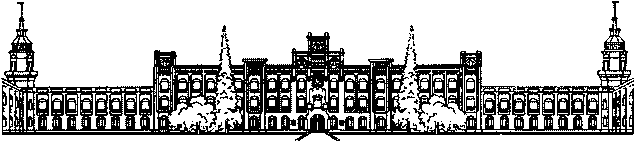
****

Міністерство освіти і науки України

Національний технічний університет України

“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет інформатики та обчислювальної техніки

Кафедра інформаційних систем та технологій

Лабораторна робота №7

**З дисципліни «Технології Computer Vision»**

*ДОСЛІДЖЕННЯ ТЕХНОЛОГІЙ ІДЕНТИФІКАЦІЇ ОБ’ЄКТІВ НА ЦИФРОВИХ ЗОБРАЖЕННЯХ ДЛЯ ЗАДАЧ COMPUTER VISION*

| Виконала  студент кафедри ІСТ ФІОТ, групи ІА-12: |  | Перевірив:  пос. Баран Д. Р. |
| --- | --- | --- |
| Яковенко Д. О. |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

Київ 2024

**І. Мета:**

Дослідити принципи та особливості підготовки даних, синтезу, навчання та застосування штучних нейронних мереж (Artificial Neural Networks) для практичних задач ідентифікації в технологіях Computer Vision

**ІІ. Завдання:**

*Реалізація проекту триває та спрямовано на збільшення функціональності*

*програмної компоненти. Лабораторія провідної ІТ-компанії реалізує масштабний проект розробки універсальної платформи з цифрової обробки зображень для задач Computer Vision. Платформа передбачає розташування back-end компоненти на власному хмарному сервері з наданням повноважень користувачам заздалегідь адаптованого front-end функціоналу універсальної платформи. Цим формується унікальна для потреб замовника ERP система з технологіями Computer Vision. Замовниками ресурсів платформи є:*

*державні та комерційні компанії, що розробляють медичне обладнання з діагностування захворювань за візуальною інформацією;*

*автоматизації аграрного бізнесу в аспекті обліку посівних територій за даними з БПЛА;*

*візуального контролю безпекових заходів на об’єктах критичної інфраструктури: аеропорти, торгівельно-розважальні центри, житлові комплекси тощо.*

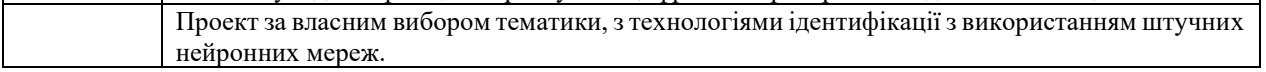
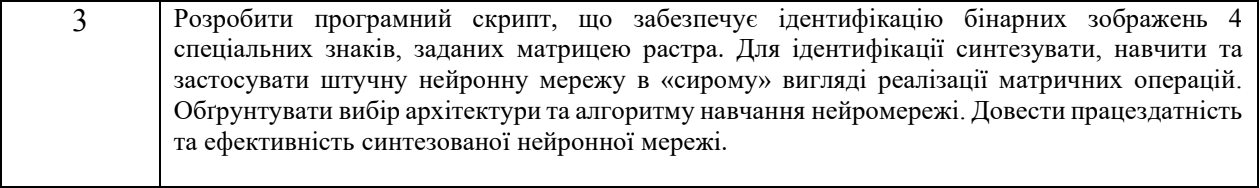
Розробити програмний скрипт мовою Python що реалізує обчислювальний алгоритм

ідентифікації об’єктів на цифрових зображеннях за технологіями штучних нейронних

мереж (Artificial Neural Networks): підготовка даних; конструювання нейромережі;

навчання штучної нейронної мережі; застосування нейромережі:

**ІІ рівень складності – максимально 9 балів. Відповідно до технічних умов, табл.2 додатку.**

**  
**

Оскільки мені була цікава реалізація завдання першого рівня – я його виконала також.

