Ministry of Education and Science of Ukraine

National University “Kyiv-Mohyla Academy”

Faculty of Informatics

Mathematics Department

Master's thesis

educational level – master

on the topic: “**RECOGNIZING GESTURES OF THE UKRAINIAN DACTYLIC ALPHABET**”

By: 2-nd year student

of the educational program “Applied Mathematics”, 113

Bikchentaev Mykola Oleksiiovych

Supervisor: Hlybovets A. M.,

Doctor of Engineering

Reviewer_____

(surname and initials)

The master's thesis was defended with a grade _____

EC secretary _____

«____» _____ 2023 p.

Kyiv – 2023

CONTENTS

INTRODUCTION

Sign language is a visual way of communicating used by people who are deaf or hard of hearing. It involves handshapes, facial expressions, and body movements to convey meaning. Sign language helps the deaf community interact with each other and the hearing world, allowing them to fully participate in society.

According to the WHO (World Health Organization) over 5% of the world's population – or 430 million people – experience problems with hearing. More than 44,000 people with hearing impairments are registered with the Ukrainian Society of the Deaf, an all-Ukrainian public organization for the disabled[1].

In this context, applications for interpreting sign language can be very beneficial as they can narrow the communication gap between sign language users and those who are not familiar with sign language. They can be used in educational and healthcare settings to facilitate communication between hearing and non-hearing people.

Therefore, the **aim of this work** is to review the techniques for gesture recognition and develop a system for detecting and classifying gestures of the Ukrainian dactyl alphabet.

The object of study: Ukrainian sign language, Ukrainian dactyl alphabet, gesture recognition, LSTM, and Google MediaPipe.

Research methods: analysis of scientific literature.

Objectives of the study:

1. Study the concept of sign language and Ukrainian sign language in particular.
2. Review approaches to gesture recognition.
3. Build a model for recognizing gestures of the Ukrainian dactyl alphabet.

The work consists of an introduction, three chapters, a conclusion, and a list of references.

In *the first chapter*, we study the concept of sign language as well as review Ukrainian sign language and the Ukrainian dactyl alphabet.

In *the second chapter,* we review existing approaches to gesture recognition.

The third chapter is devoted to the LSTM network, Google MediaPipe, and their use for building a model for recognizing gestures of the Ukrainian dactyl alphabet. The process of collecting data for the model and evaluating its effectiveness is also discussed in this section.

The scientific novelty of the obtained results: the paper presents a method that utilizes hand keypoints for recognizing hand gestures of the Ukrainian dactyl alphabet. Also, as part of the development of the gesture recognition system, a data set was collected, where each gesture corresponds to 50 videos of 65 frames.

The practical significance of the results obtained: the model obtained as a result of the study can be used to interpret the gestures of the Ukrainian dactylic alphabet. The dataset collected for training this model can be used in other works to train or validate similar models. The paper might be of use to the ones who are interested in developing similar systems for gesture recognition.

1. Sign Language

1.1. Sign Language Definition

Sign language is a communication system that relies on visible cues, such as hand gestures, eye movements, facial expressions, and body language, to convey meaning. It is primarily used by people who are deaf or hard of hearing, but it can also be used in other contexts where verbal communication is not possible or practical.

Although both sign languages and gestures involve the use of the hands (and other parts of the body), they are somewhat different. Sign languages, like spoken languages, have evolved over time and have their own grammar and structural rules, and are used instead of speaking. Gestures, on the other hand, are mostly used during the verbal communication.

Examples of gestures include waving when saying “Hello” or “Goodbye”, pointing to an object or place, shrugging to indicate uncertainty, and giving a thumbs up or thumbs down to indicate approval or disapproval. The gestures just act as a supplement to verbal communication.

Sign language has many variations across different countries and regions. These variations reflect the unique cultures and customs of the people who use them. For instance, American Sign Language (ASL) is different from British Sign Language (BSL), which is different from Australian Sign Language. This means that sign language users may have difficulty communicating with people from other countries, even if they both use sign language[2].

Today, there are more than 300 different sign languages in the world, spoken by more than 72 million deaf or hard-of-hearing people worldwide[2].

1.2. Types of Ukrainian Sign Language

Deaf individuals use two kinds of sign language - *conversational sign language* and *mimetic sign language* (ukr. калькуюча жестова мова) - that differ in their linguistic structure and functional purpose[3].

Mimetic sign language, which is used alongside spoken language, is composed of gestures that function as replacements for spoken words and maintain the same sequence as words in a typical sentence. Since mimetic sign language lacks its own grammar, it relies on the grammar of the spoken language. Typically, it is employed by the deaf in formal situations like conferences, meetings, and so forth.

Conversational sign language, which we will further refer to as Ukrainian sign language or USL, differs from mimetic sign language in that it has its own syntax and sentence construction rules, rather than simply repeating the word order of spoken language[3].

To illustrate, in mimetic sign language, the phrase "There's a chair in the bottom left corner of the room, and a floor lamp is behind the chair" would be conveyed using several gestures representing individual concepts (left, bottom, corner, room, to stand, chair, floor lamp) and alphabet letters. In conversational sign language, however, the same phrase would be demonstrated as follows: the left hand of the speaker makes a gesture representing the concept of a chair, and the right hand makes a gesture representing the floor lamp, with both gestures being performed simultaneously but the left gesture being positioned to the left and closer to the speaker, thus reflecting the specified spatial relationship between the two objects[3].

The USL is typically used in a casual and informal setting, where conversations revolve around everyday events. Therefore, certain concepts are absent from the language, such as specialized designations that are only used in the learning process[3]. When the need arises to express such concepts, deaf people who have achieved a certain level of education can use mimetic sign language. However, this does not at all indicate the incompleteness or primitiveness of the USL, but rather emphasizes its functional purpose. This also explains the absence of specialized designations in the language's vocabulary, which are always present in spoken language situations, such as designations for the head, nose, hand, etc. They are always expressed by pointing to the head, nose, hand, etc. Such gestures are called *indicative gestures*[3].

Indicative gestures are widely used in conversational sign language and have a fairly wide range of functions. For example, if a deaf person "tells" a conversation partner the color of their new coat, they can point to a passerby, thereby showing that their coat color is the same as that of the passerby's coat. The lexical features of conversational sign language may also be related to kinesics. For instance, gestures that express the meaning of "going up" and "going down" differ only in the direction of movement: upward and downward, respectively[3].

Sign language is an independent language that allows the expression of any meanings and relationships between them. Therefore, it successfully solves the communication problem of the deaf in an informal, non-official environment.

Translating from USL to Ukrainian verbal language is a complex scientific and applied problem, the solution of which requires analyzing the grammar of Ukrainian sign language, developing translation rules from Ukrainian verbal language to sign language, and vice versa[3]. The absence of large dictionaries and corpora of Ukrainian sign language increases the complexity of developing a computer translation system for USL.

Therefore, in this work, we will develop a system for translating gestures from the USL alphabet which is a less complex task that requires smaller amounts of data.

1.3. USL (Ukrainian Sign Language) Alphabet

Fingerspelling (or *dactylology*) is the representation of the letters of a writing system, and sometimes numeral systems, using only the hands. These representations are gathered into manual alphabets (also known as finger alphabets, hand alphabets, or dactyl alphabets) that are often used in deaf education and have subsequently been adopted as a distinct part of several sign languages[4].

