КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА

Факультет комп'ютерних наук та кібернетики

Кафедра теоретичної кібернетики

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА
зі спеціальності 122 «Комп'ютерні науки»
на тему:

Виявлення і дослідження емоційних станів обличчя людини

Студент 4-го курсу
Дмитерко Олександр Богданович

_____
(підпис)

Науковий керівник:
професор Крак Юрій Васильович

_____ (підпис)

Робота заслухана на засіданні кафедри теоретичної кібернетики та рекомендована до захисту в ЕК, протокол № …… від..... 2021р.

Завідувач кафедри Крак Ю.В.

Київ – 2021

Реферат

Робота складається зі вступу, 4 розділів, висновків, списку використаних джерел 12 книг 4. Робота містить 38 рисунків. Загальний обсяг становить 57 сторінок, основний текст роботи викладено на 21 сторінках.

ключові слова: Дослідження емоцій на обличчі людини, нейромережа для виявлення емоцій на обличчі людини.

Це дослідження для вивчення і розуміння, що таке нейромережа і як вона працює саме з навколишнім світом, як її навчати для певних задач, в випадку виявлення обличчя людини і визначити емоційний стан людини, цей продукт можна використовувати в багатьох сферах діяльності щоб за здалегідь взнати яка емоція в людини, коли вона заходиться в громадському транспорті, а бо в громадських місцях, щоб за здалегідь усунути трагедію в терористичних цілях проявлення агресії до людей навколо.

Мета роботи: зрозуміти як працюють алгоритми комп'ютерного зору для виявлення обличь на зображенні, навчитися аналізувати емоції людей на виявлених обличчях, розробити програму, яка буде виводити дані про емоції на екран.

Метою роботи є вибрати підходящий набір даних для тренування нейронної мережі, реалізації нейронної мережі та за створення за її допомогою натренованої моделі для виявлення емоцій на обличчі людини, створення програми, яка буде обробляти зображення з веб-камери і виявляти емоції на обличчі людей.

Методи розроблення: вивчення та дослідження роботи нейронної мережі та алгоритмом навчання нейронної мережі, застосування метода Віоли-Джоунса, також примітити OpenCV, Deep Lerning, TensorFlow, машинне навчання, нейромережа.

Результат роботи: робоча програма для виявлення емоцій та програма для тренування нейронної мережі для виявлення обличь по методам Хаара, Віоли-Джоунса, OpenCV та застосування технологій Deep Lerning, TensorFlow, машинне навчання, нейромережа.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

БЗ — база знань;

МВД — метод Віоли Джоунса;

НМ — нейронна мережа;

ТК — технічна кібернетика;

МС — мікро сервіс;

DL — deep lerning;

ЗН — загорткова нейромережа;

РА — реалізація алгоритму;

ВС — відкрита система;

ЄІП — єдиний інформаційний простір;

БД — база даних;

ІБД — інформаційна база даних;

ІС — інформаційна система;

МЗ — модель зображень;

ШНМ — Штучні нейронні мережі;

МД — масив даних;

МН — машинне навчання;

КМН — класичне машинне навчання;

ЗНМ — загорткова нейронна мережа;

СКО — середньоквадратична помилка.

ВСТУП

Виявлення і дослідження емоційних станів обличчя людини

Цілком імовірно, головними завданнями сьогодні є бачення ПК, яке має на меті зробити машину, яка виглядає як людина, комп’ютеризувати свої вправи. Ілюстрацією такого призначення є визнання людських почуттів в реальному часі. Це доручення дуже важливо, оскільки воно виявляє своє застосування у діловій та соціальній частинах людського існування.

Нейробіологи, які були зацікавлені у зв'язку між почуттями та тривожними циклами людини, спочатку надихнулись цим дослідженням. Основна робота щодо сприйняття почуттів була створена Екманом та Фрізаном, які продемонстрували зв'язок між людськими почуттями та конкретним розташуванням м'язових звужень, які виявляються на обличчі як вигляд, і відповідно є зоровою частиною обличчя.

До цього часу існує безліч технік сприйняття почуттів. Цілком можливо, найбільш відомими методами є нейронні організації та, зокрема, згорткові нейронні організації. Через те, що існує безліч альтернатив способу сформувати конкретну нейронну організацію, фахівці продовжують працювати тут, отримуючи дедалі більшу кількість нових результатів. З огляду на доречність та широкий обсяг можливих результатів, нейронні організації були обрані як спосіб піклування про проблеми.

Отже, метою цієї роботи є визначити людські почуття від відео передачі, а предмету - подумати про природу методів розпізнавання почуттів. Сенс роботи - зібрати ідеальну модель.

В першому розділі, розповім алгоритм та виконання роботи.

У другому розділі роботи, провів аналіз та огляд математичних основ для основних побудов нейромереж, а також методів які використовувались для написання та покращення роботи нейромереж для виявлення емоцій на обличчі людини.

Третій розділ присвячено процесу побудови моделей. Наведено основі етапи, асаме: збір даних про нейронну мережу deep learning, їхнє навчання, тестування та алгоритму їхнього навчання. Розробка моделей здійснювалась за допомогою бібліотек: CNN, TensorFlow та її розширення Keras із використанням, за допомогою методів Хаара, та методів Віола-Джоунса мови програмування Python у середовищі розробки PyCharm.

В четвертому розділі доповнення до програм забезпечення для роботи, ознаки методів, та описи допоміжних стиків.

РОЗДІЛ 1
АЛГОРИТМ ВИКОНАННЯ ТА ПОКРАШЕННЯ ОБРОБКИ КОДУ

1.1 Метод Хаара

У кожного метода є основа, без чого метод не зміг би нормально працювати або його б не було, а над наступними основами працювала інша частина. (Рисунок 1.1.1)

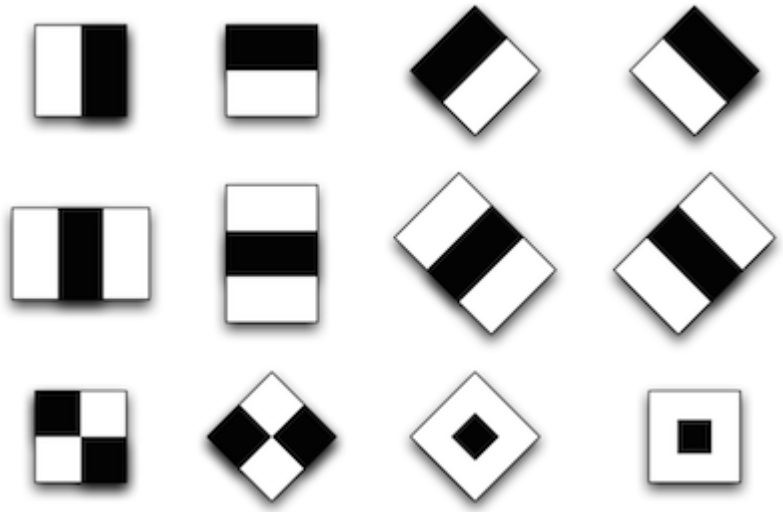

Рисунок 1.1.1

У першій формі розрахунку Віолі-Джонса використовувались просто таблиці без революції, і для обчислення якостей певна кількість яскравостей пікселей які знаходились поруч віднімалося від суми яскравостей різних імітацій. Протягом часу, витраченого на сприяння процедурі, були запропоновані елементи з поворотами до 45 градусів та відхиленими конструкціями. Так само, замість розробки стандартних планів, пропонувалося зарахувати порівняння покриття певної ваги та вартості знаків до оцінки як зважену кількість пікселів різних регіонів:

$$feature = \sum_{i \in I=1,\ldots,N} w_i \cdot RectSum(r_i)$$

Було обрано примітиви Хаара для того, щоб уникнути обходу піксельного представлення з забезпеченням швидкості обчислювального признаку, із двох пікселей багато не добитись в той час коли із двох ознак Хаара перший каскад методу по розпізнаванню обличчя, в якого вже є обґрунтування про інтерпретацію на рисунку 1.1.2:

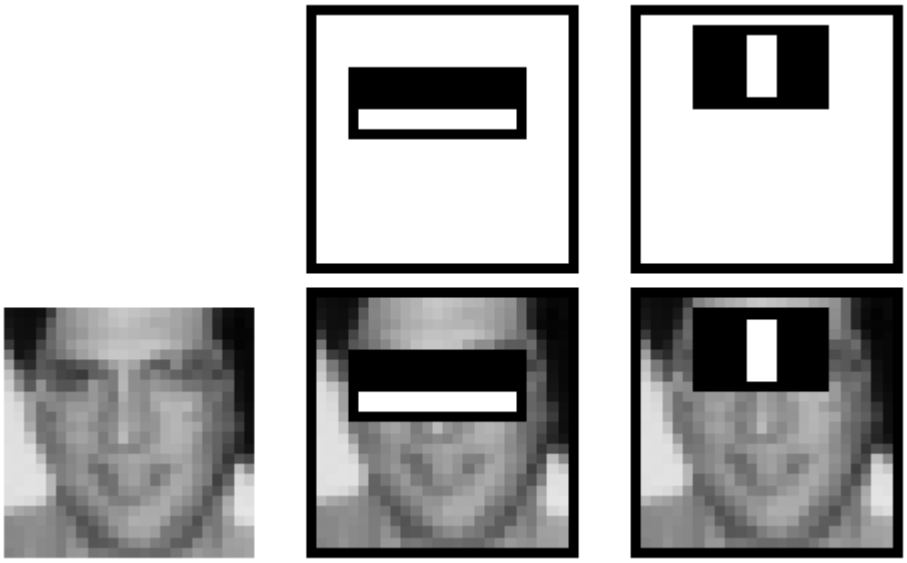

Рисунок 1.1.2

Визначення ознаки та отримання значення пікселя залишиться O(1): в області кожного із значень скомбінувати можна на 4 значення інтегрального призначення (SAT — Summed Area Table) яка в свою чергу відтворюватиме значення всього один раз для всього значення O(n), де n — значення пікселя в ілюстрації де використовуються наступна формула:

$$SAT(x,y) = SAT(x,y-1) + SAT(x-1,y) + I(x,y) - SAT(x-1,y-1)$$

$$SAT(-1,y) = SAT(x,-1) = SAT(-1,-1) = 0$$

Метод Хаара використовували для пошуку номерних знаків, оскільки в роботі з OpenCV конфліктів в них не виника, а навпаки прискорює роботу пошуку, тому винайшли фрейм як iOS в тому числі с симулятором який підтримував всю існуючу архітектуру. Функція

знаходження виразів а також присутня на Андроїдах під класом cv::CascadeClassifier або detectMultiScale.