**ІІІ. Результати виконання лабораторної роботи.**

**3.1. Синтезована математична модель перетворень графічних об’єктів відповідно до індивідуального завдання.**

**Перший рівень:**

Відповідно до умов задачі синтезовано математичну модель для класифікації символів, використовуючи просту нейронну мережу. Задачею було розпізнати чотири спеціальних символи. Створена нейронна мережа ефективно навчилася класифікувати задані символи з високою точністю. Процес навчання показав зменшення втрат та збільшення точності з кожною епохою, що свідчить про ефективність реалізованої моделі та алгоритму навчання.

* Нейронна мережа складається з одного прихованого шару з 5 нейронами та вихідного шару з 4 нейронами. Функція активації - сигмоїдна функція.
* Графіки точності та втрат на кожній епосі демонструють процес навчання та зменшення помилок.

Додатково підраховуємо кількість співпадінь контрольних точок, що відповідають лісовим насадженням. Результат виводимо в консолі програми.

**Другий рівень:**

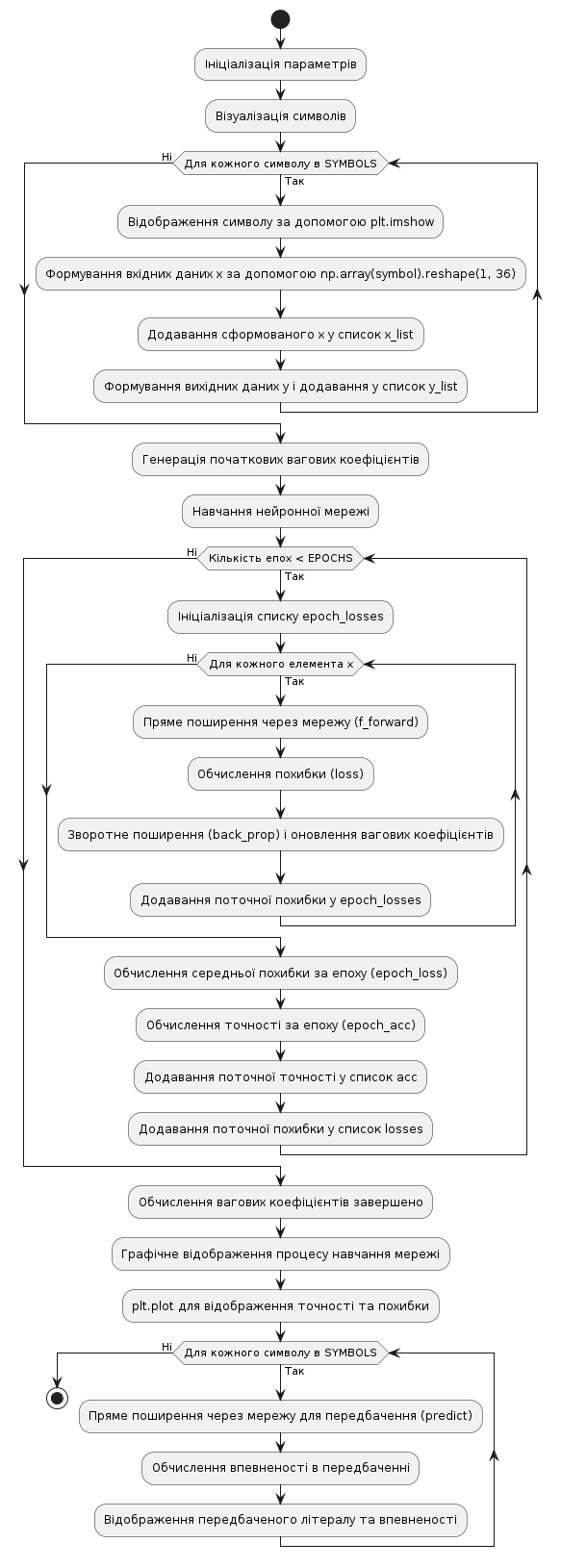
Відповідно до умов цієї задачі синтезовано математичну модель для виявлення стану очей за допомогою методу EAR (Eye Aspect Ratio) та алгоритму MediaPipe Face Mesh. Реалізовано систему, яка може розпізнавати закриті очі і активувати сигнал тривоги у разі, якщо очі залишаються закритими довше певного часу.