The Ukrainian dactyl alphabet is an auxiliary system of Ukrainian sign language in which each gesture of one hand corresponds to a single letter of the Ukrainian alphabet. The dactyl alphabet is used for pronouncing auxiliary words, words that lack gestural representation, as well as when it is necessary to clarify the meaning of a particular word. The modern Ukrainian dactyl alphabet includes 33 dactyl signs, which is the same as the number of letters in the Ukrainian alphabet[4].

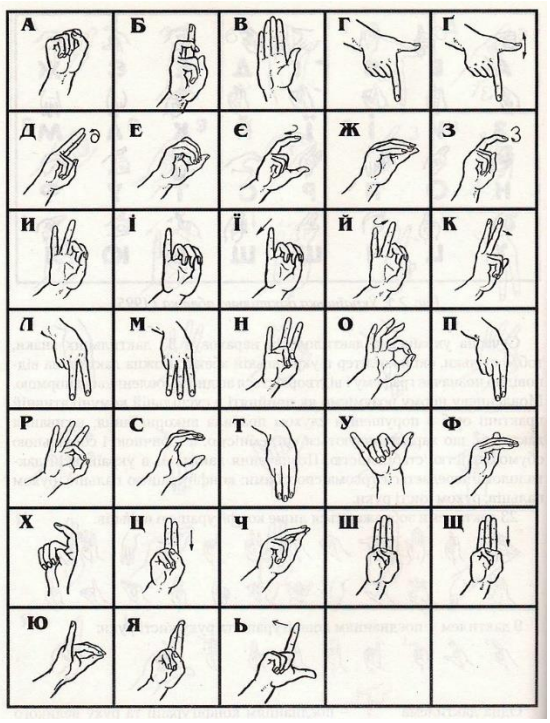


Figure 1. Ukrainian dactyl alphabet [4]

Each gesture in the Ukrainian dactyl alphabet can be reproduced in three ways:

- By finger movement;
- By wrist movement;
- By positioning one's fingers in a specific way.

23 gestures of the Ukrainian dactyl alphabet we will call “static” since they require only positioning one’s fingers in a specific way.

The remaining 10 gestures will be referred to as “dynamic” since they require one to position one's fingers in a specific way and move either wrist or finger.

The need for movement when reproducing some gestures is caused by the similarity of some of these gestures. We can list these similar gestures in the form of pairs: І-Ї, И-Й, Д-Ц, Г-Ґ, Ш-Щ, Х-З.

2. Approaches to Gesture Recognition

There are two approaches to gesture recognition: glove-based, which involves wearing some kind of gloves with sensors that capture hand motion and position in space, and CV-based which uses computer vision techniques and does not require wearing any sensors[5].

2.1. Glove-Based Gesture Recognition

Gloves with sensors can be used to capture data about hand position and movement. Such sensors can also quickly provide the exact coordinates of palm and finger locations and data about their orientation in space[6].

Nevertheless, despite the precision and variety of data this approach offers, it is fully dependent on specialized sensors, which may be quite expensive to purchase and maintain. Moreover, constantly wearing gloves to interpret gestures can be cumbersome[6].

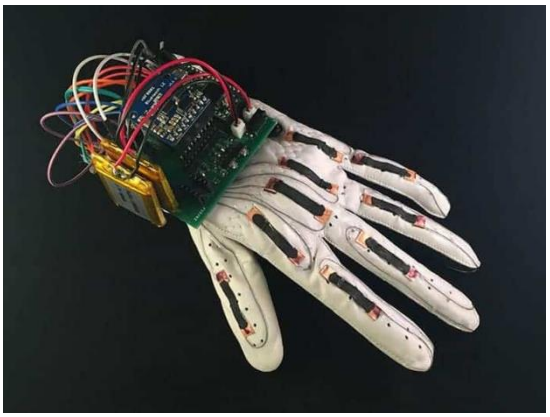


Figure 2. Wearable glove with sensors [5]

2.2. CV-Based Gesture Recognition

Computer vision or CV is a field of artificial intelligence and computer science that focuses on enabling computers to interpret and understand the visual world. It involves developing algorithms and techniques that allow computers to analyze and interpret images and video data. The CV-based approach does not

require any special equipment, except a computer camera, however, involves several challenges such as lighting variation, complex backgrounds, noisy images or videos, or occlusion[5].

We may view hand gesture recognition as an object detection task. Object detection is a set of computer vision tasks that deal with object localization and classification within an image or video. Object localization is a computer vision task that involves identifying the location of one or more objects in an image or video. Object classification is a computer vision task that assigns a class or category label to an object in an image or video. Therefore, the goal of an object detection algorithm is to locate the presence of objects with a bounding box and assign each object a class label.

There are several approaches to object detection which can be grouped into two categories: machine learning-based approaches and deep learning-based approaches[5].

Machine learning-based approaches use computer vision techniques to look at various features of an image, such as color or object edges, to identify groups of pixels that may belong to an object. These features are then fed into a machine learning algorithm, such as a support vector machine (SVM) or random forests (RDF), to classify the object.

For example, in [7, 8], approaches are presented that use image color spaces (such as RGB, HSV, and Y-Cb-Cr) to separate the hand from the background of the image. By separating the hand from the background we would have information about its shape which can be used to classify the hand gesture. In [9] Haar-like features are used for posture recognition as well as the AdaBoost learning algorithm to speed up the performance of the gesture classifier.

The performance of machine learning-based approaches is often limited by the quality of the features used and requires significant domain expertise to produce effective results.

On the other hand, deep learning-based approaches use neural networks to learn features from raw image data automatically and do not require hand-crafted features. This has led to significant improvements in the accuracy of object detection systems, and deep learning-based approaches have become state-of-the-art in the field. Deep learning-based approaches use neural network architectures like YOLO (You Only Look Once), SSD (Single Shot Detector), or CNN for object localization and the extraction of the features that are then used for gesture classification.

For instance, in [10, 11] deep convolutional neural networks (CNN) are used to classify images with hand gestures. The model proposed in [10] is used to recognize gestures from the Ukrainian dactyl alphabet and achieves an accuracy of 97%. The model presented in [11] achieves 99% accuracy in recognizing hand gestures of post-stroke people. In [12, 13] models utilize hand keypoints to classify hand gestures. In [12] hand keypoints are classified using a single-shot, heuristics-based classifier that achieves a 0.86% false positive rate and 44.4% recall rate on the test dataset. In [13] the classifier is based on the LSTM network and achieves an accuracy of 92.54% on the test dataset.

3. System for Recognizing USL Alphabet Gestures

Since deep learning-based approaches give good results in object detection and are the most popular in the field of computer vision, in this work, we will be utilizing two deep neural networks to detect and classify USL alphabet gestures. The first one, a pre-trained neural network, called *HandLandmarker*, will be used to detect a hand on the video and extract its keypoints[14]. The second one will classify a hand gesture on the video using the extracted keypoints.

3.1. Google MediaPipe

Google MediaPipe is a set of tools and libraries that allow its users to apply machine learning methods to solve such problems as face detection, hand landmark detection, or pose landmark detection. Models proposed within the MediaPipe library are open-source, fully customizable, and can be set up both on mobile devices and computers.

The HandLandmarker model is a part of the Google MediaPipe library and can be used to detect and extract hand landmarks from an image or video. This model is composed of two parts: a single-shot detector, called *BlazePalm*, which is used to detect initial hand locations, and a regression model which is used to extract hand landmarks and handedness (i.e. whether the detected hand is left or right)[14, 15].

HandLandmarker outputs a list of 21 hand keypoint, where each keypoint has x, y, and z coordinates. Below you can see an image with keypoint names and their position relative to each other[14].

