to reduce the amount of parameters and computation in the network. For this purpose we use pooling layer. The most common approach used in pooling is max pooling, that can be defined as follows:

$$x_{i,j}^{l+1} = \max_{-d \le a \le d, -d \le b \le d} z_{i+a,j+b}^{l}.$$

Example of subsampling with window of size 2 and step 2:

$$\begin{pmatrix} 0 & 1 & 2 & 1 \\ 4 & 1 & 0 & 1 \\ 2 & 0 & 1 & 1 \\ 1 & 2 & 3 & 1 \end{pmatrix} \xrightarrow{\text{max pooling}} \begin{pmatrix} 4 & 2 \\ 2 & 3 \end{pmatrix}$$

3.2 Recurrent neural networks

Recurrent neural networks analyze temporal features, because they are able to remember things from proir input(s) while predicting the output(s). One drawback of RNN is that, in practice, RNNs are not able to learn long-term dependencies. Therefore, our model uses Long Short-Term Memory (LSTM) [11] network, which is a variation of RNN with LSTM units.

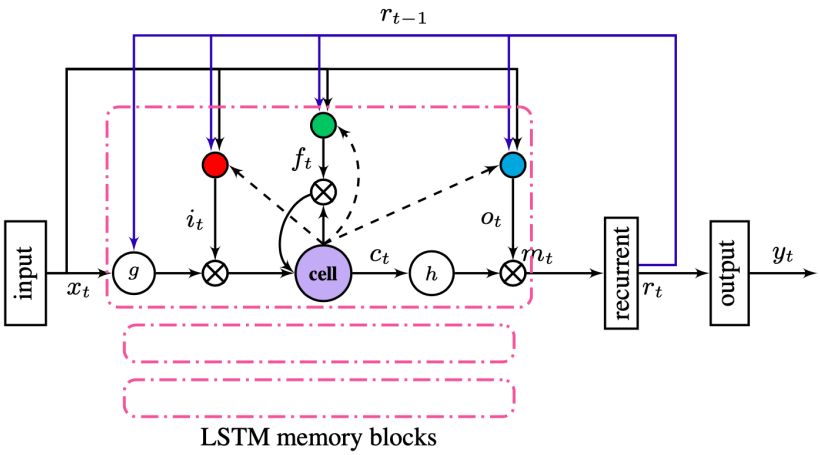


Figure 2: LSTM architecture. A single memory block is shown for clarity.

The LSTM contains special units called memory blocks in the recurrent hidden layer (Figure 2). The memory blocks contain memory cells with self-connections storing the temporal state of the network in addition to special multiplicative units called gates to control the flow of information. The input gate controls the flow of input activations into the memory cell. The output gate controls the output flow of cell activations into the rest of the network. The forget gate scales the internal state of the cell before adding it as input to the cell through the self-recurrent connection of the cell, therefore adaptively forgetting or resetting the cell's memory. In addition, the modern LSTM architecture contains peephole connections from its internal cells to the gates in the same cell to learn precise timing of the outputs.

An LSTM network computes a mapping from an input sequence $x = (x_1, \ldots, x_T)$ to an output sequence $y = (y_1, \ldots, y_T)$ by calculating the network unit activations the following

equations iteratively from $t = 1$ to T:

$$i_t = \sigma(W_{ix}x_t + W_{im}m_{t-1} + W_{ic}c_{t-1} + b_i),$$

$$f_t = \sigma(W_{fx}x_t + W_{fm}m_{t-1} + W_{fc}c_{t-1} + b_f),$$

$$c_t = f_t \odot c_{t-1} + i_t \odot g(W_{cx}x_t + W_{cm}m_{t-1} + b_c),$$

$$o_t = \sigma(W_{ox}x_t + W_{om}m_{t-1} + W_{oc}c_{t-1} + b_o),$$

$$m_t = o_t \odot h(c_t),$$

$$y_t = \phi(W_{ym}m_t + b_y),$$

where the W terms denote weight matrices (e.g. W_{ix} is the matrix of weights from the input gate to the input), W_{ic} , W_{fc} , W_{oc} are diagonal weight matrices for peephole connections, the b terms denote bias vectors (b_i is the input gate bias vector), σ is the logistic sigmoid function, and i, f, o and c are respectively the input gate, forget gate, output gate and cell activation vectors, all of which are the same size as the cell output activation vector m, $\odot$ is the element-wise product of the vectors, g and h are the cell input and cell output activation functions (default, tanh), and ϕ is the network output activation function.