1.2 Встановлення програмного забезпечення

Встановлення самої IDE як PyCharm і бібліотек та стиків до неї не тяжка справа (pip install), в наш час все можна скачати через інтернет, а в самої IDE де консоль для встановлення необхідних моделей далі про це розповім далі.

1.2.1 Встановлення PyCharm

Я вибрав мову програмування Python, для цього я встановив PyCharm - це інтегроване середовище для прогресу написання на мові програмування Python. Надає інструменти для перевірки коду, графічного інтерфейсу, пробний пристрій, а також підтримує просування веб-сайтів на Django. PyCharm створювався JetBrains в залежності від IntelliJ Thought.

Його можливості:

- Підсвічування синтезу коду його помилок.
- Легка навігація в написаному коді, та проекті, відображення структури проекту.
- Ре факторинг: перейменування, відокремлення техніки, введення змінної, введення стійкого, підвищення та збиття стратегії.
- Пристрої веб-просування з використанням структури Django
- Відбудований помічник для Python
- Базові пристрої для модульного тестування
- Поліпшення використання Google Applicotion Motor
- Підтримка керування варіантами фрейм ворку: зазвичай інтерфейс для Snconsistent, Github, Disrupt, Perforce nf CVS із підтримкою списку змін та консолідації.

Для встановлення такої чудової платформи:

1. Перейдемо на сайт його образу (ISO) https://www.jetbrains.com/pycharm/
2. Натиснемо на кнопку скачати (Download) нас перекине на вибірку версії PyCharm.
3. Виберемо під себе версію яка нам підходить, Персональна – більш поглиблена версія з повним набором функцій. Або спільну (Community) вона безкоштовна, с базовими можливостями.
4. Коли ми скачали обрану нами версію, необхідно її запустити та натиснути кнопку «Next»
5. Встановлюймо шлях куди буде встановлено PyCharm наприклад C:\Program Files\JetBrains\PyCharm Community Edition 2021.1
6. Підтверджуючі всі пункти жмемо «Next»
7. Дамо ім’я нашому PyCharm. наприклад JetBrains
8. Ждемо доки завершиться установка
9. Після завершення тиснемо на галочку і «Finish» (рисунок 1.2.1)

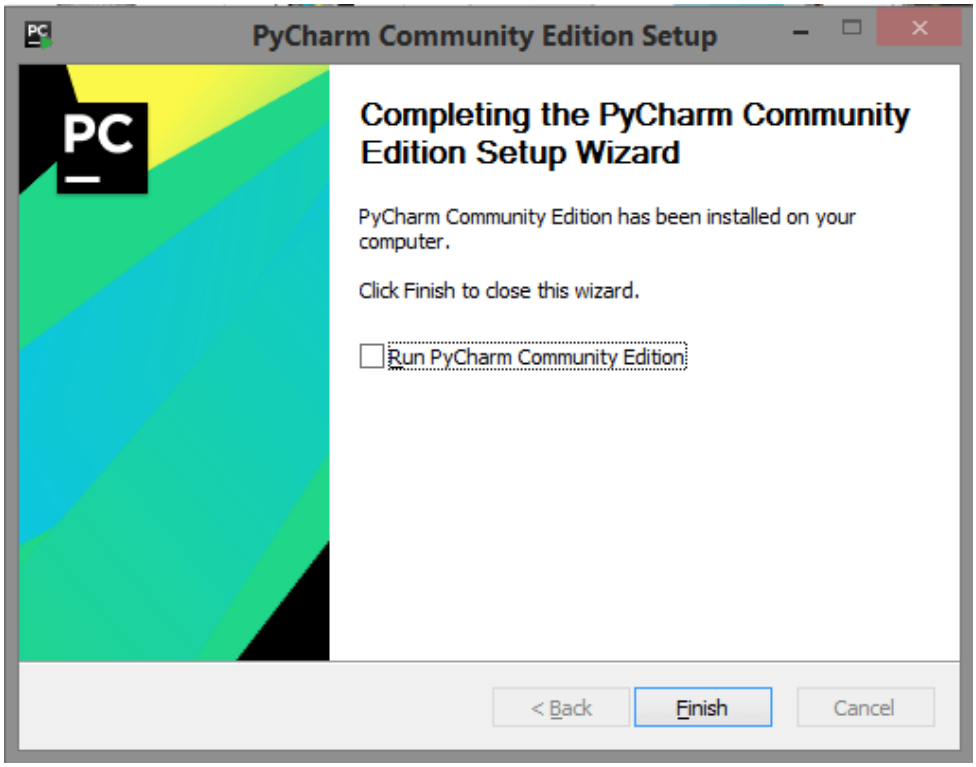

Рисунок 1.2.1

1.3 Структура програмної реалізації

Структура програмної реалізації дипломного проекту показана на рисунку 1.3.1:

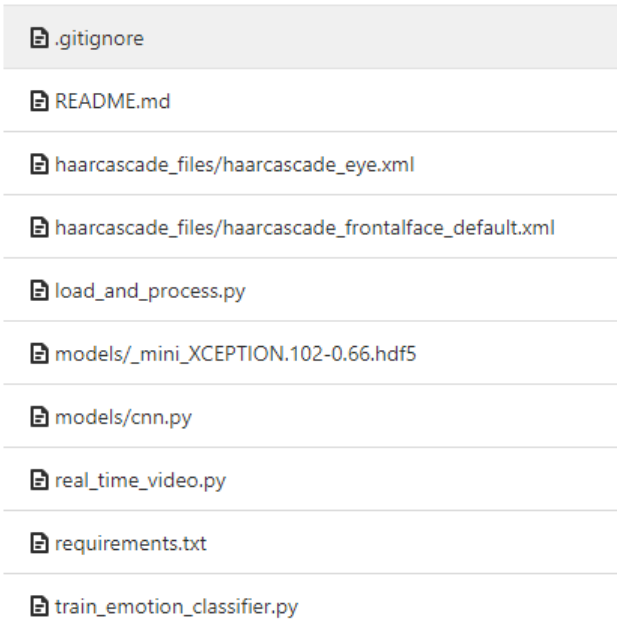

Рисунок 1.3.1

1. Readmi.md – коротка інструкція до завантаженої моделі

2. В директорії haarcascade_files містяться різні мета-дані моделі в .xml форматі, в роботі вони безпосередньо не використовуються, але необхідні для роботи програми в цілому.
3. В директорії models ми маємо два файли – реалізацію архітектуру самої нейромережи на Python в cnn.py та збережені ваги (параметри) моделі в файлі з розширенням .hdf5
4. .gitignore – файли прописані тут не будуть записуватися в Git та залишаться локальними при коміті.
5. load_and_process.py – завантаження та препроцесінг даних.
6. real_time_video.py – скріпт, що запускає модель для роботи у режимі трансляції за допомогою циклу while (True)
7. requirements.txt – залежності програми від сторонніх модулів
8. train_emotion_classifier.py – скріпт для тренування нейромереж

Розглянемо детальніше роботу коду по файлам нижче.

1.3.1 cnn.py

```
from keras.layers import Activation, Convolution2D, Dropout, Conv2D
from keras.layers import AveragePooling2D, BatchNormalization
from keras.layers import GlobalAveragePooling2D
from keras.models import Sequential
from keras.layers import Flatten
from keras.models import Model
from keras.layers import Input
from keras.layers import MaxPooling2D
from keras.layers import SeparableConv2D
from keras import layers
from keras.regularizers import l2
```

Рисунок 1.3.2

Імпортуємо з фреймворку Keras необхідні нам рівні (рисунок 1.3.2) (Активація, двовимірна згортка, Dropout, згладжуваний та ін.), повно зв’язну архітектуру нейромереж, клас Моделі, регулятори та інші – Рисунок 1.3.3.

```python
def simple_CNN(input_shape, num_classes):

    model = Sequential()
    model.add(Convolution2D(filters=16, kernel_size=(7, 7), padding='same',
                            name='image_array', input_shape=input_shape))
    model.add(BatchNormalization())
    model.add(Convolution2D(filters=16, kernel_size=(7, 7), padding='same'))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(AveragePooling2D(pool_size=(2, 2), padding='same'))
    model.add(Dropout(.5))

    model.add(Convolution2D(filters=32, kernel_size=(5, 5), padding='same'))
    model.add(BatchNormalization())
    model.add(Convolution2D(filters=32, kernel_size=(5, 5), padding='same'))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(AveragePooling2D(pool_size=(2, 2), padding='same'))
    model.add(Dropout(.5))

    model.add(Convolution2D(filters=64, kernel_size=(3, 3), padding='same'))
    model.add(BatchNormalization())
    model.add(Convolution2D(filters=64, kernel_size=(3, 3), padding='same'))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(AveragePooling2D(pool_size=(2, 2), padding='same'))
    model.add(Dropout(.5))

    model.add(Convolution2D(filters=128, kernel_size=(3, 3), padding='same'))
    model.add(BatchNormalization())
    model.add(Convolution2D(filters=128, kernel_size=(3, 3), padding='same'))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(AveragePooling2D(pool_size=(2, 2), padding='same'))
    model.add(Dropout(.5))

    model.add(Convolution2D(filters=256, kernel_size=(3, 3), padding='same'))
    model.add(BatchNormalization())
    model.add(Convolution2D(filters=num_classes, kernel_size=(3, 3), padding='same'))
    model.add(GlobalAveragePooling2D())
    model.add(Activation('softmax',name='predictions'))
    return model
```

Рисунок 1.3.3

Перший тип архітектури загорткової нейромережи, оголошений функціонально. Ця нейромережа і використовується у подальшій роботі.