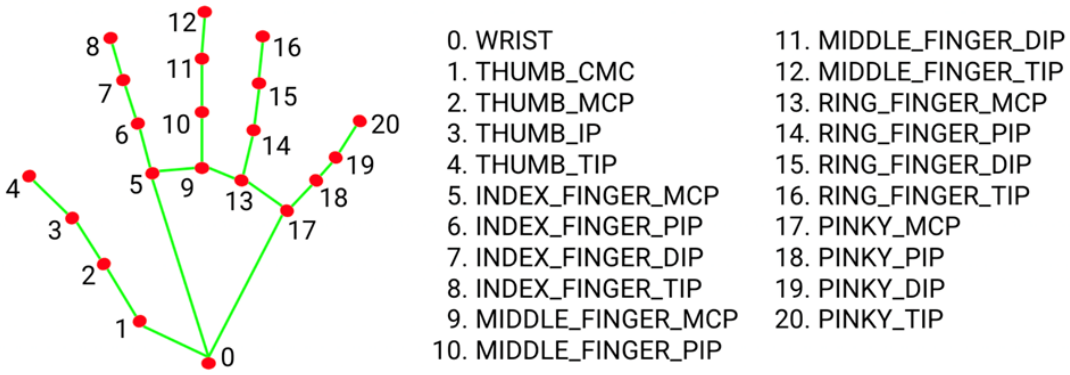


Figure 3. Keypoints output by Google MediaPipe HandLandmarker

3.2. LSTM Network

An *LSTM (Long Short-Term Memory) network* is a type of recurrent neural network (RNN) that can learn long-term relationships between input data and is capable of processing the entire sequence of data, such as video, rather than individual elements, such as video frames[16]. Since the output of HandLandmarker is a list of hand landmarks, the LSTM network is well-suited for their processing and, therefore, will be used as a basis for gesture classifier.

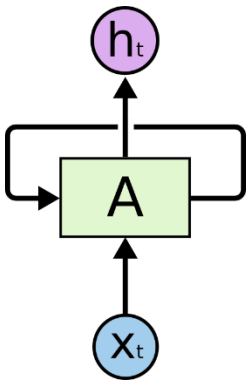


Figure 4. LSTM network structure [17]

The LSTM network, displayed in the diagram above, uses input data x_t to calculate the output h_t and has a loop that allows information to be passed from one step of input data processing to the next. We can simplify this diagram by representing LSTM as a set of repeating "modules", where each module passes data to a successor[17]. All modules have the same structure and consist of three "gates": *forget gate*, *input gate*, and *output gate*. Gates control how the information in a sequence of data comes into, is stored in, and leaves the network[17].

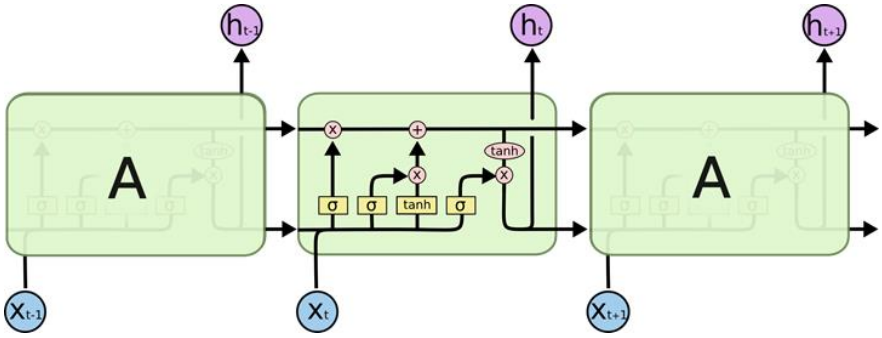


Figure 5. Simplified LSTM network structure [17]

One of the most important concepts on which LSTM is built is called *cell state*. Cell state is the long-term memory of the network which is available to each LSTM module. The state of the long-term memory is changed using gates, mentioned above[17].

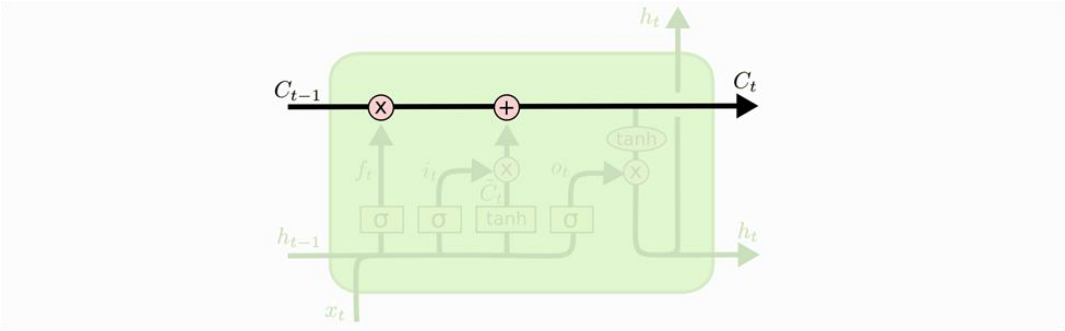
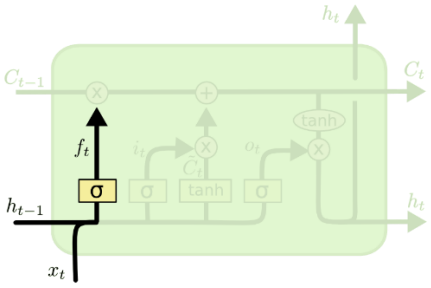


Figure 6. LSTM cell state [17]

The first gate (the forget gate) is responsible for determining how much of the information from long-term memory we can "forget." To do this, we multiply the cell state, denoted by C_{t-1}, by vector $f_t = \{v_1, v_2, \dots, v_n\} \in [0,1]^n$, which is calculated using the input data x_t, output of the previous LSTM module h_{t-1}, weight W_f, bias b_f, and *sigmoid function*[17]:

$$\sigma = \frac{e^x}{e^x + 1}$$

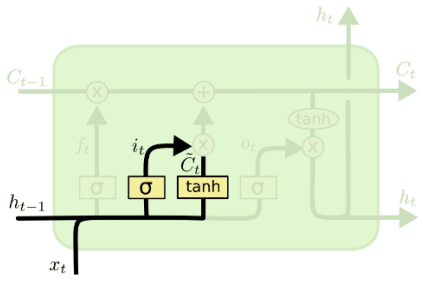


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Figure 7. LSTM forget gate [17]

In the f_t vector, a value close to 0 indicates that the corresponding long-term memory component is not relevant, and vice versa, a value close to 1 indicates that the component is relevant. We can think of the components of this vector as filters that let more information pass through them when the values are close to 1[18].

The second gate (the input gate) determines what new information should be added to the cell state. Firstly, we calculate a vector of updated long-term memory values denoted by $\tilde{C}_t$[17]. This vector shows us how much each component of the long-term memory (cell state) of the network needs to be updated with new data[18].



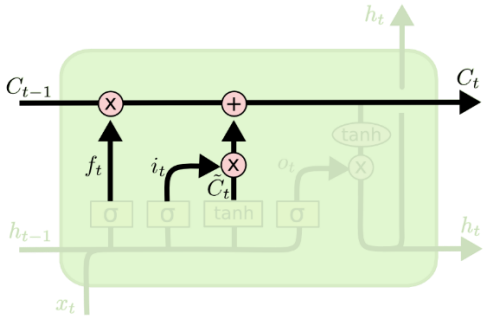
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Figure 8. LSTM input gate [17]

When we calculate $\tilde{C}_t$ we use the $tanh$ (*hyperbolic tangent*) function because its values lie in the range $[-1,1]$[17]. The possibility of negative values here is necessary if we want to reduce the influence of the update vector component on the cell state[18].

$$tanh = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

$\tilde{C}_t$ is then multiplied by vector i_t, which is similar to f_t in its purpose. However, in this case, a value close to 0 in vector i_t will indicate that a corresponding element of the cell state should not be updated[17].