* Реалізовано можливість обробки відеофайлу або відеопотоку з веб-камери.
* Функція calculate\_ear() обчислює відношення висотних відстаней між ключовими точками ока до горизонтальної відстані. Це співвідношення є показником того, чи око відкрите чи закрите.
* Встановлено поріг THRESHOLD = 0.22 для визначення стану очей. Якщо EAR нижче цього порогу, вважається, що око закрите.
* Визначено максимальний час закритих очей, після якого активується сигнал тривоги.
* На зображенні відображаються контури очей, змінюючи свій колір залежно від стану очей: червоний для закритих очей і зелений для відкритих, а також виводиться текстове повідомлення.

**3.2. Блок схема алгоритму та її опис.**

**Перший рівень:**

Застосування синтезованих моделей здійснюється у порядку, що відображає суть алгоритму реалізації завдань лабораторної роботи.



| Рис.1. Блок-схема алгоритму програми для першого рівня. |
| --- |
|  |

Розглянемо даний алгоритм. Він починається з ініціалізації параметрів, таких як кількість епох, швидкість навчання і символи для розпізнавання. Далі виконується візуалізація символів, де кожен символ відображається за допомогою бібліотеки matplotlib. У процесі цього кожен символ перетворюється у вхідний вектор і додається до списку вхідних даних x, а відповідний вихідний вектор додається до списку вихідних даних y.

Після цього генеруються початкові вагові коефіцієнти для шарів нейронної мережі. Параметри w1 і w2 ініціалізуються випадковими значеннями. Далі розпочинається процес навчання нейронної мережі, який проходить у декілька етапів, відповідно до заданої кількості епох. Кожна епоха включає ітерації по всіх вхідних даних. У кожній ітерації виконується пряме поширення через мережу, обчислюється похибка, і виконується зворотне поширення для оновлення вагових коефіцієнтів. Поточна похибка додається до списку epoch\_losses, а після обробки всіх вхідних даних обчислюється середня похибка за епоху. Відповідно обчислюється точність за епоху, яка додається до списку acc, а середня похибка додається до списку losses.

Після завершення навчання відображаються графіки, що показують зміну точності та похибки протягом епох. Наступним кроком виконується ідентифікація символів. Для кожного символу виконується пряме поширення через мережу для передбачення. Обчислюється впевненість у передбаченні, і відображаються передбачений літерал та впевненість у передбаченні.

Отже, алгоритм охоплює всі етапи від підготовки даних до навчання нейронної мережі і перевірки результатів, забезпечуючи розпізнавання спеціальних символів на основі заданих вхідних даних.