Можна перемикати на інші нейромережі (Рисунок 1.3.4)

```python
if __name__ == "__main__":
    input_shape = (64, 64, 1)
    num_classes = 7
    #model = tiny_XCEPTION(input_shape, num_classes)
    #model.summary()
    #model = mini_XCEPTION(input_shape, num_classes)
    #model.summary()
    #model = big_XCEPTION(input_shape, num_classes)
    #model.summary()
    model = simple_CNN((48, 48, 1), num_classes)
    model.summary()
```

Рисунок 1.3.4

Реалізація інших нейромереж – Рисунки 1.3.5-1.3.9:

```
if __name__ == "__main__":
    input_shape = (64, 64, 1)
    num_classes = 7
    #model = tiny_XCEPTION(input_shape, num_classes)
    #model.summary()
    #model = mini_XCEPTION(input_shape, num_classes)
    #model.summary()
    #model = big_XCEPTION(input_shape, num_classes)
    #model.summary()
    model = simple_CNN((48, 48, 1), num_classes)
    model.summary()
```

Рисунок 1.3.5

```
def simpler_CNN(input_shape, num_classes):

    model = Sequential()
    model.add(Convolution2D(filters=16, kernel_size=(5, 5), padding='same',
                            name='image_array', input_shape=input_shape))
    model.add(BatchNormalization())
    model.add(Convolution2D(filters=16, kernel_size=(5, 5),
                            strides=(2, 2), padding='same'))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(Dropout(.25))

    model.add(Convolution2D(filters=32, kernel_size=(5, 5), padding='same'))
    model.add(BatchNormalization())
    model.add(Convolution2D(filters=32, kernel_size=(5, 5),
                            strides=(2, 2), padding='same'))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(Dropout(.25))

    model.add(Convolution2D(filters=64, kernel_size=(3, 3), padding='same'))
    model.add(BatchNormalization())
    model.add(Convolution2D(filters=64, kernel_size=(3, 3),
                            strides=(2, 2), padding='same'))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(Dropout(.25))

    model.add(Convolution2D(filters=64, kernel_size=(1, 1), padding='same'))
    model.add(BatchNormalization())
    model.add(Convolution2D(filters=128, kernel_size=(3, 3),
                            strides=(2, 2), padding='same'))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(Dropout(.25))

    model.add(Convolution2D(filters=256, kernel_size=(1, 1), padding='same'))
    model.add(BatchNormalization())
    model.add(Convolution2D(filters=128, kernel_size=(3, 3),
                            strides=(2, 2), padding='same'))

    model.add(Convolution2D(filters=256, kernel_size=(1, 1), padding='same'))
    model.add(BatchNormalization())
    model.add(Convolution2D(filters=num_classes, kernel_size=(3, 3),
                            strides=(2, 2), padding='same'))

    model.add(Flatten())
    #model.add(GlobalAveragePooling2D())
    model.add(Activation('softmax',name='predictions'))
    return model
```

Рисунок 1.3.6

```
def tiny_XCEPTION(input_shape, num_classes, l2_regularization=0.01):
    regularization = l2(l2_regularization)

    # base
    img_input = Input(input_shape)
    x = Conv2D(5, (3, 3), strides=(1, 1), kernel_regularizer=regularization,
                                          use_bias=False)(img_input)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = Conv2D(5, (3, 3), strides=(1, 1), kernel_regularizer=regularization,
                                          use_bias=False)(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)

    # module 1
    residual = Conv2D(8, (1, 1), strides=(2, 2),
                      padding='same', use_bias=False)(x)
    residual = BatchNormalization()(residual)

    x = SeparableConv2D(8, (3, 3), padding='same',
                        kernel_regularizer=regularization,
                        use_bias=False)(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = SeparableConv2D(8, (3, 3), padding='same',
                        kernel_regularizer=regularization,
                        use_bias=False)(x)
    x = BatchNormalization()(x)

    x = MaxPooling2D((3, 3), strides=(2, 2), padding='same')(x)
    x = layers.add([x, residual])

    # module 2
    residual = Conv2D(16, (1, 1), strides=(2, 2),
                      padding='same', use_bias=False)(x)
    residual = BatchNormalization()(residual)

    x = SeparableConv2D(16, (3, 3), padding='same',
                        kernel_regularizer=regularization,
                        use_bias=False)(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = SeparableConv2D(16, (3, 3), padding='same',
                        kernel_regularizer=regularization,
                        use_bias=False)(x)
    x = BatchNormalization()(x)

    x = MaxPooling2D((3, 3), strides=(2, 2), padding='same')(x)
    x = layers.add([x, residual])

    # module 3
    residual = Conv2D(32, (1, 1), strides=(2, 2),
                      padding='same', use_bias=False)(x)
    residual = BatchNormalization()(residual)
```

Рисунок 1.3.7

```
                    use_bias=False)(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = SeparableConv2D(32, (3, 3), padding='same',
                        kernel_regularizer=regularization,
                        use_bias=False)(x)
    x = BatchNormalization()(x)

    x = MaxPooling2D((3, 3), strides=(2, 2), padding='same')(x)
    x = layers.add([x, residual])

    # module 4
    residual = Conv2D(64, (1, 1), strides=(2, 2),
                      padding='same', use_bias=False)(x)
    residual = BatchNormalization()(residual)

    x = SeparableConv2D(64, (3, 3), padding='same',
                        kernel_regularizer=regularization,
                        use_bias=False)(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = SeparableConv2D(64, (3, 3), padding='same',
                        kernel_regularizer=regularization,
                        use_bias=False)(x)
    x = BatchNormalization()(x)

    x = MaxPooling2D((3, 3), strides=(2, 2), padding='same')(x)
    x = layers.add([x, residual])

    x = Conv2D(num_classes, (3, 3),
            #kernel_regularizer=regularization,
            padding='same')(x)
    x = GlobalAveragePooling2D()(x)
    output = Activation('softmax',name='predictions')(x)

    model = Model(img_input, output)
    return model
```

Рисунок 1.3.8

```
def big_XCEPTION(input_shape, num_classes):
    img_input = Input(input_shape)
    x = Conv2D(32, (3, 3), strides=(2, 2), use_bias=False)(img_input)
    x = BatchNormalization(name='block1_conv1_bn')(x)
    x = Activation('relu', name='block1_conv1_act')(x)
    x = Conv2D(64, (3, 3), use_bias=False)(x)
    x = BatchNormalization(name='block1_conv2_bn')(x)
    x = Activation('relu', name='block1_conv2_act')(x)

    residual = Conv2D(128, (1, 1), strides=(2, 2),
                      padding='same', use_bias=False)(x)
    residual = BatchNormalization()(residual)

    x = SeparableConv2D(128, (3, 3), padding='same', use_bias=False)(x)
    x = BatchNormalization(name='block2_sepconv1_bn')(x)
    x = Activation('relu', name='block2_sepconv2_act')(x)
    x = SeparableConv2D(128, (3, 3), padding='same', use_bias=False)(x)
    x = BatchNormalization(name='block2_sepconv2_bn')(x)

    x = MaxPooling2D((3, 3), strides=(2, 2), padding='same')(x)
    x = layers.add([x, residual])

    residual = Conv2D(256, (1, 1), strides=(2, 2),
                      padding='same', use_bias=False)(x)
    residual = BatchNormalization()(residual)

    x = Activation('relu', name='block3_sepconv1_act')(x)
    x = SeparableConv2D(256, (3, 3), padding='same', use_bias=False)(x)
    x = BatchNormalization(name='block3_sepconv1_bn')(x)
    x = Activation('relu', name='block3_sepconv2_act')(x)
    x = SeparableConv2D(256, (3, 3), padding='same', use_bias=False)(x)
    x = BatchNormalization(name='block3_sepconv2_bn')(x)

    x = MaxPooling2D((3, 3), strides=(2, 2), padding='same')(x)
    x = layers.add([x, residual])
    x = Conv2D(num_classes, (3, 3),
            #kernel_regularizer=regularization,
            padding='same')(x)
    x = GlobalAveragePooling2D()(x)
    output = Activation('softmax',name='predictions')(x)

    model = Model(img_input, output)
    return model
```

Рисунок 1.3.9

1.3.2 load_and_process.py

```
import pandas as pd
import cv2
import numpy as np


dataset_path = 'fer2013/fer2013/fer2013.csv'
image_size=(48,48)
```

Рисунок 1.3.10

Завантаження базових бібліотек (рисунок 1.3.10), та оголошення констант – шлях до дата сету та розмір зображень (рисунок 1.3.11).

```
def load_fer2013():
        data = pd.read_csv(dataset_path)
        pixels = data['pixels'].tolist()
        width, height = 48, 48
        faces = []
        for pixel_sequence in pixels:
            face = [int(pixel) for pixel in pixel_sequence.split(' ')]
            face = np.asarray(face).reshape(width, height)
            face = cv2.resize(face.astype('uint8'),image_size)
            faces.append(face.astype('float32'))
        faces = np.asarray(faces)
        faces = np.expand_dims(faces, -1)
        emotions = pd.get_dummies(data['emotion']).as_matrix()
        return faces, emotions
```

Рисунок 1.3.11

Завантаження даних з дата сету та збереження окремих фрагментів даних у різні змінні для подальшої роботи з ними.

```
def preprocess_input(x, v2=True):
    x = x.astype('float32')
    x = x / 255.0
    if v2:
        x = x - 0.5
        x = x * 2.0
    return x
```

Рисунок 1.3.12

Препроцесінг даних – зведення піксельного масштабу до масштабу [0:1] (рисунок 1.3.12)

1.3.3 real_time_video.py

```
from keras.preprocessing.image import img_to_array
import imutils
import cv2
from keras.models import load_model
import numpy as np

# parameters for loading data and images
detection_model_path = 'haarcascade_files/haarcascade_frontalface_default.xml'
emotion_model_path = 'models/_mini_XCEPTION.102-0.66.hdf5'

# hyper-parameters for bounding boxes shape
# loading models
face_detection = cv2.CascadeClassifier(detection_model_path)
emotion_classifier = load_model(emotion_model_path, compile=False)
EMOTIONS = ["angry" ,"disgust","scared", "happy", "sad", "surprised",
 "neutral"]
```

Рисунок 1.3.13

Підключення модулів та оголошення констант, таких як перелік емоцій на рисунку 1.3.13, шляхи то опису моделі та її параметрів, створення класифікаторів та ін.

```
# starting video streaming
cv2.namedWindow('your_face')
camera = cv2.VideoCapture(0)
while True:
    frame = camera.read()[1]
    frame = imutils.resize(frame,width=300)
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    faces = face_detection.detectMultiScale(gray,scaleFactor=1.1,minNeighbors=5,minSize=(30,30),flags=cv2.CASCADE_SCALE_IMAGE)
    canvas = np.zeros((250, 300, 3), dtype="uint8")
    frameClone = frame.copy()
    if len(faces) > 0:
        faces = sorted(faces, reverse=True,
        key=lambda x: (x[2] - x[0]) * (x[3] - x[1]))[0]
        (fX, fY, fW, fH) = faces
                    # Extract the ROI of the face from the grayscale image, resize it to a fixed 28x28 pixels, and then prepare
            # the ROI for classification via the CNN
        roi = gray[fY:fY + fH, fX:fX + fW]
        roi = cv2.resize(roi, (64, 64))
        roi = roi.astype("float") / 255.0
        roi = img_to_array(roi)
        roi = np.expand_dims(roi, axis=0)


        preds = emotion_classifier.predict(roi)[0]
        emotion_probability = np.max(preds)
        label = EMOTIONS[preds.argmax()]
    else: continue


    for (i, (emotion, prob)) in enumerate(zip(EMOTIONS, preds)):
                # construct the label text
                text = "{}: {:.2f}%".format(emotion, prob * 100)

                w = int(prob * 300)
                cv2.rectangle(canvas, (7, (i * 35) + 5),
                (w, (i * 35) + 35), (0, 0, 255), -1)
                cv2.putText(canvas, text, (10, (i * 35) + 23),
                cv2.FONT_HERSHEY_SIMPLEX, 0.45,
                (255, 255, 255), 2)
                cv2.putText(frameClone, label, (fX, fY - 10),
                cv2.FONT_HERSHEY_SIMPLEX, 0.45, (0, 0, 255), 2)
                cv2.rectangle(frameClone, (fX, fY), (fX + fW, fY + fH),
                              (0, 0, 255), 2)


    cv2.imshow('your_face', frameClone)
    cv2.imshow("Probabilities", canvas)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

camera.release()
cv2.destroyAllWindows()
```

Рисунок 1.3.14

Початок роботи з камерою в режимі стриму cv2 через «вічний цикл» while(True) з перериванням по умові на рисунку 1.3.14.

