**МІНІСТЕРСТВО ОСВІТИ і НАУКИ УКРАЇНИ**

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ

імені ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра системного програмування і спеціалізованих комп’ютерних систем

**Лабораторна робота №1**  
**з дисципліни «Штучні нейронні мережі»**

Виконали:

студенти групи КВ-83

Кубай Олег

Пащенко Антон

Мельник Юрій

та студент групи КВ-81

Кравчук Віктор

Київ – 2022

Завдання

1. Сформувати навчальну вибірку для навчання згорткової нейронної мережі (ЗНМ), призначеної для виділення на зображенні обличчя людини ключових точок, що використовуються для розпізнавання емоцій. Навчальна вибірка має складатись із навчальних прикладів, що представляють собою фотографії зображень обличчя з нанесеними ключовими точками. Обсяг навчальної вибірки повинен перевищувати 50 прикладів. Для формування можливо використовувати вільнодоступні бази даних, однак не менше ніж 10 прикладів мають бути сформовані самостійно.

2. Розробити програмне забезпечення, призначене для перетворення сформованих прикладів до виду придатному для подачі на вхід ЗНМ.

3. Розробити програмне забезпечення (ПЗ) для реалізації ЗНМ, призначеної для визначення координат ключових точок на фронтальному зображенні обличчя. Передбачити можливість відображення отриманих ключових точок на піддослідних фотографіях.

4. Провести комп’ютерні експерименти спрямовані на верифікацію розробленого ПЗ. ПЗ вважається верифікованим, якщо точність розпізнавання тренувальних прикладів >0,8, а точність розпізнавання тестових прикладів >0,7.

5. Оформити звіт в якому відобразити:

a. Прізвище, ініціали та номер групи виконавця.

b. Назву та завдання лабораторної роботи.

c. Інструкцію для використання розробленого ПЗ.

d. Скріншоти, що демонструють використання розробленого ПЗ.

e. Висновки.

Використаний датасет: <https://www.kaggle.com/datasets/astraszab/facial-expression-dataset-image-folders-fer2013>

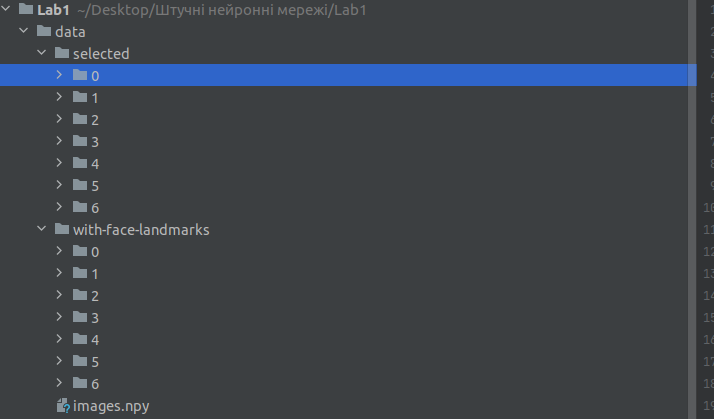
Було відібрано по 20 зображень з кожної категорії (Всього 120)

Для визначення ключових точок на обличі людини які будуть використані для тренування нейронної мережі було використано готову нейронну мережу знайдену на github.com. Код програми для визначення точок:

**preprocessor.py**

# This file is used to extract face landmarks from images in data/selected folder  
# Dumps nupmy array in binary and images with face landmarks  
# array structure: [[imageTypeNumber, grayImageArray, imageLandmarksArray]]  
  
import cv2  
import dlib  
import numpy as np  
import PIL.Image  
import os  
  
IMAGES\_BASE\_DIR = 'data/selected'  
IMAGES\_DUMP\_DIR = 'data/with-face-landmarks'  
  
detector = dlib.get\_frontal\_face\_detector()  
predictor = dlib.shape\_predictor('./models/shape\_predictor\_68\_face\_landmarks.dat')  
  