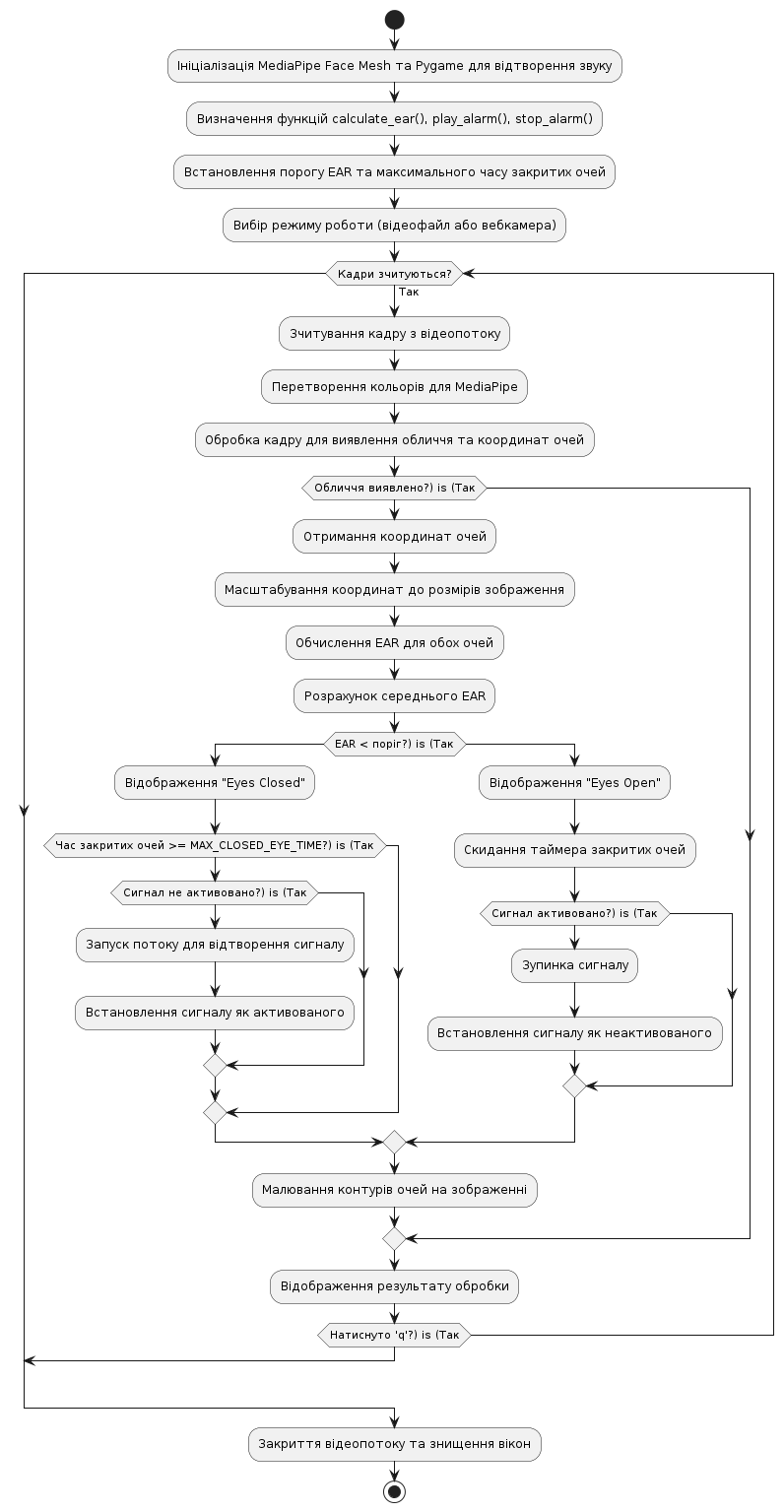
**Другий рівень:**

Алгоритм програми починається з ініціалізації **MediaPipe Face Mesh** для відстеження обличчя. Далі ініціалізується Pygame для відтворення звуків. Програма визначає функцію для обчислення коефіцієнта аспекта ока (EAR), яка розраховує EAR на основі координат шести точок навколо кожного ока. Встановлюється поріг EAR для визначення закритих очей, максимальний час закритих очей, щоб викликати звуковий сигнал, і шлях до аудіофайлу.

Наступним кроком є визначення функцій для відтворення і зупинки звуку за допомогою Pygame. Потім користувачеві пропонується вибрати режим роботи: обробка відеофайлу або використання вебкамери для відстеження напрямку погляду. Відповідно до вибору користувача відкривається відеофайл або вебкамера. Після цього починається основний цикл обробки кадрів. Програма зчитує кадр з відеопотоку і перетворює кольори для сумісності з MediaPipe. Обробляється кадр для виявлення обличчя і отримання координат основних точок обличчя. Якщо обличчя було виявлено, програма отримує координати шести точок навколо кожного ока і масштабує їх до розмірів зображення. Далі обчислюється EAR для обох очей і розраховується середнє значення EAR.