3.3 Used technologies

Main programming language is Python3. Next libraries and frameworks was used:

- OpenCV - for video processing;
- Keras - for creating, training and using deep learning models;
- NumPy - for numerical computations;
- Matplotlib - for visualization.

Kivy framework for Python was used to create an application.

Since training a model requires a lot of computational resources, Google Collab was used as cloud platform for this task, because of its free access to GPU.

4 Experiments

For our work the Argentinean Sign Language (LSA64) [12] dataset was chosen. The database was recorded in two sets. The first one is in an outdoors environment, with natural lightning, while the second – in an indoors environment, with artificial lightning, to provide differences in illumination between signs.

(a) one-handed gesture (b) two-handed gesture

Figure 3: Frames from dataset

To simplify the problem of hand segmentation within an image, subjects wore fluorescent-colored gloves (Figure 3). This substantially simplify the problem of recognizing the position of the hand and performing its segmentation, and remove all issues associated to skin color variations, while fully retaining the difficulty of recognizing the handshape.

Data preproccesing consists of following steps:

1. Obtain exactly 10 frames (uniformly distributed) from each video.
2. Apply masks that remove background and make it black-and-white (Figure 4).
3. Combine all 10 frames into 1 training sample.

Source code for functions are available in Appendix A.

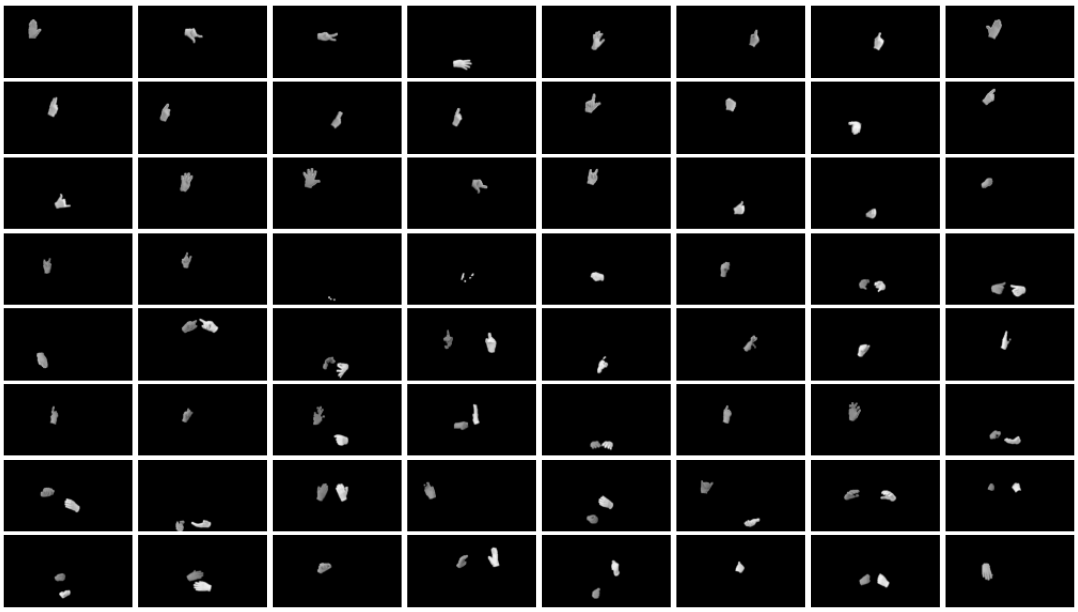


Figure 4: Examples of masked frames for each gesture

4.1 Implementation Details

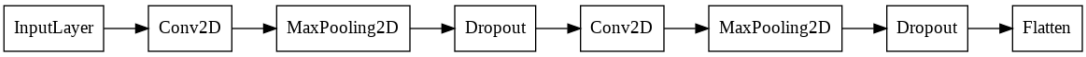


Figure 5: CNN scheme

For spatial features analysis we used convolutional neural network. Overall scheme is shown on Figure 5. This single CNN is applied to each frame (time_stamp) in a sample sequentially. The implementation uses TimeDistributed layer from Keras. The outputs are passing to LSTM layer to analyze temporal dependencies. The LSTM output is passed to two Dense layers to generalize and make prediction (Figure 6). Categorical crossentropy is chosen as loss function and ADAM [10] - as an optimizer. Source code for model is available in Appendix B.