Після знімку зображення та перевірок, воно оброблюється (обрізається та протестується) після чого відправляється на вхід до моделі.

Після відповіді моделі необхідно інтерпретувати цю відповідь та намалювати прямокутник + зробити необхідні написи.

1.3.4 Train_emotion_classifier

```
from keras.callbacks import CSVLogger, ModelCheckpoint, EarlyStopping
from keras.callbacks import ReduceLROnPlateau
from keras.preprocessing.image import ImageDataGenerator
from load_and_process import load_fer2013
from load_and_process import preprocess_input
from models.cnn import mini_XCEPTION
from sklearn.model_selection import train_test_split

# parameters
batch_size = 32
num_epochs = 10000
input_shape = (48, 48, 1)
validation_split = .2
verbose = 1
num_classes = 7
patience = 50
base_path = 'models/'
```

Рисунок 1.3.15

Підключення модулів та оголошення гіпюр-параметрів моделі таких як розмір партії для навчання, кількість ітерацій навчання, вхідний розмір даних, кількість вихідних класів (нейронів) та інших можна бачити на рисунку 1.3.15.

```
# data generator
data_generator = ImageDataGenerator(
                        featurewise_center=False,
                        featurewise_std_normalization=False,
                        rotation_range=10,
                        width_shift_range=0.1,
                        height_shift_range=0.1,
                        zoom_range=.1,
                        horizontal_flip=True)

# model parameters/compilation
model = mini_XCEPTION(input_shape, num_classes)
model.compile(optimizer='adam', loss='categorical_crossentropy',
              metrics=['accuracy'])
model.summary()
```

Рисунок 1.3.16

Створюємо генератор даних для навчання (сирі дані беруться з завантаженого дата сету) та оголошуємо модель, використовуючи її

функціональне представлення з cnn.py , після чого модель компілюється (рисунок 1.3.16).

```
    # callbacks
log_file_path = base_path + '_emotion_training.log'
csv_logger = CSVLogger(log_file_path, append=False)
early_stop = EarlyStopping('val_loss', patience=patience)
reduce_lr = ReduceLROnPlateau('val_loss', factor=0.1,
                              patience=int(patience/4), verbose=1)
trained_models_path = base_path + '_mini_XCEPTION'
model_names = trained_models_path + '.{epoch:02d}-{val_acc:.2f}.hdf5'
model_checkpoint = ModelCheckpoint(model_names, 'val_loss', verbose=1,
                                   save_best_only=True)
callbacks = [model_checkpoint, csv_logger, early_stop, reduce_lr]

# loading dataset
faces, emotions = load_fer2013()
faces = preprocess_input(faces)
num_samples, num_classes = emotions.shape
xtrain, xtest,ytrain,ytest = train_test_split(faces, emotions,test_size=0.2,shuffle=True)
model.fit_generator(data_generator.flow(xtrain, ytrain,
                                        batch_size),
                    steps_per_epoch=len(xtrain) / batch_size,
                    epochs=num_epochs, verbose=1, callbacks=callbacks,
                    validation_data=(xtest,ytest))
```

Рисунок 1.3.17

Оголошуємо всі необхідні сутності для навчання – логер, раннє переривання навчання (спрацює, коли почнеться oversampling), колбеки (рисунок 1.3.17).

Завантажуємо дані з дата сету через оголошену раніше функцію для цього load_fer2013 , розбиваємо дані та мітки на тренувальну і тестову вибірки, після чого запускається генератор навчання.

1.4 Як працює програма

Так ось, як це все працює, це вказано часткою роботи та показати що вийшло на (ілюстрації 1.4.1), та вказано трішечки коду:

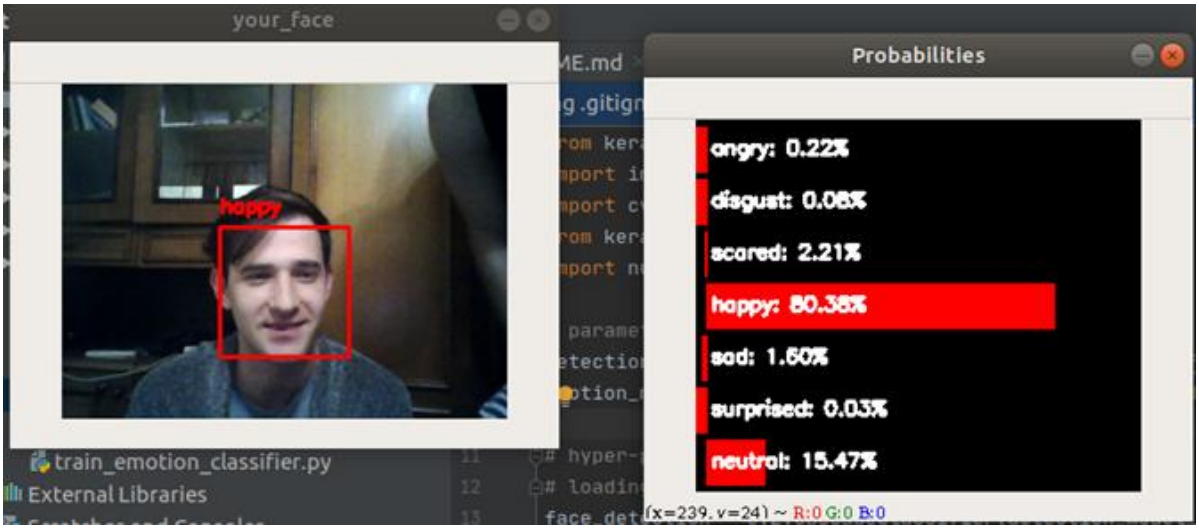

Рисунок 1.4.1

Програма створить вікно для відображення захоплення сцени веб-камерою та вікно, що представляє ймовірність виявлених емоцій (рисунок 1.4.2)

```
8   detection_model_path = 'haarcascade_files/haarcascade_frontalface_default.xml'
9   emotion_model_path = 'models/_mini_XCEPTION.102-0.66.hdf5'
```

Рисунок 1.4.2

Рисунок 1.4.3 містить в собі параметри для завантаження даних та зображень:

```
13   face_detection = cv2.CascadeClassifier(detection_model_path)
14   emotion_classifier = load_model(emotion_model_path, compile=False)
15   EMOTIONS = ["angry" ,"disgust","scared", "happy", "sad", "surprised",
16    "neutral"]
```

Рисунок 1.4.3

У коді 1.1 вказані гіперпараметри для обмежувальних форм коробок та завантаження моделей для обробки нашої нейромережи.

feelings_faces.append(cv2.imread('emojis/' + emotion + '.png', -1)) (Код 1.1)

По цьому індексу йде обробка моє вигляд ось такий:

Запускається відео с веб-камери і проводить обробку отриманих даних (рисунок 1.4.4)

```
cv2.namedWindow('your_face')
camera = cv2.VideoCapture(0)
while True:
    frame = camera.read()[1]
    #reading the frame
    frame = imutils.resize(frame,width=300)
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    faces = face_detection.detectMultiScale(gray,scaleFactor=1.1,minNeighbors=5,minSize=(30,30),flags=cv2.CASCADE_SCALE_IMAGE)

    canvas = np.zeros((250, 300, 3), dtype="uint8")
    frameClone = frame.copy()
    if len(faces) > 0:
        faces = sorted(faces, reverse=True,
        key=lambda x: (x[2] - x[0]) * (x[3] - x[1]))[0]
        (fX, fY, fW, fH) = faces
```

Рисунок 1.4.4

Далі будує текст мітки і рисує ймовірну смужку (мітку) для емоції на обличчі, та обробляє данні, ілюструє в нам в зрозумілу нам формі на рисунку 1.4.5 та коді 1.2:

text = "{}: {:.2f}%".format(emotion, prob * 100) (Код 1.2)

```
                w = int(prob * 300)
                cv2.rectangle(canvas, (7, (i * 35) + 5),
                (w, (i * 35) + 35), (0, 0, 255), -1)
                cv2.putText(canvas, text, (10, (i * 35) + 23),
                cv2.FONT_HERSHEY_SIMPLEX, 0.45,
                (255, 255, 255), 2)
                cv2.putText(frameClone, label, (fX, fY - 10),
                cv2.FONT_HERSHEY_SIMPLEX, 0.45, (0, 0, 255), 2)
                cv2.rectangle(frameClone, (fX, fY), (fX + fW, fY + fH),
                              (0, 0, 255), 2)


        cv2.imshow('your_face', frameClone)
        cv2.imshow("Probabilities", canvas)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

camera.release()
cv2.destroyAllWindows()
```

Рисунок 1.4.5

Це вигляде наступним чином (рисунок1.4.6):

Рисунок 1.4.6

Опис роботи: Модель класифікації тренувань емоцій на рисунку 1.4.7:

```
from keras.callbacks import CSVLogger, ModelCheckpoint, EarlyStopping
from keras.callbacks import ReduceLROnPlateau
from keras.preprocessing.image import ImageDataGenerator
from load_and_process import load_fer2013
from load_and_process import preprocess_input
from models.cnn import mini_XCEPTION
from sklearn.model_selection import train_test_split
```

Рисунок 1.4.7

Параметри на рисунку 1.4.8:

```
batch_size = 32
num_epochs = 10000
input_shape = (48, 48, 1)
validation_split = .2
verbose = 1
num_classes = 7
patience = 50
base_path = 'models/'
```

Рисунок 1.4.8

Генератор на рисунку 1.4.9:

```
data_generator =
ImageDataGenerator(
                    featurewise_center=False,
                    featurewise_std_normalization=False,
                    rotation_range=10,
                    width_shift_range=0.1,
                    height_shift_range=0.1,
                    zoom_range=.1,
                    horizontal_flip=True)
```