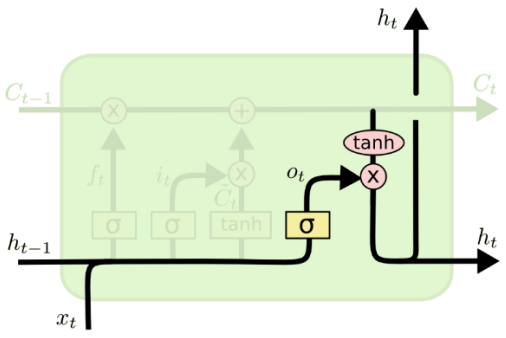


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Figure 9. Updating LSTM cell state [17]

The result of the multiplication of $\tilde{C}_t$ by i_t is added to the cell state resulting in the long-term memory of the network being updated.

The third and last gate (the output gate) decides which data will be output by the LSTM module[17]. To determine the output we will use the output of the previous LSTM module h_{t-1}, input data x_t, and the newly updated cell state C_t.



$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Figure 10. LSTM output gate [17]

Firstly, we create a filter o_t , which is similar to f_t in the forget gate; the inputs are the same as well as the activation function. This filter is applied to the updated cell state C_t to make sure that only the necessary information is output. Before applying the filter, however, we pass the cell state to the tanh function so that output values were in the $[-1,1]$ interval[17].

3.3. Dataset Creation

Deep neural networks require a significant amount of data to train them. For the Ukrainian dactyl alphabet, there are no publicly-available data sets that can be used to train the model, so it was decided to create our own set.

The resulting dataset consists of 33 classes. Each class corresponds to 50 videos, which are further divided into 65 frames.

Initially, it was decided to use YouTube videos to create the dataset. However, even though the quality of the videos found was high, the number of videos was too small to train the network and required additional preprocessing. In total, six videos from YouTube were used, as well as videos from the Spread The Sign resource, an online multilingual sign languages dictionary.

Each of the six YouTube videos was divided into video segments depicting a particular gesture of the dactyl alphabet. After that, the video fragments were divided into 65 frames; if a video fragment consisted of fewer frames, it was "looped" by adding the original frames to the end of the fragment.

To record the rest of the video, a program was developed in the Python programming language using the OpenCV and Google MediaPipe libraries. The program uses a webcam to record a 65-frame video. After recording, the video frames are saved in JPG format on a computer disk.

After each recorded video, the program pauses for 5 seconds to let you prepare for the recording of the next video. During the video recording, the program displays the code of the gesture, for which a video is being recorded as well as the number of the video.

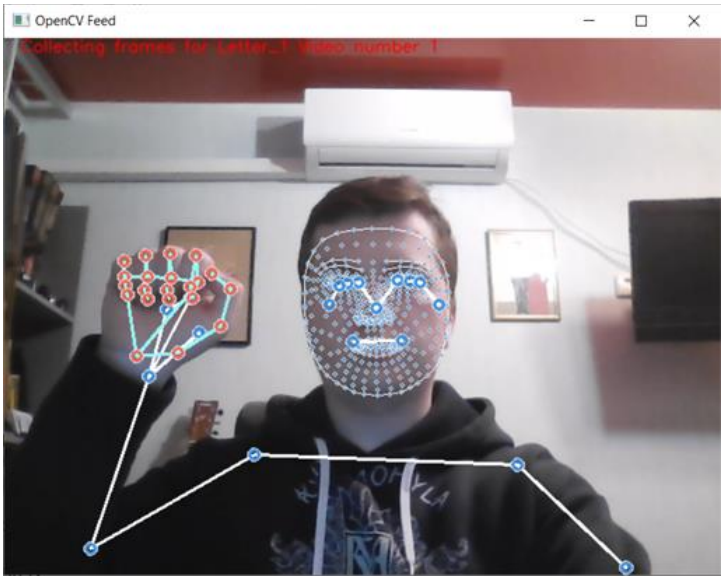


Figure 11. Screenshot of a program for recording gesture videos

Code for the video recording program can be seen below. The `capture_videos` function reads frames from the computer web camera and saves them as JPG files on the computer disk using the `imwrite` function. Frames are

also shown in the program window with Google MediaPipe keypoints drawn on them using the `draw_styled_landmarks` function.

```
def capture_videos(mp_holistic, mp_drawing, letter, videos_nums, frames_num, destination):
    cap = cv2.VideoCapture(0)
    with mp_holistic.Holistic(min_detection_confidence=0.5, min_tracking_confidence=0.5) as holistic:
        for video_num in videos_nums:
            if not os.path.exists(os.path.join(destination, letter, f'{letter}_{video_num}')):
                os.makedirs(os.path.join(destination, letter, f'{letter}_{video_num}'))
            for frame_num in range(frames_num):
                ret, frame = cap.read()
                image, results = mediapipe_detection(frame, holistic)
                image_copy = image.copy()
                draw_styled_landmarks(image_copy, results, mp_drawing)
                if frame_num == 0:
                    cv2.putText(image_copy, '!STARTING COLLECTION IN 5 SEC. GET READY!', (120, 200),
                                cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 4, cv2.LINE_AA)
                    cv2.putText(image_copy,
                                f'Collecting frames for {letter} Video number {video_num}',
                                (15, 12),
                                cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 1, cv2.LINE_AA)
                else:
                    cv2.putText(image_copy,
                                f'Collecting frames for {letter} Video number {video_num}',
                                (15, 12),
                                cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 1, cv2.LINE_AA)
                cv2.imwrite(os.path.join(destination,
                                         letter,
                                         f'{letter}_{video_num}',
                                         f'{letter}_{video_num}_Frame_{frame_num + 1}.jpg'), image)
                cv2.imshow('OpenCV Feed', image_copy)
                if frame_num == 0:
                    cv2.waitKey(5000)
                if cv2.waitKey(10) & 0xFF == ord('q'):
                    break
    cap.release()
    cv2.destroyAllWindows()
```

Figure 12. The capture_videos program

On Figure 13 you can see a few videoframes from the dataset with keypoints (obtained using Google MediaPipe) displayed on top of them. It is worth noting that only the right hands are depicted in all video frames in the resulting dataset.

After collecting the required number of video frames for each gesture, Google MediaPipe and NumPy are used to extract key points from each video frame and save them to a computer disk. This significantly speeds up the processing of keypoints, as we can no longer need to work with images, instead we can work with lists of numbers.