Далі ми визначаємо стан очей на основі середнього EAR і встановлюємо відповідний колір для контурів очей. Якщо очі закриті, програма перевіряє, чи закриті вони довше встановленого порогу часу (у нас встановлено 3 секунди). Якщо так, і сигнал ще не активовано, запускається окремий потік для відтворення звукового сигналу. Якщо користувач відкрив очі – програма припиняє відтворення звуку, якщо сигнал був активований.

Далі програма малює контури очей на зображенні і відображає результат обробки. Якщо користувач натискає клавішу 'q', програма завершує роботу, закриває відеопотік і знищує всі вікна OpenCV.



| Рис.2. Блок-схема алгоритму програми для другого рівня. |
| --- |
|  |

**3.3. Опис структури проекту програми в середовищі PyCharm.**

Для реалізації розробленого алгоритму мовою програмування Python з використанням можливостей інтегрованого середовища PyCharm сформовано проект.

Проект базується на лінійній бізнес-логіці функціонального програмування та має таку структуру.

|  |
| --- |
| Рис.3. Структура проекту. |

CVLab\_7 – головний каталог проекту

alarm.wav – звукова доріжка для файлу розпізнавання відкритих/закритих очей

eyeDetection.py – файл програмного коду лабораторної роботи (другий рівень)

neuralNetworks.py – файл програмного коду лабораторної роботи (перший рівень)

video.mov – відео для файлу з розпізнаванням відкритих/закритих очей при режимі 1;

**3.4. Результати роботи програми відповідно до завдання.**

Результатом роботи програми є програма, в якій при її запуску отримуємо такий результат:

**Перший рівень:**

|  |
| --- |
| Рис.4. Результат навчання при 2000 епохах.      Рис.5. Результат навчання при 2 епохах. |

Можна зробити висновок, що епохи впливають на процес навчання нейронної мережі. У даному випадку було використано спочатку дві тисячі, та потім дві епохи навчання. Кожна епоха представляє один повний прохід через весь набір даних для навчання. Під час кожної епохи нейронна мережа піддається тренуванню з використанням вхідних даних (x) та очікуваних вихідних даних (y). Після кожної ітерації у мережі оновлюються вагові коефіцієнти з використанням алгоритму зворотного поширення помилки. Цей процес триває доти, доки не буде завершено вказану кількість епох. Отже можна сказати, що їх кількість впливає на ефективність навчання моделі.

**Другий рівень:**

|  |
| --- |
| Рис.6. Результат роботи MediaPipe Face Mesh при виборі першого режиму.  Результат роботи MediaPipe Face Mesh при виборі другого режиму:  <https://drive.google.com/file/d/1Y8zXDbQs59jib76SMz5KUT035RaoWHNI/view?usp=sharing>  MediaPipe Face Mesh використовує методи машинного навчання, включаючи штучні нейронні мережі, для виявлення та трекінгу ключових точок обличчя в реальному часі. Це розширений алгоритм, який дозволяє відстежувати положення 468 ключових точок обличчя з високою точністю. MediaPipe Face Mesh пропонує високу точність визначення ключових точок обличчя, що робить його ефективним для різних застосувань, включаючи анімацію обличчя, розпізнавання емоцій та медичні аналізи. Однією з ключових переваг є здатність працювати в реальному часі, що ми можемо спостерігати при виборі другого режиму. |
|  |

**3.5. Програмний код.**

Програмний код послідовно реалізує алгоритми для виконання завдання другого рівня складності.

При цьому використано можливості Python бібліотек: openCV, numpy, sys, time, threading, MediaPipe та PyGame.