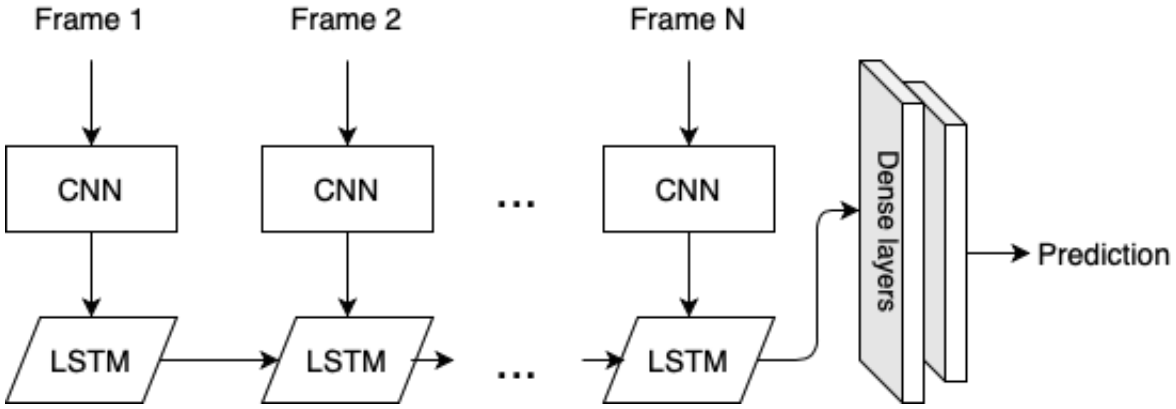


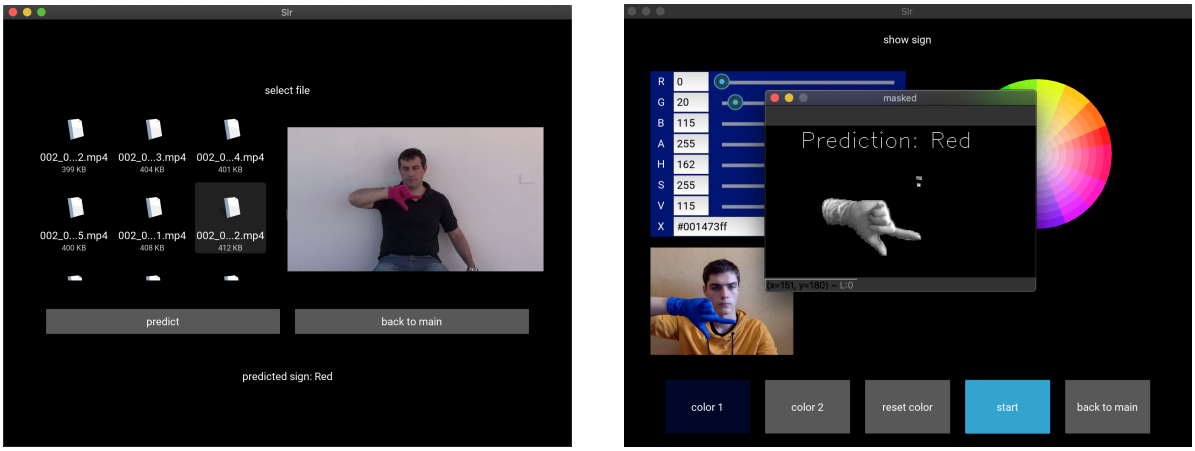
Figure 6: Model overall scheme

To implement the model, we created an application (Figure 7) based on the Kivy Python framework. There are two ways to recognize signs:

- recognition for an already created video file,
- real-time recognition.

For the first way, the user needs to select an existing file and upload it. In this case, a preview of the video is available.

For real-time recognition mode, it is advisable to do a little preparation. You need to set the color(s) of the hand(s) or glove(s) to be able to distinguish it from background. User can set color for one or two hands just by succesive pressing on them on the picture. After that, the real-time recognition begins.



(a) recognition for file

(b) real-time recognition

Figure 7: Application overview

4.2 Result

Two types of tests were made: on small part of data (first 10 signs) and on the whole dataset (64 signs). For the first type, accuracy was up to 90% (Figure 8).