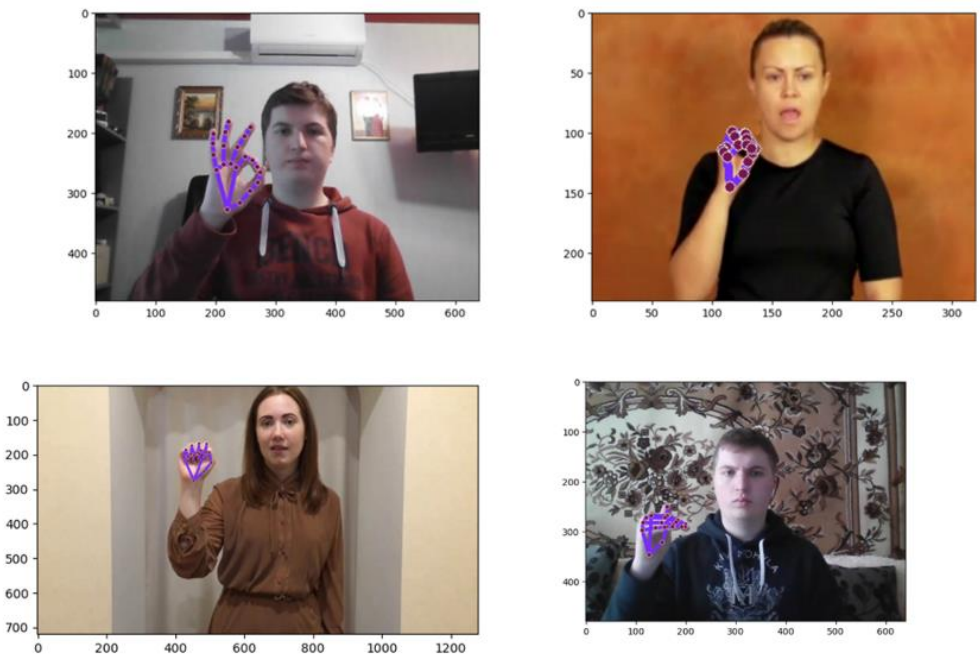


Figure 13. Sample of images from the collected dataset

3.4. Model Training

The gesture classification model consists of 6 layers. The first 3 layers use LSTMs with the activation function $tanh$ (hyperbolic tangent). The remaining layers are regular densely-connected layers, 2 of which use ReLU as an activation function, while the last layer uses Softmax.

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}, \quad Softmax(y)_i = \frac{e^{y_i}}{\sum_{j=1}^{n} e^{y_j}}, \quad ReLU(x) = \max(0, x)$$

Adam algorithm is used as an optimizer for the model. Optimizers are algorithms that are used to adjust learnable parameters of the model (i.e. weights and biases) to minimize the loss function and maximize mode efficiency. Adam optimization algorithm is an extension of stochastic gradient descent and is widely used for deep learning applications in computer vision and natural language processing[19].

The *cross-entropy loss* will be used as a loss function. This function is often used for multi-class classification problems and is defined as follows:

$$CE = -\sum_{i=1}^{n} t_i \log\left(Sofrmax(s)_i\right)$$

where n is the total number of classes, t_i is a true probability for the i-th class, and $Softmax(s)_i$ is a predicted probability for the same class[20].

The model was trained on an NVIDIA GeForce 1660 Ti GPU. The training lasted 1416 epochs, with the model's accuracy stopping at 99% after 960 epochs. The training dataset was 20% of the total dataset. The total number of parameters, used by the model, is 188321.

In the images below, we can see the number of parameters used by the mode and how the model's accuracy and loss function value changed during the training.

Layer (type)	Output Shape	Param #
lstm_12 (LSTM)	(None, 65, 64)	32768
lstm_13 (LSTM)	(None, 65, 128)	98816
lstm_14 (LSTM)	(None, 64)	49408
dense_12 (Dense)	(None, 64)	4160
dense_13 (Dense)	(None, 32)	2080
dense_14 (Dense)	(None, 33)	1089

Total params: 188,321
Trainable params: 188,321
Non-trainable params: 0

Figure 14. Model layers and number of paramaters

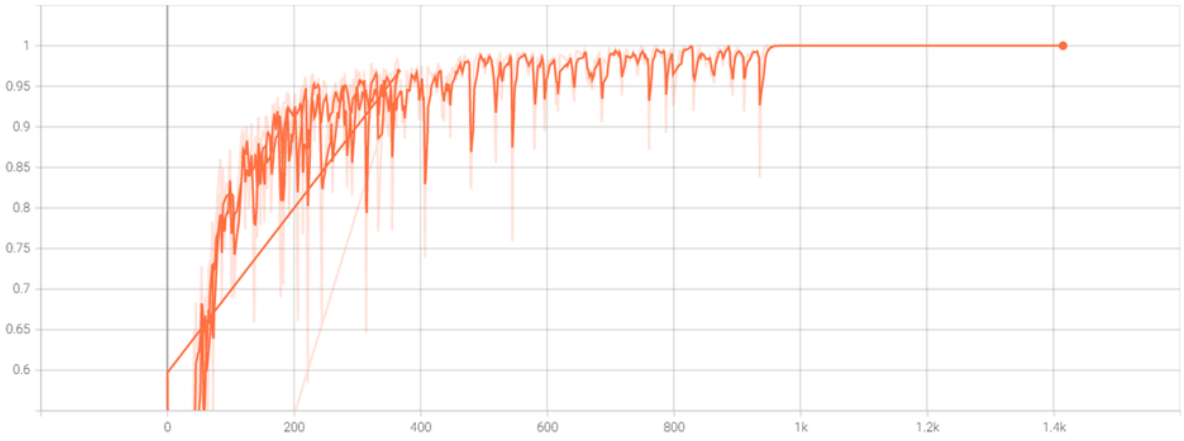


Figure 15. Changes in model accuracy as the number of epochs increases

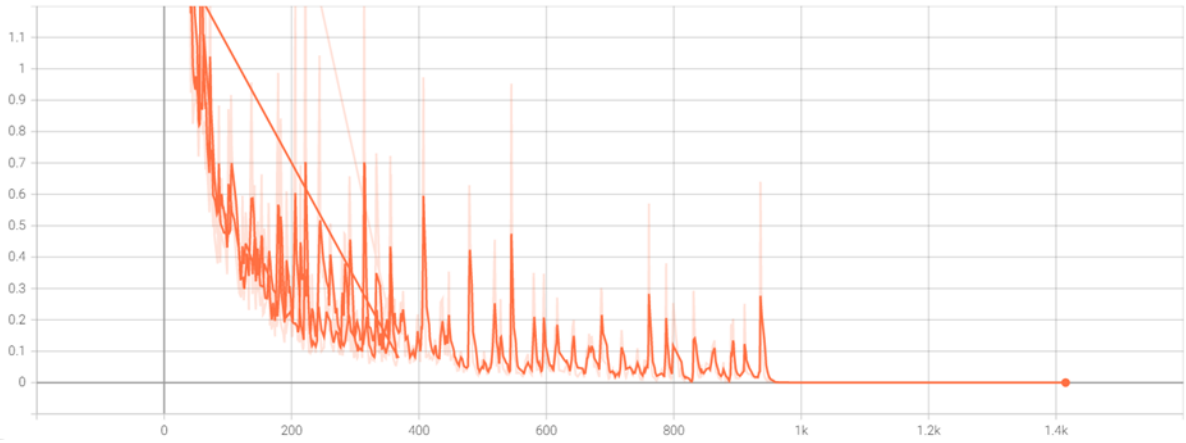


Figure 16. Changes in the model loss function value with an increasing number of epochs

3.5. Model Evaluation

To evaluate the performance of the trained model we will use metrics like classification accuracy, precision, recall, and F1 score; we will also plot the confusion matrix and ROC curve.

Classification accuracy (or *accuracy*) is the ratio of correct predictions to the total number of predictions made[21].

$$Accuracy = \frac{Num.of\ Correct\ Predictions}{Total\ Num.of\ Predicitons}$$

The accuracy of the model for gesture classification on the test dataset is 98.4%.

```
In [49]: print(f'Model accuracy: {round(accuracy_score(ytrue, yhat) * 100, 2)}%')
         Model accuracy: 98.4%
```

Figure 17. Model accuracy

However, this metric may be misleading if the dataset is imbalanced (i.e. when some classes have more samples than others). For this, we will supplement accuracy with other metrics that measure classification performance.