**Перший рівень:**

import numpy as np

import matplotlib.pyplot as plt

# Параметри

EPOCHS = 2

ALPHA = 0.1

SYMBOLS = ["?", "!", "%", "("]

QUESTION\_MARK = [0, 1, 1, 1, 1, 0,

1, 0, 0, 0, 0, 1,

0, 1, 0, 0, 1, 0,

0, 0, 0, 1, 0, 0,

0, 0, 0, 0, 0, 0,

0, 0, 0, 1, 0, 0]

EXCLAMATION\_MARK = [0, 0, 1, 1, 0, 0,

0, 0, 1, 1, 0, 0,

0, 0, 1, 1, 0, 0,

0, 0, 1, 1, 0, 0,

0, 0, 0, 0, 0, 0,

0, 0, 1, 1, 0, 0]

PERCENT\_SIGN = [1, 1, 0, 0, 0, 1,

1, 1, 0, 0, 1, 0,

0, 0, 0, 1, 0, 0,

0, 0, 1, 0, 0, 0,

0, 1, 0, 0, 1, 1,

1, 0, 0, 0, 1, 1]

LEFT\_PARENTHESIS = [0, 0, 0, 1, 0, 0,

0, 0, 1, 0, 0, 0,

0, 0, 1, 0, 0, 0,

0, 0, 1, 0, 0, 0,

0, 0, 1, 0, 0, 0,

0, 0, 0, 1, 0, 0]

# ----------------------------------- Вхідні дані DataSet ------------------------------------

def data\_x():

# Візуалізація символів

symbols = [QUESTION\_MARK, EXCLAMATION\_MARK, PERCENT\_SIGN, LEFT\_PARENTHESIS]

plt.figure(figsize=(10, 2))

for i, symbol in enumerate(symbols):

plt.subplot(1, 4, i+1)

plt.imshow(np.array(symbol).reshape(6, 6), cmap='viridis')

plt.show()

# Формування вхідних даних

x = [np.array(symbol).reshape(1, 36) for symbol in symbols]

return x

def data\_y():

# Формування вихідних даних

out\_dataset = [[1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 1, 0], [0, 0, 0, 1]]

y = np.array(out\_dataset)

return y

# ----------------------------------- Конструювання нейромережі ------------------------------------

def sigmoid(x):

return 1 / (1 + np.exp(-x))

def f\_forward(x, w1, w2):

# Прихований прошарок

z1 = x.dot(w1)

a1 = sigmoid(z1)

# Вихідний прошарок

z2 = a1.dot(w2)

a2 = sigmoid(z2)

return a2

def generate\_wt(x, y):

return np.random.randn(x, y)

def loss(out, Y):

return np.sum(np.square(out - Y)) / len(y)

def back\_prop(x, y, w1, w2, alpha):

# Прихований прошарок

z1 = x.dot(w1)

a1 = sigmoid(z1)

# Вихідний прошарок

z2 = a1.dot(w2)

a2 = sigmoid(z2)

# Похибка на вихідному прошарку

d2 = a2 - y

d1 = np.multiply(w2.dot(d2.T).T, np.multiply(a1, 1 - a1))

# Градієнт для w1 і w2

w1\_adj = x.T.dot(d1)

w2\_adj = a1.T.dot(d2)

# Оновлення параметрів

w1 -= alpha \* w1\_adj

w2 -= alpha \* w2\_adj

return w1, w2

def train(x, Y, w1, w2, alpha, epochs):

acc = []

losses = []

for epoch in range(epochs):

epoch\_losses = []

for i in range(len(x)):

out = f\_forward(x[i], w1, w2)

epoch\_losses.append(loss(out, Y[i]))

w1, w2 = back\_prop(x[i], y[i], w1, w2, alpha)

epoch\_loss = sum(epoch\_losses) / len(x)

epoch\_acc = (1 - epoch\_loss) \* 100

print(f"Epoch {epoch + 1}: Accuracy = {epoch\_acc:.2f}%")

acc.append(epoch\_acc)

losses.append(epoch\_loss)

return w1, w2, acc, losses

def predict(x, w1, w2):

output = f\_forward(x, w1, w2)

prediction = np.argmax(output)

confidence = output[0][prediction]

return prediction, confidence

# ----------------------------------- Основний процес ---------------------------------------

x = data\_x()

y = data\_y()