Рисунок 1.4.9

Модель параметрів компілятора на (код 1.3)

```
model = mini_XCEPTION(input_shape,num_classes)                    (Код 1.3)
model.compile(optimizer='adam', loss='categorical_crossentropy',
              metrics=['accuracy'])
model.summary()
```

Зворотні дзвінки на рисунку 1.4.10:

```
log_file_path = base_path + '_emotion_training.log'
csv_logger = CSVLogger(log_file_path, append=False)
early_stop = EarlyStopping('val_loss', patience=patience)
reduce_lr = ReduceLROnPlateau('val_loss', factor=0.1,
                                  patience=int(patience/4), verbose=1)
trained_models_path = base_path + '_mini_XCEPTION'
model_names = trained_models_path + '.{epoch:02d}-{val_acc:.2f}.hdf5'
model_checkpoint = ModelCheckpoint(model_names, 'val_loss', verbose=1,
                                                    save_best_only=True)
callbacks = [model_checkpoint, csv_logger, early_stop, reduce_lr]
```

Рисунок 1.4.10

Завантаження набору даних на рисунку 1.4.11:

```
faces, emotions = load_fer2013()
faces = preprocess_input(faces)
num_samples, num_classes = emotions.shape
xtrain, xtest,ytrain,ytest = train_test_split(faces, emotions,test_size=0.2,shuffle=True)
model.fit_generator(data_generator.flow(xtrain, ytrain,
                                        batch_size),
                    steps_per_epoch=len(xtrain) / batch_size,
                    epochs=num_epochs, verbose=1, callbacks=callbacks,
                    validation_data=(xtest,ytest))
```

Рисунок 1.4.11

Підключення бібліотек Pandas (pd), cv2, numpi (np), і його робота на рисунку 1.4.12:

```
dataset_path = 'fer2013/fer2013/fer2013.csv'
image_size=(48,48)

def load_fer2013():
        data = pd.read_csv(dataset_path)
        pixels = data['pixels'].tolist()
        width, height = 48, 48
        faces = []
        for pixel_sequence in pixels:
            face = [int(pixel) for pixel in pixel_sequence.split(' ')]
            face = np.asarray(face).reshape(width, height)
            face = cv2.resize(face.astype('uint8'),image_size)
            faces.append(face.astype('float32'))
        faces = np.asarray(faces)
        faces = np.expand_dims(faces, -1)
        emotions = pd.get_dummies(data['emotion']).as_matrix()
        return faces, emotions

def preprocess_input(x, v2=True):
    x = x.astype('float32')
    x = x / 255.0
    if v2:
        x = x - 0.5
        x = x * 2.0
    return x
```

Рисунок 1.4.12

1.5 Опис роботи Каутеризації

Це задача розбита на заданих вибірках об’єктів (ступень ситуації) на підмножини, що не перетинаються, називаються кластерами, так, щоб кожен кластер складався з схожих об'єктів, а об'єкти різних кластерів істотно відрізнялися.

Робота Кластерів на рисунку 1.5.1:

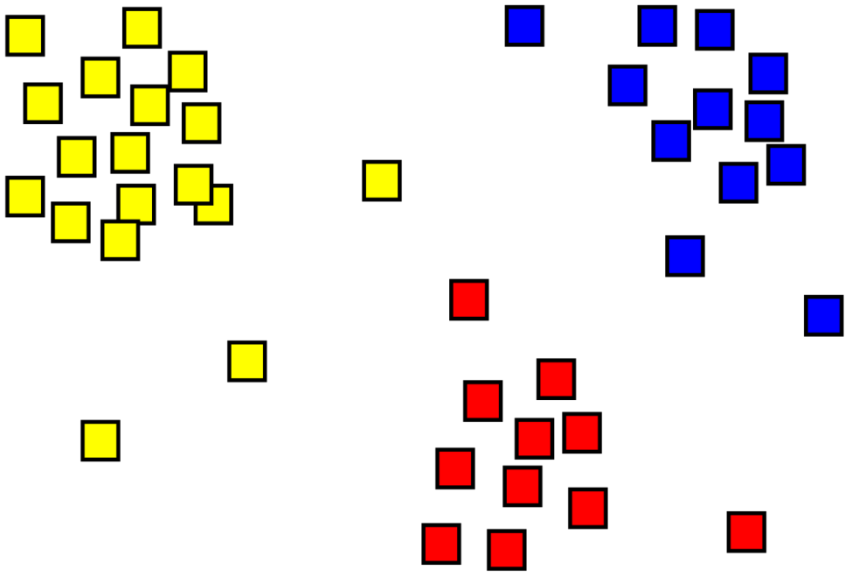

Рисунок 1.5.1

Нехай X — набір об'єктів, Y — номери (імен, мітки) кластерів, Задані функцією відстані між об'єктами $\rho(x,x')$. Є фінальні навчені пункти об'ємами $X^m = \{x_1,\cdots,x_m\} \subset X$. Вам потрібно розділити виділення на іншу проміжну, яка називається кластерами, щоб кожен кластер складався з об'єктів, близькими за показником ρ, а об'єкти різних кластерах дещо різних між собою. При цьому кожен об'єкт $x_i \in X^m$ приписується номер кластера y_i.

Алгоритм каутеризації - це функція $a: X \to Y$, яка будь-якого об'єкта $x \in X$ ставить у відповідність номера кластера $y \in Y$. У цьому Y випадку кожному об’єкту буде присвоєно номер кластера. Алгоритм каутеризації - це функція узгодження будь-якого об’єкта з номером кластера. У деяких випадках багато ситуацій попередньо встановлені, але більше завдань

полягає у визначенні оптимальної кількості кластерів на основі стандартів якості та категорій кластерів. Каутеризація (навчання без програмістів) відрізняється від деяких класифікацій (навчання з програмістами) тим, що мітка вихідного об’єктаy_i спочатку не вказана, і багато об’єктів можуть навіть не бути відомими.Y.

Вирішення проблеми каутеризації є принципово неоднозначним, тому є кілька причин:

- Не можна сказати, що явно існує оптимальний стандарт якості кластера. Існує багато евристичних стандартів, і багато алгоритмів, які не мають чіткого стандарту, але розумного та розумного каутеризації структури, всі вони можуть дати різні результати алгоритму.
- Кількість кластерів, як правило, невідома заздалегідь і встановлюється відповідно до тематичного стандарту алгоритму
- Результат каутеризації повністю залежить від метрики. Вибір метрики, як правило, суб'єктивний і визначається експертами в цій галузі.[6]

РОЗДІЛ 2

DEEP LEARNING ТА МАШИННЕ НАВЧАННЯ

2.1 Модулі які були використані

Keras, Pandas, Tensorflow, CNN, OpenCV і так далі.[16]

2.1.1 Keras

Використана оболонка або фронтону для потужного фрейм ворка Tensorflow, що дозволяє виконувати операції над тензорами, матричні множення і організовує перцептори в цілі рівні, дозволяючи створювати глибокі моделі, з багатьма прихованими рівнями. Keras має зрозумілий та інтуїтивний інтерфейс і дозволяє побудувати різноманітні топології нейромереж (повно зв’язні, рекурентні, складені, ендодерми та їх комбінації). Для роботи з ними існує функціональна та декларативна парадигма, що користувач може обрати. Окрім цього Keras має реалізації багатьох різних оптимізаторів, (loss-функцій та метрики). Все це робить Keras дуже вдалим інструментом для створення нейромереж.[16]

2.1.2 Pandas

Один з класичних інструментів для Data Scientist. Дозволяє розпаковувати **.csv** датасети у спеціальні Python-об’єкти такі як Dataframe, що підтримуються усіма фрейм ворками МН. Також Pandas має багато методів для візуалізації даних, що є дуже корисним на етапі дослідження.[16]

2.1.3 Tensorflow

Потужний фрейм ворк, що дозволяє реалізувати будь-яку топологію НМ та навіть писати власні реалізації рівнів, функцій втрат, оптимізаторів

та ін. Його робота ґрунтується на тензорах – особливих типів багатовимірних даних.[13]

2.2 Визначення шляхів і методів вирішення роботи

2.2.1 Опис програм та стиків

Найзручнішою та найефективнішою мовою програмування для моєї задачі є Python, адже вона має дуже високий рівень продуктивності при обробці даних, дозволяє поєднувати різні типи даних, має хорошу підтримку моєї операційної системи та, найголовніше, для неї наявні дуже важливі бібліотеку для машинного навчання на аналізу інформації — tensorflow, keras та opencv**.**

Однією з найкращий моделей нейронних мереж для задачі аналізу обличь є загорткова нейронна мережа (convolutional neural network) адже натхнення для її створення бралося із зорової кори тварин. Наукові праці стверджують про те, що ця модель є порівняно швидкою і має більш ніж 95% рівень правильності розпізнавання об'єктів на зображенні.[15]

З двох найпоширеніших наборів даних для тренування нейронної мережі Cohn-Kanade+ та FER300 я вибрав FER300, адже він зручніший у використанні для моєї нейронної мережі.

Для виявлення емоцій, серед усіх алгоритмів, я вибрав алгоритм бібліотеки opencv, що називається fisherfaces.

Для більшої наочності правильності роботи програми я вибрав подачу зображення на екран з веб-камери, хоча систему дуже просто переробити для роботи з відеоматеріалом або фотографіями.

Робота додатка з веб-камерою дозволить якнайкраще продемонструвати можливості машинного навчання при роботі з зображеннями та виявленні об'єктів на відео. [12]

2.3 Визначення та застосування ШІ

ШІ (Artificial intelligence, AI)Технічні данні с БД які науково рішені і методидики, за допомогую яких формуеться ШІ обробляе всі вхідні данні на яких вивчае данну йому інформацію і за нею створюе певну свою БД з якої бере інформацію вже обробленну і знахидить вже вирішену задачу на котрій вже навчилось мае аргументовну відповіть як у людському мозку, л огічно підходить до певної задачі. Artificial intelligence обробляе безліч алг оритмів заданних людиною с певними інструментами, алгоритм в системі, до яких входять складові Data science і Machine learning для праці в постав лених задачах.[9]

2.3.1 Класифікація

Найбільше популяризована завданнями МН в ШІ. Вона в чомусь схожа з тим, як людське дитя навчається визначати форму і розмір ширину та об’єми предметів навколо свого зору, складаючи їх в якісь підрозділи своїх розумінь які вклали в них із БЗ людини.