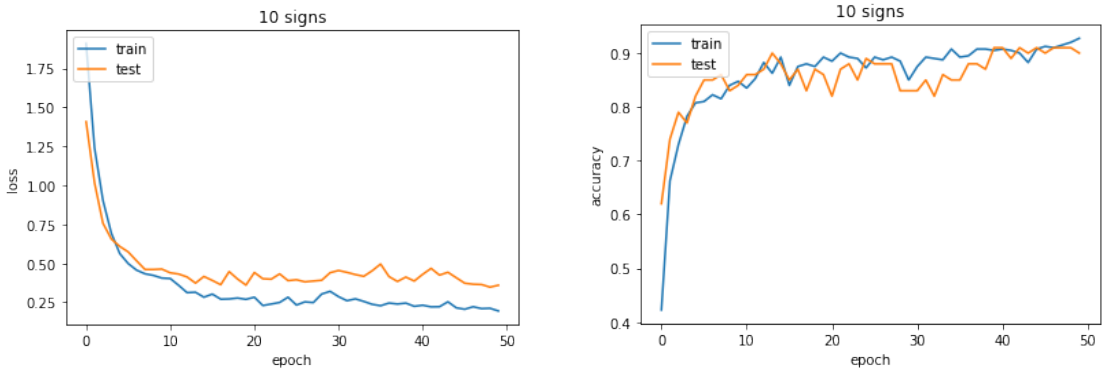


Figure 8: Results on 10 signs

For 64 signs our first model was too small and unable to achieve accuracy higher than 55%. Obviously, it did not have enough generalization possibilities. So we made it "deeper" and it achieved higher results: up to 83% (Figure 9)

5 Conclusions

In this work, we investigated the sign language recognition problem and proposed a visual-based method to solve it. It uses convolutional neural network for analyzing spatial features and recurrent neural networks – for temporal features. We have achieved accuracy up to 90% for small part of the dataset (10 signs) and accuracy up to 83% for the whole

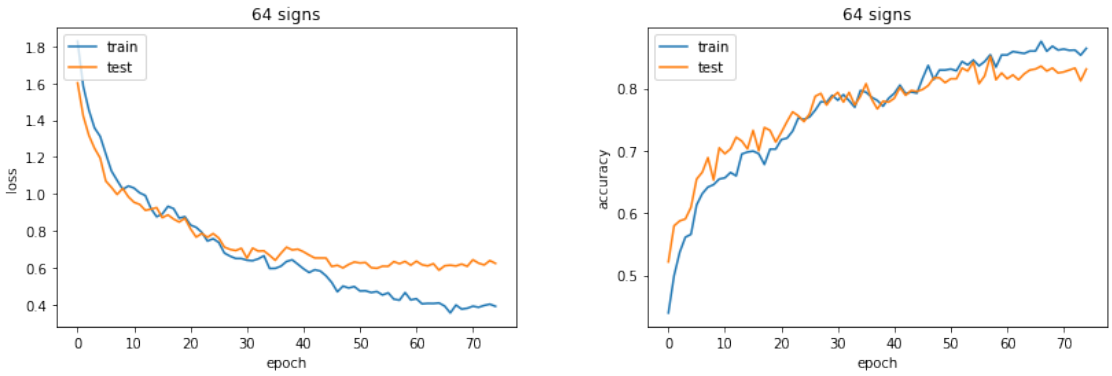


Figure 9: Results on 64 signs

dataset (64 signs). This shows that combination of CNN and RNN can be successfully used to analyze videos in order to classify sigh language gestures.

Our results is not as high as it could be, due to the limitations of computational power and dataset size and variability. In future, we would like to continue our work and create own dataset of Ukrainian Sign Language to implement real-time bilateral translation application.

References

[1] G. Huang, Z. Liu, L. Van Der Maaten and K. Q. Weinberger, "Densely Connected Convolutional Networks" 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, 2017, pp. 2261-2269, doi: 10.1109/CVPR.2017.243.

[2] Николенко С., Кадурин А., Архангельская Е. Глубокое обучение. — СПб.: Питер, 2018. — 480 с.: ил. — (Серия «Библиотека программиста»). ISBN 978-5-496-02536-2

[3] Chai, Xiujuan, Guang Li, Yushun Lin, Zhihao Xu, Yiqing Biocompatibl Tang and Xilin Chen. “Sign Language Recognition and Translation with Kinect.” (2013).

[4] C. Vogler and D. Metaxas. Parallel Midden Markov Models for American Sign Language Recognition. In IEEE Interna- tional Conference on Computer Vision (ICCV), 1999.