A *confusion matrix* is a $n \times n$ matrix, where n is the number of classes the model is trying to predict. It provides a detailed summary of the model's predictions and the actual values from a dataset by grouping the classification results into four categories: *True Positive*, *False Positive*, *True Negatives*, and *False Negatives*[21].

Suppose we are dealing with a binary classification problem, where we have a set of samples categorized into two classes: *Yes* and *No*. Additionally, we have developed our own classifier that assigns a class to each input sample. After evaluating our model on 100 samples, we obtained the following outcome.

$n = 100$	**Predicted No**	**Predicted Yes**
Actual No	40	10
Actual Yes	5	45

In this case, the four categories, mentioned above, can be interpreted the following way:

- *True Positives (TP)*: the cases when both actual and predicted values are “Yes”.
- *False Positives (FP)*: the cases in which the actual value is “No” but the predicted value is “Yes”.
- *True Negatives (TN)*: the cases when both actual and predicted values are “No”.

- *False Negatives (FN)*: the cases in which the actual value is “Yes” but the predicted value is “No”.

Applied to the example above, values will be the following:

- True Positives (TP): 45
- False Positives (FP): 10
- True Negatives (TN): 40
- False Negatives (FN): 5

In the case of the model for classifying USL alphabet gestures, the confusion matrix will be a 33×33 matrix, which can be seen below.

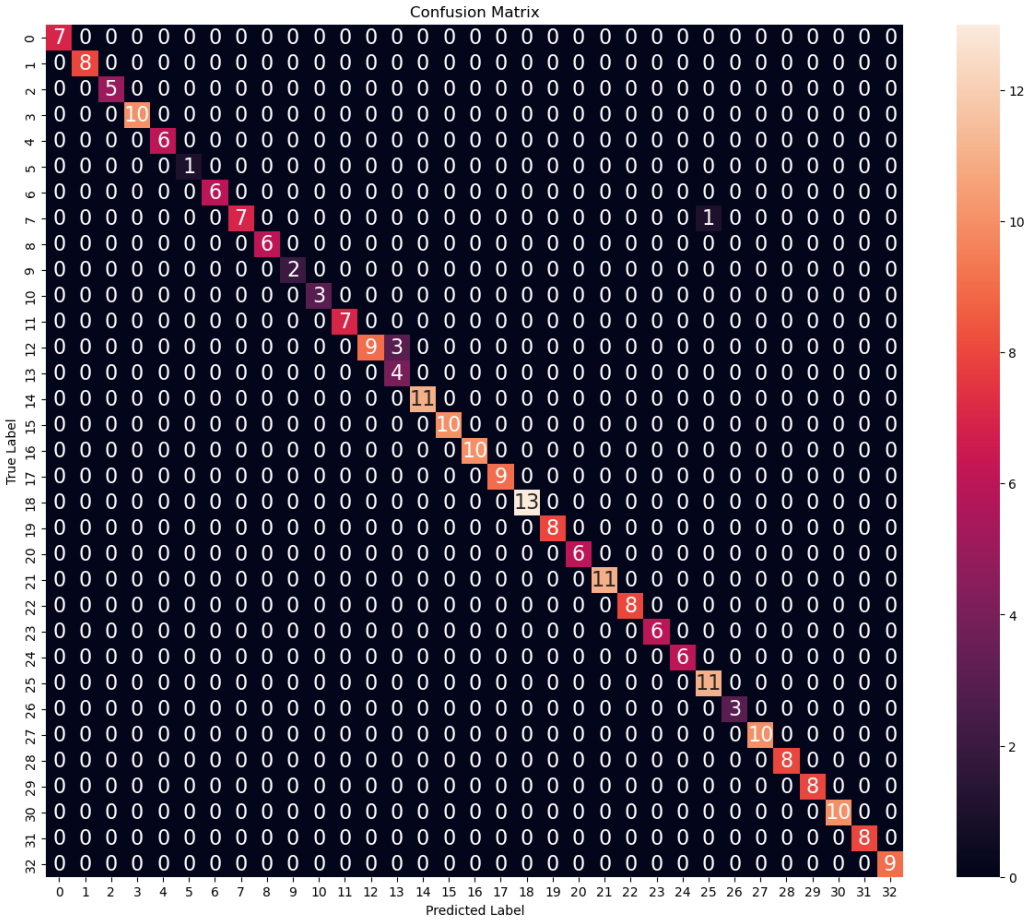


Figure 18. Model confusion matrix

Using the values from the confusion matrix we can calculate *precision* and *recall* for each class using the following formulas[21]:

$$Precision = \frac{TP}{TP + FP}, \quad Recall = \frac{TP}{TP + FN}$$

Precision refers to the percentage of positive identifications that were accurate, while recall informs us about the proportion of actual positives that were identified correctly.

To comprehensively assess model performance, it is necessary to consider both precision and recall. However, enhancing precision typically leads to a decrease in recall, and vice versa.

The *F1 score* is the harmonic mean between precision and recall. It provides a value between 0 and 1, indicating the precision of the classifier (how many instances are classified correctly) and its robustness (it does not miss relevant instances)[21]. The F1 score can be calculated using the following formula:

$$F1 = \frac{2 * Precision * Recall}{Precision * Recall}$$

Below we can see precision, recall, and F1 score calculated for each class of the USL alphabet gesture classification model.

	precision	recall	f1-score	support
0	1.00	1.00	1.00	7
1	1.00	1.00	1.00	8
2	1.00	1.00	1.00	5
3	1.00	1.00	1.00	10
4	1.00	1.00	1.00	6
5	1.00	1.00	1.00	1
6	1.00	1.00	1.00	6
7	1.00	0.88	0.93	8
8	1.00	1.00	1.00	6
9	1.00	1.00	1.00	2
10	1.00	1.00	1.00	3
11	1.00	1.00	1.00	7
12	1.00	0.75	0.86	12
13	0.57	1.00	0.73	4
14	1.00	1.00	1.00	11
15	1.00	1.00	1.00	10
16	1.00	1.00	1.00	10
17	1.00	1.00	1.00	9
18	1.00	1.00	1.00	13
19	1.00	1.00	1.00	8
20	1.00	1.00	1.00	6
21	1.00	1.00	1.00	11
22	1.00	1.00	1.00	8
23	1.00	1.00	1.00	6
24	1.00	1.00	1.00	6
25	0.92	1.00	0.96	11
26	1.00	1.00	1.00	3
27	1.00	1.00	1.00	10
28	1.00	1.00	1.00	8
29	1.00	1.00	1.00	8
30	1.00	1.00	1.00	10
31	1.00	1.00	1.00	8
32	1.00	1.00	1.00	9
accuracy			0.98	250
macro avg	0.98	0.99	0.98	250
weighted avg	0.99	0.98	0.98	250

Figure 19. Precision, recall, and F1 score calculate for each class

The *ROC (Receiver Operating Characteristic) curve* is a graphical representation of the trade-off between the *true positive rate (TPR)* and the *false positive rate (FPR)* as the classification threshold of the model is varied[22].

True positive rate is a synonym for recall, which was mentioned earlier; the false positive rate is defined as follows:

$$FPR = \frac{FP}{FP + TN}$$

The curve is created by plotting the TPR on the y-axis against the FPR on the x-axis for different threshold values. Below you can see an example of a ROC curve.