imagesData = []  
for imageTypeName in os.listdir(IMAGES\_BASE\_DIR):  
 for imageFileName in os.listdir(os.path.join(IMAGES\_BASE\_DIR, imageTypeName)):  
 imageFullPath = os.path.join(IMAGES\_BASE\_DIR, imageTypeName, imageFileName)  
 image = PIL.Image.open(imageFullPath)  
 np\_image = np.array(image)  
 rects = detector(np\_image, 1)  
 if len(rects) < 1:  
 print("Cannot find face on image " + imageFullPath)  
 continue  
 # There is only one face in each image  
 rect = rects[0]  
 # Get the landmark points  
 shape = predictor(np\_image, rect)  
 # Convert it to the NumPy Array  
 shape\_np = np.zeros((68, 2), dtype="int")  
 for i in range(0, 68):  
 shape\_np[i] = (shape.part(i).x, shape.part(i).y)  
 shape = shape\_np  
  
 np\_image\_copy = np.copy(np\_image)  
 # Draw landmarks and save image  
 for i, (x, y) in enumerate(shape):  
 # Draw the circle to mark the keypoint  
 cv2.circle(np\_image\_copy, (x, y), 1, (0, 0, 255), -1)  
  
 imageWithLandmarks = PIL.Image.fromarray(np\_image\_copy)  
 imageDir = os.path.join(IMAGES\_DUMP\_DIR, imageTypeName)  
 os.makedirs(imageDir, exist\_ok=True)  
 imageWithLandmarks.save(os.path.join(imageDir, imageFileName))  
  
 # Save images to common binary  
 joined = np.array([int(imageTypeName), np\_image, shape\_np])  
 imagesData.append(joined)  
  
np\_imagesData = np.array(imagesData)  
np.save('data/images.npy', np\_imagesData)

Програма загружає зображення з папки data/selected в якій вони посортовані відповідно до класів, знаходить ключові точки і зберігає результат в бінарний файл (data/images.py) який може в подальшому бути використані для тренування нейронної мережі. Також вона зберігає зображення з нанесеними ключовими точками. Папка data після виконання програми:



Приклад зображення з нанесеними ключовими точками (68 точок):



Код програми призначеної для тренування нейронної мережі:

**lab1\_image2points.py**:

import tensorflow as tfimport numpy as npimport cv2import PIL.Imageimport osWITH\_EVALUATED\_LANDMARKS\_BASE\_PATH = 'data/with-evaluated-landmarks'def pairwise(np\_array): *"""s -> (s0, s1), (s2, s3), (s4, s5), ..."""*return np\_array.reshape(len(np\_array[0]) // 2, 2)def load\_data(file\_name='data/images.npy'): image\_data = np.load(file\_name, allow\_pickle=True) np.random.shuffle(image\_data) return image\_datadef split\_data(image\_data): images = [] landmarks = [] for (imageType, image, points) in image\_data: images.append(image) landmarks.append(points.flatten()) return np.array(images), np.array(landmarks)def split\_train\_test(images, landmarks): images\_20percent\_size = len(images) // 5 landmarks\_20percent\_size = len(landmarks) // 5 test\_images = images[:images\_20percent\_size] test\_landmarks = landmarks[:landmarks\_20percent\_size] train\_images = images[images\_20percent\_size:] train\_landmarks = landmarks[landmarks\_20percent\_size:] return train\_images, train\_landmarks, test\_images, test\_landmarksdef create\_model(output\_n=136): model = tf.keras.models.Sequential([ tf.keras.layers.InputLayer(input\_shape=[48, 48, 1]), tf.keras.layers.Conv2D(filters=32, kernel\_size=[5, 5], padding='same', use\_bias=False), tf.keras.layers.LeakyReLU(alpha=.1), tf.keras.layers.BatchNormalization(), tf.keras.layers.Conv2D(filters=32, kernel\_size=[5, 5], padding='same', use\_bias=False), tf.keras.layers.LeakyReLU(alpha=.1), tf.keras.layers.BatchNormalization(), tf.keras.layers.MaxPool2D(pool\_size=[2, 2]), tf.keras.layers.Conv2D(filters=64, kernel\_size=[3, 3], padding='same', use\_bias=False), tf.keras.layers.LeakyReLU(alpha=.1), tf.keras.layers.BatchNormalization(), tf.keras.layers.Conv2D(filters=64, kernel\_size=[3, 3], padding='same', use\_bias=False), tf.keras.layers.LeakyReLU(alpha=.1), tf.keras.layers.BatchNormalization(), tf.keras.layers.MaxPool2D(pool\_size=[2, 2]), tf.keras.layers.Conv2D(filters=128, kernel\_size=[3, 3], padding='same', use\_bias=False), tf.keras.layers.LeakyReLU(alpha=.1), tf.keras.layers.BatchNormalization(), tf.keras.layers.Conv2D(filters=128, kernel\_size=[3, 3], padding='same', use\_bias=False), tf.keras.layers.LeakyReLU(alpha=.1), tf.keras.layers.BatchNormalization(), tf.keras.layers.MaxPool2D(pool\_size=[2, 2]), tf.keras.layers.Conv2D(filters=256, kernel\_size=[3, 3], padding='same', use\_bias=False), tf.keras.layers.LeakyReLU(alpha=.1), tf.keras.layers.BatchNormalization(), tf.keras.layers.Conv2D(filters=256, kernel\_size=[3, 3], padding='same', use\_bias=False), tf.keras.layers.LeakyReLU(alpha=.1), tf.keras.layers.BatchNormalization(), tf.keras.layers.MaxPool2D(pool\_size=[2, 2]), tf.keras.layers.Conv2D(filters=512, kernel\_size=[3, 3], padding='same', use\_bias=False), tf.keras.layers.LeakyReLU(alpha=.1), tf.keras.layers.BatchNormalization(), tf.keras.layers.Conv2D(filters=512, kernel\_size=[3, 3], padding='same', use\_bias=False), tf.keras.layers.LeakyReLU(alpha=.1), tf.keras.layers.BatchNormalization(), tf.keras.layers.Flatten(), tf.keras.layers.Dense(units=512, activation='relu'), tf.keras.layers.Dropout(.1), tf.keras.layers.Dense(units=output\_n), ]) model.compile(optimizer=tf.keras.optimizers.Adam(learning\_rate=1e-4), loss="mean\_squared\_error", metrics=['mae', 'accuracy']) return modelnp\_image\_data = load\_data()np\_images, np\_landmarks = split\_data(np\_image\_data)np\_train\_images, np\_train\_landmarks, np\_test\_images, np\_test\_landmarks = split\_train\_test(np\_images, np\_landmarks)model\_136 = create\_model(136)# model\_136 = tf.keras.models.load\_model('models/lab1/lab1\_3.keras')model\_136.fit(np\_train\_images, np\_train\_landmarks, validation\_split=.1, batch\_size=64, epochs=1000)print("Evaluating on test data:")model\_136.evaluate(np\_test\_images, np\_test\_landmarks)model\_136.save('models/lab1/lab1\_4.keras')print("Evaluating landmarks on images...")image\_counter = 0for (image\_type, image, points) in np\_image\_data: image\_expanded = np.expand\_dims(image, axis=0) predicted\_shape = model\_136.predict(image\_expanded) predicted\_shape\_int = predicted\_shape.astype(int) pairwise\_predicted\_shape = pairwise(predicted\_shape\_int) for x, y in pairwise\_predicted\_shape: cv2.circle(image, (x, y), 1, (0, 0, 255), -1) image\_with\_landmarks = PIL.Image.fromarray(image) image\_dir = os.path.join(WITH\_EVALUATED\_LANDMARKS\_BASE\_PATH, str(image\_type)) os.makedirs(image\_dir, exist\_ok=True) image\_with\_landmarks.save(os.path.join(image\_dir, str(image\_counter) + '.png')) image\_counter += 1

Програма загружає попередньо оброблені дані в бінарному форматі, перемішує їх, розбиває на тренувальні і тестові, тренує нейронну мережу та розставляє ключові точки на зображеннях з використанням результатів роботи тренованої мережі. Приклад зображення з тестового набору з розставленими ключовими точками (з папки data/with-face-landmarks):



Точність для тренувальних даних склала 95.3%, для тестових 84%.

Оскільки на виході з нейронної мережі отримані дані типу float, при оцінці точності вони заокруглювались.

Висновок:

Під час даної лаборатоної роботи було розроблено дві програми, перша – для знаходження ключових точок на зображенні (з використанням знайденої на просторах інтернету нейронної мережі) і вигрузки результатів у бінарному форматі, друга – для тренування нейронної мережі нанесення результатів (ключових точок) на зображення. Була використана згорткова нейронна мережа, такий вибір можна пояснити тим, що більшість нейронів у мережі з повним з’єднанням (Dense network) зазвичай не впливають на результат, але значно сповільнюють швидкість навчання.