# Генерація початкових вагових коефіціентів

w1 = generate\_wt(36, 5)

w2 = generate\_wt(5, 4)

print("============== Вхідні параметри DataSet =============")

for i, xi in enumerate(x):

print(f"x{i} \n", xi)

print("==================== Вихідна частина DataSet ====================")

for i, yi in enumerate(y):

print(f"y{i} \n", yi)

print("===================== Навчання нейронної мережі =====================")

trained\_wts = train(x, y, w1, w2, ALPHA, EPOCHS)

print("==================== Обчислені вагові коефіціенти ====================")

print("w1 \n", trained\_wts[0])

print("w2 \n", trained\_wts[1])

# Графічне відображення процесу навчання мережі

plt.plot(trained\_wts[2])

plt.ylabel('Accuracy')

plt.xlabel('Epoch')

plt.title('Accuracy over Epochs')

plt.show()

plt.plot(trained\_wts[3])

plt.ylabel('Loss')

plt.xlabel('Epoch')

plt.title('Loss over Epochs')

plt.show()

# Ідентифікація літералів / передбачення

for i, symbol in enumerate(SYMBOLS):

print(f'Вхідні параметри відповідають літералу "{symbol}"')

print('Результат ідентифікації:')

prediction, confidence = predict(x[i], trained\_wts[0], trained\_wts[1])

print(f'Передбачений літерал: "{SYMBOLS[prediction]}" з впевненістю {confidence \* 100:.2f}%\n')

**Другий рівень:**

import cv2

import mediapipe as mp

import numpy as np

import sys

import time

import threading

import pygame

# Ініціалізація MediaPipe Face Mesh

mp\_face\_mesh = mp.solutions.face\_mesh

face\_mesh = mp\_face\_mesh.FaceMesh(static\_image\_mode=False, max\_num\_faces=1, min\_detection\_confidence=0.5,

min\_tracking\_confidence=0.5)

# Ініціалізація Pygame для відтворення звуку

pygame.mixer.init()

# Функція для обчислення EYE Aspect Ratio (EAR)

def calculate\_ear(eye\_landmarks):

A = np.linalg.norm(eye\_landmarks[1] - eye\_landmarks[5])

B = np.linalg.norm(eye\_landmarks[2] - eye\_landmarks[4])

C = np.linalg.norm(eye\_landmarks[0] - eye\_landmarks[3])

ear = (A + B) / (2.0 \* C)

return ear

# Поріг EAR для визначення закритих очей

EAR\_THRESHOLD = 0.23

# Максимальний час закритих очей, щоб викликати звук (у секундах)

MAX\_CLOSED\_EYE\_TIME = 3

# Шлях до аудіофайлу

AUDIO\_FILE\_PATH = 'alarm.wav'

# Функція відтворення звуку

def play\_alarm():

pygame.mixer.music.load(AUDIO\_FILE\_PATH)

pygame.mixer.music.play(-1) # Повторювати звук поки не буде зупинено

# Функція зупинки звуку

def stop\_alarm():

pygame.mixer.music.stop()

# Меню вибору режиму

print("Select mode:")

print("1: Process video file")

print("2: Use webcam to track gaze direction")

mode = input("Enter 1 or 2: ")

if mode == '1':

cap = cv2.VideoCapture('video.mov')

elif mode == '2':

cap = cv2.VideoCapture(0)

else:

print("Invalid selection")

sys.exit()

start\_time = None

alarm\_thread = None

alarm\_triggered = False

while cap.isOpened():

success, image = cap.read()

if not success:

break

# Перетворення кольорів для MediaPipe

image\_rgb = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)

results = face\_mesh.process(image\_rgb)

if results.multi\_face\_landmarks:

for face\_landmarks in results.multi\_face\_landmarks:

landmarks = face\_landmarks.landmark

# Отримання координат очей

left\_eye\_landmarks = np.array([(landmarks[i].x, landmarks[i].y) for i in [362, 385, 387, 263, 373, 380]])

right\_eye\_landmarks = np.array([(landmarks[i].x, landmarks[i].y) for i in [33, 160, 158, 133, 153, 144]])