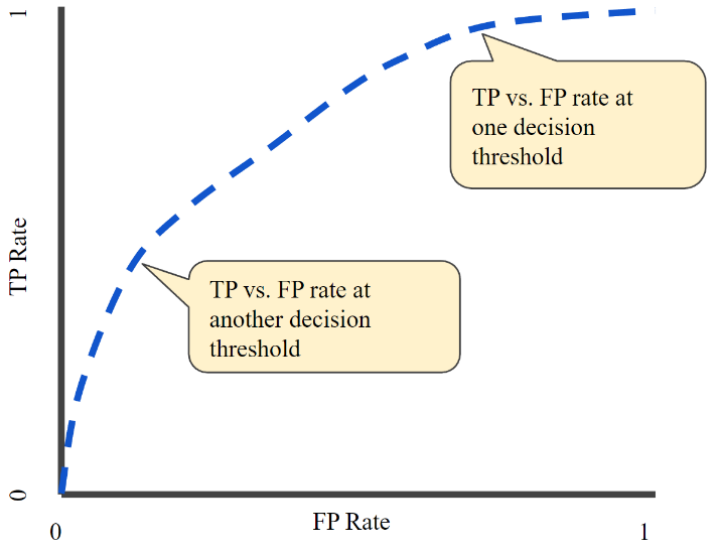


Figure 20. ROC curve example [22]

AUC (Area Under the Curve) measures the entire two-dimensional area underneath the entire ROC curve from (0,0) to (1,1). The AUC gives a general measure of how well the model performs across all possible thresholds. The higher the AUC, the more accurate the model is in predicting 0 classes as 0 and 1 classes as 1[22].

AUC is scale-invariant since it measures how well accurate the predictions are rather than their absolute values. AUC is also classification-threshold-invariant as it measures the quality of the model's predictions irrespective of the chosen classification threshold.

ROC curves are commonly utilized in scenarios involving binary classification, where the true positive rate (TPR) and false positive rate (FPR) can be clearly defined. However, when dealing with multiclass classification, as in our case, determining the TPR or FPR requires transforming the output into a binary form. This can be done in 2 different ways[23]:

- The *One-vs-Rest (OvR) scheme*: compares each class against all the others.
- The *One-vs-One (OvO) scheme*: compares every unique pairwise combination of classes.

To evaluate the model for classifying USL alphabet gestures we will use the One-vs-Rest scheme. This strategy consists in computing a ROC curve per each of the n classes. In each step, a given class is regarded as the positive class and the remaining classes are regarded as the negative class as a bulk.

Below you can see a ROC curve that measures the performance of the USL alphabet gestures classifier plotted using the OvR technique.

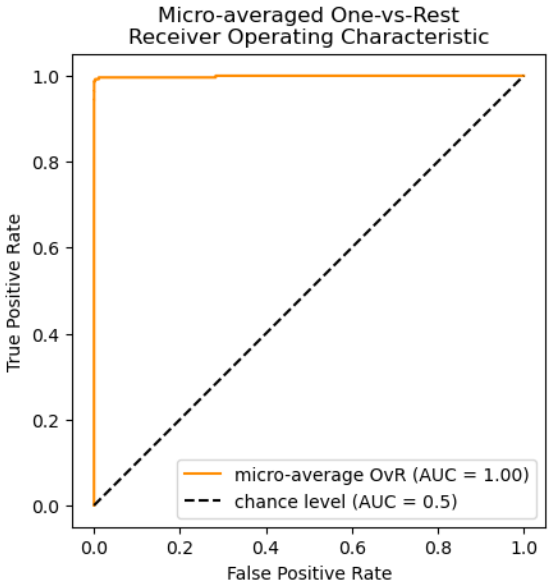


Figure 21. Model ROC curve

3.6. A Program for Gestures Recognition

A program to classify USL alphabet hand gestures is written in Python programming language and uses a video feed from a web camera to detect and classify hand gestures in real-time. The operation of the program is supported by libraries such as OpenCV, which is used to display a program window as well as classification results, and Google MediaPipe, which extracts hand keypoints from video frames.

Extracted keypoints are then processed by a pre-trained classifier that outputs a probability distribution with probabilities for each of the 33 letters of the Ukrainian alphabet. Three letters with the highest probabilities are shown in the program window.

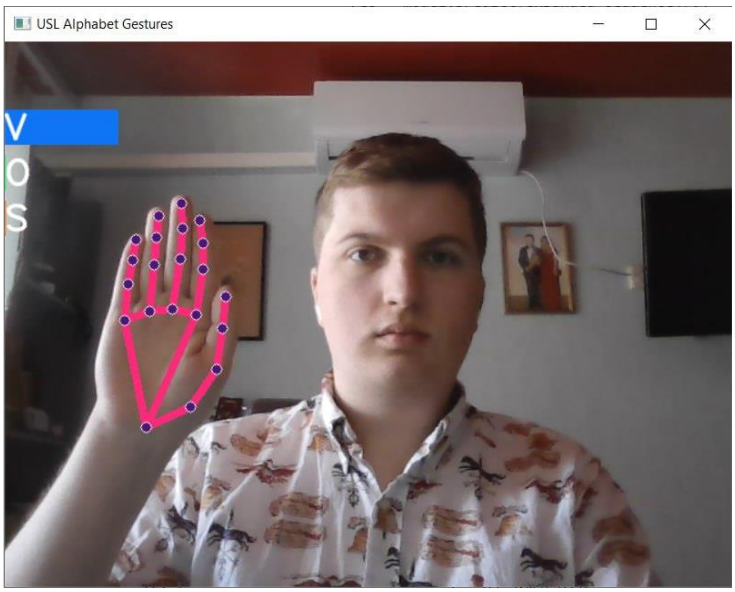


Figure 22. Screenshot of a program for gesture recognition

Figure 21 displays a screenshot of the program window that displays classification results for a gesture corresponding to the letter “B” of the Ukrainian alphabet. Since cyrillic letters are not supported by the fonts, provided by OpenCV, all letters from the Ukrainian alphabet were transliterated.

Gesture detection and classification are carried out only when a hand is visible to the web camera. By pressing “q” on the keyboard one can shut down the program. Below you can see a function that preforms gesture recognition.

```
def interpret_gestures(model, ukrainian_alphabet_indexed, window_size = 65, verbose = False):
    sequence = []
    colors = [(245,117,16), (117,245,16), (16,117,245)]
    cap = cv2.VideoCapture(0)
    with mp_hands.Hands(min_detection_confidence=0.5, min_tracking_confidence=0.5, max_num_hands=2) as
hands:
        while cap.isOpened():
            ret, frame = cap.read()
            image, results = mediapipe_detection(frame, hands)
            if results.multi_hand_landmarks:
                draw_styled_hands_landmarks(image, results)
                keypoints = extract_hands_keypoints(results)
                if(len(keypoints) == 63):
                    sequence.append(keypoints)
                    sequence = sequence[-window_size:]
                if len(sequence) == window_size:
                    try:
                        expanded_sequence = np.expand_dims(sequence, axis=0)
                        res = model.predict(expanded_sequence)[0]
                        res_enumerated = list(enumerate(res))
                        res_sorted = sorted(res_enumerated, key = lambda x: x[1])
                        res_sorted.reverse()
                        most_likely_predictions = list(map(lambda item: (labels[item[0]], item[1]),
                                                           res_sorted))
                        most_likely_prediction_letters = get_letters(most_likely_predictions,
                                                                     ukrainian_alphabet_indexed)[:3]
                        if verbose:
                            print("Most Likely Letter:", most_likely_prediction_letters[0])
                            print(most_likely_prediction_letters)
                        image = score_viz(most_likely_prediction_letters, image, colors)
                    except:
                        print("Something went wrong!")
            else:
                sequence = []
            cv2.imshow('USL Alphabet Gestures', image)
            if cv2.waitKey(10) & 0xFF == ord('q'):
                break
    cap.release()
    cv2.destroyAllWindows()
```

Figure 23. The interpret_gestures function

This function reads 65 frames from a web camera and passes them to the Google MediaPipe model for keypoints extraction. Number of frames read can be changed using the `window_size` argument of the function.