Задача класифікації: передбачити алгоритм критерій об'єкта і розподіл об'єктів згідно з визначеними і вивчених наперед ознаками об’єкту. Тобто машина сортирує отримані данні по потрібним критеріям: одяг - за кольорами і масивом та складових матеріалу, сезон в який його вдягати або тканини з якої виготовлена річь, книги – по категоріям, авторам, мов написання, алфавітному порядку, речовини — за складовим виготовлення (на смак який заданий в БЗ), папір – за товчінню і розміром або галузях в

яких вона використовуються, повідомлення – по ступеню спаму чи повідомлення характерних знань для навчання чи роботи.[15]

2.3.2 Регресія

Це коли потрібно зробити обробку даних, с готової БЗ запозичує відповідь та мінімізує помилки які будуть фатальними.

Курс регресії —для передбачення позиції на вказаній числовій прямій. Наприклад: завантаженість процесів у програмі і виділення оперативної пам’яті, частот процесора (ядр), відео карти та її ресурсів для використання певної програми чи роботи в ній.

Оскільки регресія запрограмована на роботу з точними числами, її вбудовують в різні обчислювальні системи, навіть в класичний Word для праці на ньому та поправок.

Систему класифікації можна «довчити» і навчити вирішувати завдання регресії або комплексних числах. На прикладі це виглядає як розуміння не тільки класу об'єкта, але і його близькості до того чи іншого показника. Наприклад: сировина дерева – якого кольору а бо структури, розміру та якісну дерева і виведе відсоток який скаже на скільки деревина зіпсована.[13]

2.3.3 Штучний інтелект

(Artificial intelligence, AI) – це багато технологічних та наукових вирішень та методів с заданих алгоритмом, програми які роблять ШІ подобним на інтелект людини.[15]

Рисунок 2.3.1

2.4 ШІ в Data science

Це методи які вивчають аналіз даних і вилучають з них цінну інформацію та знання. Вона працює в таких галузях як МН і наука про розсуд як людський мозок (Cognitive Science), а також працює з технологіями для роботи з великим масивом даних (МД) (Big Data). досягненням роботи Data science являться проаналізовані вхідні данні і знаходжень правильного підходу для подальшої обробки даних масивів, розсортування, вибірки та пошуку даних.

Штучні нейронні мережі (ШНМ) це програмна імплементації даних нейронної структури людського мозку. Це не обговорюється складною біологією людської голови, досить знати, що мозок містить нейрони, які є свого роду органічні перемикачі. Вони можуть змінити тип вхідних сигналів в залежності від електричних або біологічних елементів та почуттів як сигнал, які передаються людиною. Нейронна мережа у людському мозку велика і взаємозв’язана підсистема нейронів, де сигнал, який переправляється одним нейроном, може передаватися в мільйони

інших нейронів. Навчання відбувається через повторну структуру активацій деяких нейронних навчань.

Через це збільшується ймовірність знаходження потрібного результату задач по відповідним вхідним інформаціям (сигналах). Такий вид вивчення використовується зворотнім зв’язком при правильному результаті нейронних зав’язків, які виводять його, і стають більше щільнішими. ШНМ повторюють поведінку і стан мозку у простому вигляді (алгоритму). Вони можуть бути навчені контрольованими та неконтрольованими шляхами. У контрольованій ШНМ, алгоритм ШІ вивчає шляхом передачі відповідної вхідної інформації або даних та прикладів вихідної інформації та алгоритму. На прикладі спам фільтруються у електронній поштовій скриньці: вхідна інформація може бути як список речень, які зазвичай знаходяться у (спам) повідомленнях, а вихідною інформацією класифікаційних даних для відповідного повідомлення (спам, чи не спам). Такий вигляд навчання дає вагу и зв’язкам ШНМ.

Неконтрольоване навчання у ШНМ намагається "заставити" ШНМ "розуміти" структуру переданих вхідних інформацій чи даних "самостійно" (рисунок 2.4.1)[15]:

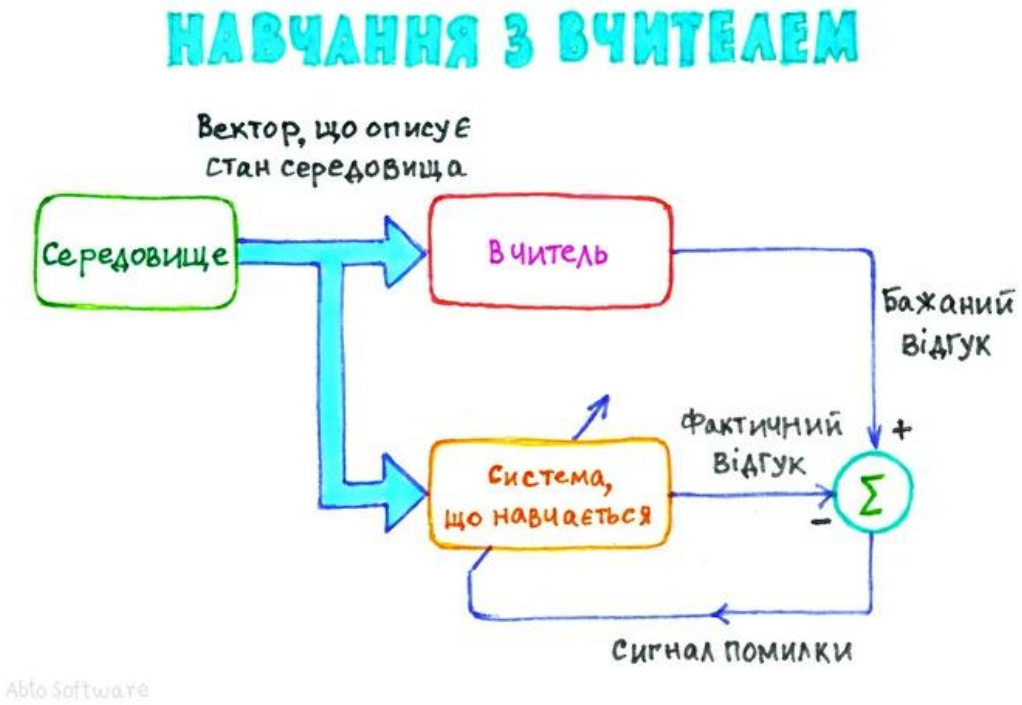

Рисунок 2.4.1

2.4.1 Основні завдання, що вирішуються машинним навчанням Data science

У класичному машинному навчанні з програмістом програмісту, який веде навчання систем нейронних мереж, розмічає дані, відтворює певні машинні приклади та алгоритми за якими навчається ШІ і спостерігає за її розвитком та корегує вхідні данні. Завданням які вирішують за допомогою штучного навчання з програмістом є, приклад алгоритму, класифікація і регресія. Машинне навчання без програміста несе в собі наступні види: каутеризації, узагальнених алгоритмів і БД, пошук як правило заданих в алгоритмі програмістом. Над цими алгоритм часто застосовують програмісти в Data Mining і їх можем виявляти як частину Data Science навчання, рисунок 2.4.1.[1]:


1
Математика
Основа Data Science та машинного навчання знадобиться для глибокого розуміння аналізу даних та принципів машинного навчання
2
Мова програмування
Можете пропустити цей пункт, якщо маєте хоча б якийсь досвід програмування, оскільки Python доволі легка мова програмування
3
Базові алгоритми машинного навчання
Обов'язковий пункт – починайте з нього, якщо ви новачок у Data Science. Python можна і потрібно вивчати під час курсів. Саме таке вивчення буде найпростішим
4
Deep learning
Приступайте до цього пункту після освоєння попереднього
5
Окремі спеціалізації
Приступайте до цього пункту після освоєння двох попередніх

Рисунок 2.4.1

РОЗДІЛ 3

ЗГОРТКОВА НЕЙРОМЕРЕЖА

3.1 Cnn

Найкращій результат в роботі з розпізнаванням цілей показала Convolutional Neural Network або ЗНМ (CNN), яка являться логічним розвитком ідеї певних архітектури НС як когнітронну і неокогнітронну мережу. Успіхи обумовлені можливостю обліку двовимірною топологію зображення, на відмінну від багатошарових персептронів, зокрема з того CNN вимагає менше ресурсів для роботи з зображенням, ніж вимагав би перцептрон при аналогічній якістю робіт, хоч найчастіше класичний перцептрон не може зрівнятися по точності при роботі з зображеннями із CNN так як друге направлення.[3-7]

3.2 Згорткові нейронні мережі

Забезпечують часткову стійкість до змін масштабу, зсувів, поворотів, зміні ракурсу та інших спостережень. ЗНМ об'єднують три архітектурних значень, для забезпечення інваріантності що до заміни масштабу, повороту зрушення і просторовим спостереженням:

- Локальні рецепторні поля (забезпечують локальну двовимірну зв'язність нейронів в цілому);
- Загальну вагову коефіцієнтну синапсичність (забезпечують детектування деяких фотографій або рисунків в будь-яких місць зображення і зменшують загальне число вагових коефіцієнтів);

- Ієрархічна організації з просторовими підвибірками.

На теперішній час момент загорткової нейронної мережі і її модифікації вважаються найкращими по точності і швидкості алгоритму знаходження і визначення об'єктів на сцені. Починаючи з 2012 року, нейромережи займає перше місце на відомому міжнародному конкурсі з розпізнавання образів ImageNet і в навчанні нейронної мережі.