# Масштабування координат до розмірів зображення

h, w, \_ = image.shape

left\_eye\_landmarks = np.array([(int(x \* w), int(y \* h)) for x, y in left\_eye\_landmarks])

right\_eye\_landmarks = np.array([(int(x \* w), int(y \* h)) for x, y in right\_eye\_landmarks])

# Обчислення EAR для обох очей

left\_ear = calculate\_ear(left\_eye\_landmarks)

right\_ear = calculate\_ear(right\_eye\_landmarks)

# Середнє значення EAR

ear = (left\_ear + right\_ear) / 2.0

# Визначення стану очей і вибір кольору для контурів

if ear < EAR\_THRESHOLD:

eye\_color = (0, 0, 255) # Червоний

cv2.putText(image, 'Eyes Closed', (50, 50), cv2.FONT\_HERSHEY\_SIMPLEX, 1, (0, 0, 255), 2, cv2.LINE\_AA)

if start\_time is None:

start\_time = time.time()

else:

elapsed\_time = time.time() - start\_time

if elapsed\_time >= MAX\_CLOSED\_EYE\_TIME and not alarm\_triggered:

alarm\_thread = threading.Thread(target=play\_alarm)

alarm\_thread.start()

alarm\_triggered = True

else:

eye\_color = (0, 255, 0) # Зелений

cv2.putText(image, 'Eyes Open', (50, 50), cv2.FONT\_HERSHEY\_SIMPLEX, 1, (0, 255, 0), 2, cv2.LINE\_AA)

start\_time = None

if alarm\_triggered:

stop\_alarm()

alarm\_triggered = False

# Малювання контурів очей

cv2.polylines(image, [left\_eye\_landmarks], isClosed=True, color=eye\_color, thickness=2)

cv2.polylines(image, [right\_eye\_landmarks], isClosed=True, color=eye\_color, thickness=2)

# Відображення результату

cv2.imshow('Frame', image)

if cv2.waitKey(10) & 0xFF == ord('q'):

break

cap.release()

cv2.destroyAllWindows()

**3.6. Аналіз результатів відлагодження та верифікації результатів роботи програми.**

Результати відлагодження та тестування довели працездатність розробленого коду.

Верифікація функціоналу програмного коду, порівняння отриманих результатів з технічними умовами завдання на лабораторну роботу доводять, що завдання виконано у повному обсязі.

**IV. Висновки.**

У ході виконання лабораторної роботи я освоїла базові принципи роботи нейронної мережі для класифікації символів. Візуалізація процесу навчання допомагає краще зрозуміти, як працюють нейронні мережі.

Якщо говорити про систему моніторингу стану очей з використанням технологій обробки зображень та звукових сигналів тривоги – вона може бути застосована у багатьох сферах. Система може бути інтегрована в автомобілі для моніторингу стану водія. Виявлення закритих очей або тривалої відсутності зорової уваги активує сигнал тривоги, що попередить водія про можливість засинання за кермом, і це значно знижує ризик аварій. Система може використовуватися для моніторингу стану пацієнтів, наприклад, під час тривалих процедур або реабілітації, виявлення стану уваги студентів під час онлайн-навчання. Система може нагадати студентам про необхідність зробити перерву або зосередитися. Також можна моніторити увагу пілотів та операторів безпілотних літальних апаратів, де втрата концентрації може призвести до катастрофічних наслідків. Загалом, система може знайти застосування в будь-якій сфері, де важлива постійна увага та концентрація людини. Вона забезпечує додатковий рівень безпеки та допомагає запобігти негативним наслідкам втрати концентрації або засинання.

Виконала: студент Яковенко Д.О.