Extracted keypoints are then passed to the classifier. The classification results are then transformed into a list of tuples using the `get_letters` function. Each tuple will contain a transliterated letter of the Ukrainian alphabet and the probability corresponding to it.

Three predictions with the highest probabilities are then shown on the screen using the `score_viz` function.

CONCLUSION

In this paper, the concepts of sign language, on the example of the Ukrainian sign language, and the Ukrainian dactylic alphabet were considered. Different approaches to gesture recognition were considered and a model based on LSTM and Google MediaPipe was built to classify Ukrainian dactylic alphabet gestures.

Comparing the resulting model with similar works, it can be noted that its accuracy is higher than that of the models presented in [12] and [10], where it is equal to 92.54% and 97% respectively.

To train the model, a dataset of video recordings of Ukrainian dactylic alphabet gestures was collected, where each gesture corresponds to 50 videos of 65 frames each. To record video gestures, a program was created in the Python programming language.

The classifier for the Ukrainian dactylic alphabet gestures was used to develop a program that, using a video stream from a computer webcam, recognizes the gesture shown and displays the recognition result on the screen.

The program can be further improved by adding the ability to compose sentences from the detected letters of the Ukrainian dactylic alphabet.

REFERENCES

1. Люди з вадами слуху потребують особливої уваги і захисту в період воєнної агресії // *Секретаріат Уповноваженого Верховної Ради України з прав людини*. URL: https://ombudsman.gov.ua/news_details/lyudi-z-vadami-sluhu-potrebuyut-osoblivoyi-uvagi-i-zahistu-v-period-voyennoyi-agresiyi (visited 03.06.2023).
2. International Day of Sign Languages // *United Nations*. URL: https://www.un.org/en/observances/sign-languages-day (visited 04.06.2023).
3. Крак Ю.В., Бармак О.В., С.О. Романишин. Узагальнені граматичні конструкції для автоматизованого перекладу з української мови на українську жестову мову // *Штучний інтелект*. 2011. №3. С. 136-146. URL: http://dspace.nbuv.gov.ua/bitstream/handle/123456789/59837/11-Krak.pdf?sequence=1 (visited 03.06.2023).
4. Засенко В.В., Кульбіда С.В. Дактилологія // *Енциклопедія сучасної України.* Київ: Інститут енциклопедичних досліджень. 2007. Т №7. С. 496. URL: https://lib.iitta.gov.ua/711107/ (visited 04.06.2023).
5. Munir Oudah, Ali Al-Naji, Ali Al-Naji. Hand Gesture Recognition Based on Computer Vision: A Review of Techniques // *Journal of Imaging* 6, no. 8: 73. 2020. URL: https://doi.org/10.3390/jimaging6080073 (visited 03.06.2023).
6. L. Dipietro, A. M. Sabatini and P. Dario. A Survey of Glove-Based Systems and Their Applications. // *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews).* 2008*.* vol. 38, no. 4. P. 461-482. URL: http://dx.doi.org/10.1109/TSMCC.2008.923862 (visited 03.06.2023).
7. Shaik K.B., Ganesan P., Kalist V., Sathish B.S., Jenitha J.M.M. Comparative study of skin color detection and segmentation in HSV and YCbCr color space // *Procedia Comput. Sci.* 2015. P. 41-48.
8. Zarit B.D., Super B.J., Quek F.K.H. Comparison of five color models in skin pixel classification // *In Proceedings of the International Workshop on Recognition, Analysis, and Tracking of Faces and Gestures in Real-Time Systems, In Conjunction with ICCV'99 (Cat. No. PR00378).* Corfu, Greece. 1999. P. 58-63.
9. Chen Q., Georganas N.D., Petriu E.M. Real-time vision-based hand gesture recognition using haar-like features // *In Proceedings of the 2007 IEEE Instrumentation & measurement technology conference IMTC.* Warsaw, Poland. 2007. P. 1-6.

10. Кондратюк С. С. Розпізнавання та моделювання жестів української дактильної абетки за допомогою кросплатформених технологій // 2021.
11. Alnaim N., Abbod, M., Albar, A. Hand Gesture Recognition Using Convolutional Neural Network for People Who Have Experienced A Stroke // *In Proceedings of the 2019 3rd International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT)*. Ankara, Turkey. 2019. P. 1-6.
12. Марчук Д.К., Левківський В.Л., Марчук Г.В., Голенко М.Ю. Система розпізнавання дактильної мови української абетки // 2022. URL: https://doi.org/10.32782/2663-5941/2022.6/19 (visited 03.06.2023).
13. George Sung, Kanstantsin Sokal, Esha Uboweja, Valentin Bazarevsky, Jonathan Baccash, Eduard Gabriel Bazavan, Chuo-Ling Chang, Matthias Grundmann. On-device Real-time Hand Gesture Recognition // 2021. URL: https://doi.org/10.48550/arXiv.2111.00038 (visited 03.06.2023).
14. Hand landmarks detection guide // *Google MediaPipe*. URL: https://developers.google.com/mediapipe/solutions/vision/hand_landmarker (visited 04.06.2023).
15. MediaPipe Hand (Lite/Full) Model Card. URL: https://drive.google.com/file/d/1-rmIgTfuCbBPW_IFHkh3f0-U_lnGrWpg/preview (visited 03.06.2023).
16. Klaus Greff, Rupesh K. Srivastava, Jan Koutnik, Bas R. Steunebrink, Jurgen Schmidhuber. LSTM: A Search Space Odyssey // *IEEE Transactions on Neural Networks and Learning Systems*. 2017. vol. 28, no. 10. P. 2222-2232. URL: https://doi.org/10.48550/arXiv.1503.04069 (visited 03.06.2023).
17. Christopher Olah. Understanding LSTM Networks // 2015. URL: https://colah.github.io/posts/2015-08-Understanding-LSTMs/ (visited 03.06.2023).
18. Rian Dolphin. LSTM Networks | A Detailed Explanation // 2020. URL: https://towardsdatascience.com/lstm-networks-a-detailed-explanation-8fae6aefc7f9 (visited 03.06.2023).
19. Jason Brownlee. Gentle Introduction to the Adam Optimization Algorithm for Deep Learning // 2017. URL: https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/ (visited 03.06.2023).
20. Raúl Gómez Bruballa. Understanding Categorical Cross-Entropy Loss, Binary Cross-Entropy Loss, Softmax Loss, Logistic Loss, Focal Loss, and all those confusing names // 2018. URL: https://gombru.github.io/2018/05/23/cross_entropy_loss/ (visited 03.06.2023).
21. İrem Tanrıverdi. Model Evaluation Metrics in Machine Learning // 2021. URL: https://medium.com/analytics-vidhya/model-evaluation-metrics-in-machine-learning-928999fb79b2 (visited 03.06.2023).

22. Google Machine Learning Education. Classification: ROC Curve and AUC. URL: https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc (visited 03.06.2023).
23. Multiclass Receiver Operating Characteristic (ROC) // *scikit-learn*. URL: https://scikit-learn.org/stable/auto_examples/model_selection/plot_roc.html (visited 04.06.2023)