Саме тому в своїй роботі я використовував CNN, засновану на принципах неокогнітрона з доповненим навчанням за алгоритмом зворотного навчання та поширенню помилок на рисунку 3.2.1.[4-5]

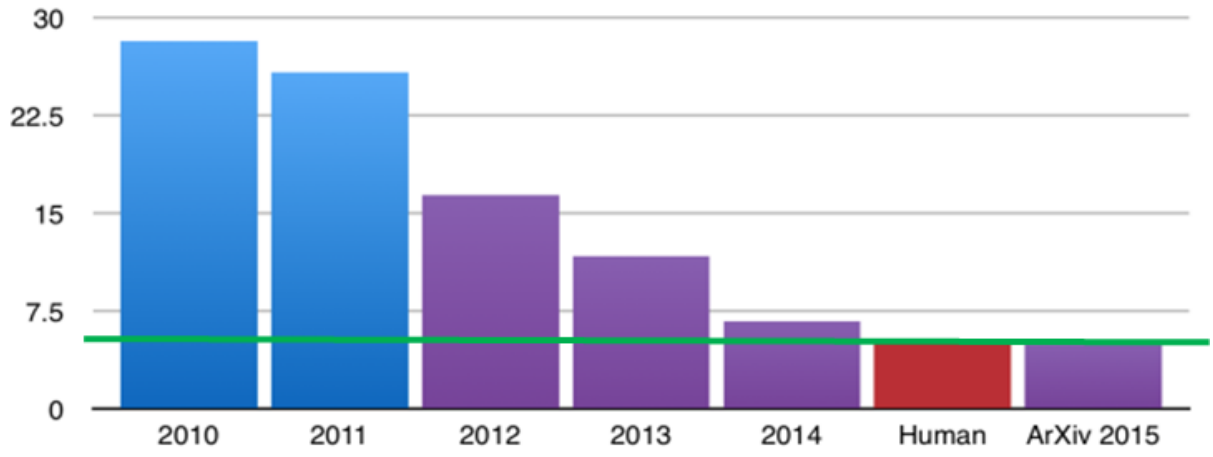

Рисунок 3.2.1

3.2.1 Алгоритм зворотного розпорення помилок

У попередньому розділі обговорювались методи побудови гібридних нейронних мереж, і оскільки нейронні мережі формуються за власними алгоритмами, ці методи побудови можна назвати методами навчання та побудови. Наприклад, при послідовному побудові колекцій за допомогою розширених нейронних мереж продовжуйте вчитися. У магістерській роботі ми будемо використовувати підслуховування

алгоритму навчання та вдосконалювати його. Переглянутий алгоритм навчання включає наступне:

- Форматування навчальних зразків та тестових зразків. Спочатку дані були розділені на дані про навчання та дані тестів. У нашому випадку дані навчання складають 80%, а дані тесту - 20%. 47 Потім кожна вхідна партія S повинна бути розділена на X-вхідні вибірки Value і Y-вибірку вихідного значення.
- 2. Навчальні модулі. Модуль може включати більше двох нейронних мереж, один або кілька типів архітектури. Модуль розділений на три фази для досліджень.[11]

3.3 Алгоритм навчання загорткових нейронних організацій

Щоб почати збирати нашу організацію, вам потрібно вибрати, як вимірювати характер підтвердження. Для нашої ситуації ми використаємо найбільш відому здатність у гіпотезі нейронної організації - це стандартний промах (СКО, MSE), формула 3.1:

$$E^p = \frac{1}{2}(D^p - O(I^p, W))^2 \quad (3.1)$$

У цьому рівнянні Ep - помилка підтвердження для реальної альтернативної пари, що готується, Dp - ідеальна дохідність організації, O (Ip, W) - дохід організації, яка покладається на p-го інформацію та зважування коефіцієнтів W, які включають центри згортки, противаги, навантаження. шари S - і F. Завдання підготовки полягає в зміні ваги W так, щоб для будь-якої пари приводу (Ip, Dp) вони давали базову помилку Ep. Щоб обчислити помилку для всього підготовчого тесту, ми по суті беремо математичне середнє значення помилок для всіх наборів студентів. Помилка будь-якої демонстрації означає Ep.

Щоб обмежити роботу помилок Ер, найкращі стратегії кутів. Розглянемо квінтесенцію стратегій схильності на найпростішій одновимірній моделі (тобто, коли ми маємо лише одну вагу). У випадку, якщо ми погіршимо помилку роботи Ер в домовленості Тейлора, ми отримаємо супровідну артикуляцію, формула 3.2:

$$E(W) = E(W_c) + (W - W_c)\frac{dE(W_c)}{dW} + \frac{1}{2}(W - W_c)^2\frac{d^2E(W_c)}{dW^2} + \dots \quad (3.2)$$

Тут Е - подібна помилка, Wc - основна оцінка ваги. У шкільній математиці ми пам’ятаємо, що, щоб виявити екстремум здатності, нам потрібно взяти її допоміжну і порівняти її з нулем. Отже, ми беремо дочірню компанію за роботу з помилками у вазі, опускаючи умови до наступного запиту, формула 3.3:

$$\frac{dE(W)}{dW} = \frac{dE(W_c)}{dW} + (W - W_c)\frac{d^2E(W_c)}{dW^2} \quad (3.3)$$

з цієї артикуляції випливає, що навантаження, при якій вартість помилки буде незначною, можна визначити з супроводжуючої артикуляції, формула 3.4:

$$W_{min} = W_c - \left(\frac{d^2E(W_c)}{dW^2}\right)^{-1}\frac{dE(W_c)}{dW} \quad (3.4)$$

Тобто, ідеальна вага визначається як поточна вага за вирахуванням допоміжної частини ваги, що не відповідає вазі помилки. Як би там не було, для багатовимірного випадку (тобто для вагової решітки) просто основний підлеглий змінюється нахилом (вектором дробових дочірніх підприємств), а наступне дочірнє підприємство стає гессіанським (сітка приватних дочірніх підприємств секунди). Відтепер дві альтернативи. Якщо ми не помічаємо подальшого підлеглого, ми отримуємо найшвидший розрахунок падіння нахилу.

Зазвичай гессіан витісняється чимось менш складним. Наприклад, надзвичайна в порівнянні з іншими відомими та найкращими стратегіями стратегія Левенберга-Маркара (LM) замінює Гессія, однак для нас важливо подумати про ці два прийоми, це те, що обчислення LM вимагає підготовки всього навчального тесту, тоді як розрахунок занурення нахилу може працювати з будь-ким. інша модель навчання. В останньому випадку розрахунок відомий як стохастичний нахил. Оскільки наш набір даних містить 60 000 навчальних тестів, ми більш пристосовані до стохастичних нахилів. Ще однією перевагою стохастичного нахилу є його менша схильність до найближчого найменшого контрасту з LM. Існує додатково стохастичне коригування розрахунку LM, про яке мова піде далі. Введені рецепти працюють з оцінкою помилки, спричиненої навантаженнями в підсилюючому шарі. Обчислення помилки в прихованих шарах дозволяє помітити в Розділі II техніку зворотного поширення помилки.[4-5]

3.4 Швидкість навчання загорткових нейронних організацій

Навчання під контролем, припускаючи, що кожен вхідний вектор (xi) має вектор вихідних значень (di). Разом ці два вектори називаються навчальними парами (xi, di), а сукупність навчальних пар називається навчальними зразками. Спростить процес навчання, щоб по черзі подавати вхідні дані до навчальної нейронної мережі, обчислюйте похибку між фактичним значенням та очікуваним значенням нейронного δ = y-d та коригуйте параметри мережі, щоб зменшити помилку.Навчання без нагляду - це метод машинного навчання. У цьому рішенні система що тестується вчиться виконувати завдання спонтанно, без втручання експериментаторів. Що стосується кібернетики, то це свого роду експеримент з кібернетики. Зазвичай це стосується лише проблеми відомих описів наборів об'єктів (навчальних зразків), і необхідно виявити

внутрішні взаємозв'язки, залежності та закономірності, що існують між об'єктами.[10]

РОЗДІЛ 4

ДОПОВНЕННЯ ПРОГРАММНОГО ЗАБЕСПЕЧЕННЯ

4.1 Опис програмних засобів

Програмне забезпечення, що є результатом цієї роботи реалізоване за допомогою методів машинного навчання та deep lerning, а саме за допомогою моделей opencv та методів Хаара та Віоли-Джоунса.

Для розробки даного програмного забезпечення мною було обрано мову програмування Python 3, як стандартну мову програмування для нейромереж.

Python є інтерпретованою мовою програмування запозиченою с платформи C++. На початку створення в основному, його основою була об'єктно-орієнтовне програмування як і парадигма, так як Python підтримує всі парадигми, тому є одним з найбільш гнучких мов програмування. Він дуже простий в використанні та вивченню і містить невелику кількість ключових слів і в той же час дуже гнучкий і виразний та легко чіпаються навіть на мізерних знаннях про програмування. Він більш високого рівня ніж C чи C++, C#, java це досягається, в основному, за рахунок вбудованих високорівневих структур даних та бібліотек для запозичень і до нього легко підключити декілька мов програмування (lists, dictionaries, tuples). Всі переваги Python були реалізовані як об’єкти програмування.

Великою перевагою є те, що інтерпретатор Python реалізований практично на всіх платформах відомих нам язиків програмування і в операційних системах, таким чином він є кросплатформенним. Вперше таке було реалізовано з C, проте його типи даних на різних машинах

могли займати різну кількість даних чи ресурсів машин. Python такими недоліками не володіє.

Наступна важлива риса Python — розповсюдженість мови, цьому надається велике значення і, мова була задумана саме як розширювана. Це означає, що є можливість вдосконалення мови з усіма зацікавленими програмістами (тобто Open-Source). Інтерпретатор написаний на C і вихідний код доступний для будь-яких маніпуляцій язику. У разі необхідності, можна вставити його в свою програму і використовувати як вбудовану оболонку. Або ж, написавши на C свої доповнення бібліотек до Python і скомпілювати програму, та отримати "розширений" інтерпретатор з новими можливостями написання коду.[13]

4.2 Переваги Python

Наступна перевага - наявність великого числа програмних модулів, які забезпечують різні додаткові можливості. Такі модулі пишуться на C і C++ на самому Python і можуть бути розроблені усіма досить кваліфікованими програмістами. Наприклад можна навести такі стеки як:

Numerical Python (NumPy) — поширені математичні можливості, такі як маніпуляції з цілими векторами і матрицями;

Tkinter — побудова доданків з використанням графічного інтерфейсу користувача (GUI — Graphical User Interface) на основі широко розповсюдженого на любу версію Windows Tk-інтерфейсу;

OpenGL — використання та запозичення великої бібліотеки графічного моделювання двійкових тривимірних об'єктів (Open Graphics Library фірми Silicon). [13]

4.2.1 Недоліки мови Python

Основним недоліком мови Python є нездатність виконання коду в багатопоточному режимі в повній мірі. Виконання коду в межах одного процесу з розподіленням певних функцій в різні потоки скоріше походить на асинхронний підхід та в деяких випадках працює значно повільніше ніж, якби той самий код працював лише в одному потоці. Така природа виконання передбачена тим, що Python є інтерпретованою мовою програмування. Основний інтерпретатор Python який зараз широко використовується, називається Cython. Так як більшість бібліотек які писались під цей інтерпретатор були написані на мові C, то треба було вирішити проблему перехресного блокування пам’яті. Саме через глобальний блокувальний інтерпретатор, так званий “GIL” – Global Interpreter Lock був побудований механізм який передбачає виконання програми лише в одному потоці. Однак, це дозволяє використовувати під капотом C, для швидких обчислень (наприклад tensorflow), при цьому в наявності доволі велика різноманітність бібліотек та простота реалізації штучних нейромереж, що значною мірою спрощує розробку та не впливає на швидкодію кінцевого продукту.

Зручність цієї мови і розроблений для неї Jupiter-блокнот зробили її улюбленою для розробників моделей машинного навчання, а також для дослідників також є середовище розробки Pycharm.