[5] A. Thangali, J. P. Nash, S. Sclaroff and C. Neidle, "Exploiting phonological constraints for handshape inference in ASL video," CVPR 2011, Providence, RI, 2011, pp. 521-528, doi: 10.1109/CVPR.2011.5995718.

[6] C. Zimmermann and T. Brox, “Learning to Estimate 3D Hand Pose from Single RGB Images“, ICCV 2017.

[7] Dimitris N Metaxas, Mark Dilsizian, Carol Neidle. 2018. "Linguistically-driven Framework for Computationally Efficient and Scalable Sign Recognition.." Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)

[8] Koller, O, Zargaran, O, Ney, H and Bowden, R (2016) Deep Sign: Hybrid CNN-HMM for Continuous Sign Language Recognition In: The British Machine Vision Conference (BMVC) 2016, 2016-09-19 - 2016-09-22, York.

[9] Masood S., Srivastava A., Thuwal H.C., Ahmad M. (2018) Real-Time Sign Language Gesture (Word) Recognition from Video Sequences Using CNN and RNN. In: Bhateja V., Coello Coello C., Satapathy S., Pattnaik P. (eds) Intelligent Engineering Informatics. Advances in Intelligent Systems and Computing, vol 695. Springer, Singapore

[10] Kingma, D., Ba, J.: Adam: a method for stochastic optimization (2014). arXiv preprint arXiv:1412.6980

[11] Sak, Hasim, Andrew W. Senior, and Françoise Beaufays. "Long short-term memory recurrent neural network architectures for large scale acoustic modeling." (2014).

[12] Ronchetti, F., Quiroga, F., Estrebou, C., Lanzarini, L., Rosete, A.: LSA64: a dataset of Argentinian sign language. In: XX II Congreso Argentino de Ciencias de la Computación (CACIC) (2016)

Appendix A

```
def apply_mask(frame, color, gap=10, lower_s=50, lower_v=50, upper_s=255,
↪  upper_v=255):
    """
    :param: color - must be an integer representing H-value of color in
↪  HSV format
    :return: resulting mask
    """
    # add some gap
    lower_bound = np.array([color - gap, lower_s, lower_v])
    upper_bound = np.array([color + gap, upper_s, upper_v])
    # first num ∈ [0, 180]
    if (lower_bound[0] < 0):
        lower_bound[0] = 0
    if (upper_bound[0] > 180):
        upper_bound[0] = 180
    return cv2.inRange(frame, lower_bound, upper_bound)

def parse_frame(frame, colors):
    """
    :param frame: original frame
    :param colors: iterable of integers represinting H value of the color
    :return: parsed frame
    """
    resized = cv2.resize(frame, FRAME_SIZE)  # resize frame
    hsv = cv2.cvtColor(resized, cv2.COLOR_BGR2HSV)  # from RGB to HSV
    masks = []
    for h in colors:
        masks.append(apply_mask(hsv, h))
    mask = sum(masks)
    mask = cv2.morphologyEx(mask, cv2.MORPH_OPEN,
    ↪  np.ones((3,3),np.uint8))
    mask = cv2.morphologyEx(mask, cv2.MORPH_DILATE,
    ↪  np.ones((3,3),np.uint8))
    # Segmenting the cloth out of the frame using bitwise_and with the
    ↪  inverted mask
    masked = cv2.bitwise_and(hsv, hsv, mask=mask)
    h, s, masked_grey = cv2.split(masked)
    return masked_grey
```

Appendix B

```
model = Sequential()
model.add(TimeDistributed(Conv2D(filters=32,
                                 kernel_size=(5,5),
                                 activation='relu',
                                 input_shape=(10, 212, 380, 1))))
model.add(TimeDistributed(MaxPooling2D(pool_size=(2,2))))
model.add(TimeDistributed(Dropout(0.25)))
model.add(TimeDistributed(Conv2D(filters=32,
                                 kernel_size=(3,3),
                                 activation='relu')))
model.add(TimeDistributed(MaxPooling2D(pool_size=(2,2))))
model.add(TimeDistributed(Dropout(0.25)))

model.add(TimeDistributed(Flatten()))

model.add(LSTM(64, dropout=0.25))
model.add(Dropout(0.25))

model.add(Dense(256, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))

model.compile(
    loss='categorical_crossentropy',
    optimizer='adam',
    metrics=['categorical_accuracy']
)
```