Для написання модуля, що відповідає за дослідження та навчання нейронної мережі я використав Google Colab. Він є дуже зручним для досліджень, експериментів і розв’язування задач машинного навчання так як в ньому все є певні стеки технологій їх треба тільки вказати. Має величезну кількість переваг в своїй ніші перед стандартною IDE:

По-перше, всі необхідні модулі уже інстальовані, необхідно їх лише імпортувати і надає користувачу потужне апаратне забезпечення з хмари.

По-друге, дозволяє написати код окремими блоками, що можуть запускатися незалежно один від одного, при цьому зберігаючи всі стани локальних змінних та даних. Така система дуже зручна для експериментів, адже для того щоб протестувати невелику ділянку коду вручну, більше не потрібно перезапускати весь код а просто перезапустити код який не відповідає та падає. Ділянка коду, що уже була запущена зберігає свій стан то кінця сесії або примусового перезапуску сесії колі час роботи машини в google colab спливає.

По-третє, такий блокнот дозволяє писати код, та малювати графіки і писати розмічені за допомогою Markdown коментарі в одному із середовищ.

Звичайно google colab має певні недоліки та недопрацьовані категорії, хоча це скоріше межі його компетенції, а саме кожна сесія обмежена в часі на 12 годин, що не дозволить тренувати надто великі моделі. Також Colab не підходить для розробки традиційного софту, такого як два інших модуля цієї роботи: парсер Вікіпедії, та google API.

Для розробки цих програмних модулів я використовував професійне та найзручніше (на мій погляд) IDE від JetBrains – PyCharm.

PyCharm виділяється величезною кількістю різних опцій, налаштувань під себе.

Найбільше мені подобається те, що PyCharm на відміну від більшості IDE та редакторів коду був створений спеціально для розробки програм на мові Python, тому в PyCharm можна інсталювати модулі без

допомоги командної строчки, використовуючи лише контекстне вікно самої IDE.[13]

4.3 Ознаки Хаара

Ці ознаки уже традиційно використовують у машинному навчанні для виявлення та розпізнавання зображень. Ці ознаки також інколи називають «примітиви Хаара» або ще «вейвлети». Вони використовуються для роботи метода Віола-Джонса, про який буде більш детально описано вище. [16]

4.3.1 Примітиви Хаара

Спершу для методу Віола-Джонса використовувалися лише стандартні примітиви (без повороту на 45 градусів), але згодом цей алгоритм модернізували, додавши усі примітиви.

Основною перевагою використання ознак Хаара перед піксельним представленням є значно вища швидкість роботи. В той час, коли з пари пікселів не можливо отримати виважену інформацію для класифікації, примітив Хаара уже дозволяє будувати каскад. Взагалі, пряме призначення ознак Хаара — робота з відео, пошук різних «розривів» між кадрами одного ряду. Також великою перевагою ознак Хаара при роботі з відео є висока швидкість роботи.

Основа роботи методу Хаара — використання коваріацій для порівняння зображень не попіксельно, а шукаючи примітиви Хаара на зображенні і подальша робота із знайденими вейвлетами.

Підхід Хаара використовує чотири основні концепції:

1. Прості прямокутні функції
2. Інтегральне зображення для спростування пошуку
3. Метод ML — AdaBoost

4. Каскадний класифікатор для ефективного зміщення функцій багатьох змінних.

Бібліотека cv2 реалізовує метод Віола-Джонса саме через вейвлети Хаара.[16]

4.4 Метод Віола-Джонса

Виявлення об'єктів це один із алгоритмів машинного навчання, який дозволяє виявляти об'єкти на зображеннях в режимі реального часу зображення, тобто у відео форматі. Він був запропонований Полом Альт Віола та Майклом Джонсом в 2001 році. Цей алгоритм може розпізнавати об'єкти у зображеннях, проте основне завдання при його створенні полягало у виявленні обличь.

Ознаки, використовувані алгоритмом (примітиви Хаара), базуються на сумаризації пікселів по прямокутній області. Наразі, ознаки, запропоновані Віола та Джонс містять більше однієї прямокутної області і є трохи складнішими. Величина кожної ознаки розраховується як кількість пікселів у білих прямокутниках, з яких обчислюється кількість пікселів у чорних областях. Прямокутні ознаки є примітивніші ніж фільтровані, які були реалізовані для пошуку похилих та вертикальних ліній, проте результат їх пошуку у загальному є більш грубим.

Проблема була дещо вирішена при збереженні зображення в інтегральному виді, тобто це означає, що кожен піксель зображення записує суму всіх пікселів ліворуч та вище від себе. При проходженні по інтегральному форматі зображення, прямокутні перевірки ознак на окремих позиціях виконуються постійно, що і дає перевагу при порівнянні.[16]

4.5 Опис OpenCV

Безкоштовна бібліотека комп'ютерного зору (з відкритим вихідним кодом комп'ютерної бібліотеки Vision) є одною з най масштабніших бібліотек з відкритим вихідним кодом, що реалізовує різноманітне програмне забезпечення для комп'ютерного зору і машинного навчання.

OpenCV призначений для забезпечення спільної інфраструктури комп'ютерних доданків, щоб зробити процес взаємодії та інтеграції більш ефективним для використання подальших розробок і рішень великої кількості розробників (програмістів) в готових продуктах. Оскільки OpenCV розповсюджується за ліцензією BSD, кожен може використовувати його як в дослідницьких цілях, так і для комерційних цілей. Бібліотека має більш ніж 2500 оптимізованих алгоритмів, які включають повний набір класичних і сучасних алгоритмів комп'ютерного зору та моделей машинного навчання, серед яких є як уже готові навчені моделі які можна запозичити, так і моделі для попереднього навчання.

Ці алгоритми можуть використовуватися для виявлення та розпізнавання обличь, визначення об'єктів, класифікувати людські дії у відео, відслідковувати рух камери, відслідковувати рухомі об'єкти, витягувати 3D-моделі об'єктів, створювати 3D хмарні точки зі стерео камер, зшивати зображення разом, щоб отримати зображення з високою роздільною здатністю всієї сцени, знайти аналогічні зображення з бази даних зображень, видалити червоні очі з зображень, зроблених з морганням, контролювати рухи очей, розпізнавати позиції очей та частин обличчя (що і дозволяє працювати з класифікацією емоцій на відео і встановити маркери для накладення їх на доповнену реальність і т. д.

Спільнота навколо OpenCV постійно зростає, і тепер вона має більш ніж 150000 користувачів, а кількість завантажень перевищила 20 000 000. Бібліотека глибоко використовується в компаніях, дослідницьких групах та державних органах. Поряд з відомими компаніями, такими як Google, Yahoo, Microsoft, Intel, IBM, Sony, Honda, Toyota, які використовують дану бібліотеку, є багато стартапів, таких JetBrains, VideoServing та інші, які широко використовують OpenCV.

Використання бібліотеки починається з зшивання зображень та дозволяє виявляти об'єкти на відео в Україні чи в Америці, контролювати обладнання в шахтах Китаю, допомагати роботам у навігації та збору об'єктів, виявляти аварії в Європі, а також створювати інтерактивне мистецтво в Іспанії та Нью-Йорку, перевіряти злітно-посадкові смуги в Туреччині на наявність сміття, перевіряючи етикетки на заводах по всьому світу та ін. Бібліотека доступна на багатьох популярних мовах, таких як C, C, Python, Java та MATLAB з підтримкою Методу ВІОЛИ – ДЖОНСА та методом Хаара. [9]

ВИСНОВКИ

В даній роботі було розроблено програмне забезпечення за допомогою засобів машинного навчання. Виконуючи цю роботу, навчився створювати власну нейромережу та тренувати її. Проведена робота аналізує перспективи та проблеми багатокласової семантичної класифікації емоцій. Було розкрита проблематика, та складнощі, що супроводжуються вирішенням проблеми класифікації, зокрема face capture.

Виявлено, що система має високу залежність від освітлення, кількості обличь на екрані, і, найбільше, від апаратних ресурсів комп'ютера.

Незважаючи на всі складнощі, система на машині працює вправно, може виявити яку емоцію показує будь-яке людське обличчя.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. http://itlnu.lviv.ua/machine-learning-ai/ [Електронний ресурс]
2. https://evergreens.com.ua/ua/articles/machine-learning-overview.html [Електронний ресурс]
3. https://evergreens.com.ua/ru/articles/cnn.html [Електронний ресурс]
4. https://habr.com/ru/post/348000/ [Електронний ресурс]
5. https://evergreens.com.ua/ru/articles/classical-machine-learning.html [Електронний ресурс]
6. http://www.machinelearning.ru/wiki/index.php?title=Кластеризация [Електронний ресурс]
7. Real-time Convolutional Neural Networks for Emotion and Gender Classification by Octavio Arriaga, Paul G. Ploeger, Matias Valdenegro https://arxiv.org/abs/1710.07557 2017 [Електронний ресурс]
8. Fisherfaces by Aleix Martinez, Ohio State University, OH http://www.scholarpedia.org/article/Fisherfaces 2011 [Електронний ресурс]
9. Emotion Recognition With Python, OpenCV and a Face Dataset by Paul Vangent http://www.paulvangent.com/2016/04/01/emotion-recognition-with-python-opencv-and-a-face-dataset/ 2016 [Електронний ресурс]
10. FER 2013 Dataset by Microsoft inc. https://www.kaggle.com/c/challenges-in-representation-learning-facial-expression-recognition-challenge/data 2013 [Електронний ресурс]
11. Распознаем лица на фото с помощью Python и OpenCV by galvanom https://habr.com/ru/post/301096/ 2016 [Електронний ресурс]
12. Emotion Recognition using Facial Landmarks, Python, DLib and Open CV by Paul Vangent http://www.paulvangent.com/2016/08/05/emotion-recognition-using-facial-landmarks/#more-565 2016 [Електронний ресурс]

13. Шолле Франсуа Ш78 Глибоке навчання на Python. - Спб.: Пітер, 2018. - 400 с.: мул. -(Серія "Библиотека програміста"). ISBN 978-5-4461-0770-4
14. Рашид, Тарик. С58 Создаем нейронную сеть. : Пер. с англ. — СПб. : ООО “Альфа-книга”, 2017. — 272 с. : ил. — Парал. тит. англ. ISBN 978-5-9909445-7-2 (рус.)
15. Каниа Алексеевич Кан «Нейронный сети. Эволюция» SelfPub; 2018
16. Пол Бэрри: «Изучаем программирование на Python», 2-е издание [пер. с англ. М.А. Райтнан] —Москва: Издательство «Э», 2017.-624 с. : ил.—(Мировой компьютерный бестселлер).