

### **Лабораторная работа №3.**

#### **Разработка модели для определения психо-эмоционального состояния по видеоизображениям**

**Целью** данной лабораторной работы является написание программы, которая позволит распознать и сказать нам, какую эмоцию испытывает человек в данный момент, просто взглянув на выражение его лица. Нам предстоит определить, сердится ли он, счастлив, грустен и т. д.

**Данные:** Готовых датасетов с классификацией эмоций по выражению лица в открытом доступе оказалось еще меньше, чем аудио-датасетов. Выбрали датасет fer2013, сокращение от Facial Expression Recognition (распознавание мимики) и представляет собой набор данных с открытым исходным кодом (<https://www.kaggle.com/msambare/fer2013>). Он содержит изображения лиц в оттенках серого размером 48\*48 пикселей. Также датасет включает в себя сразу не только обучающую выборку, но и тренировочную и тестовую тоже. Датасет включает в себя набор из 35887 изображений, из них 80% - обучающая выборка, 10% - тренировочная, 10% - тестовая. В данных присутствует семь категорий: 0 = гнев, 1 = отвращение, 2 = страх, 3 = счастье, 4 = грусть, 5 = удивление, 6 = нейтральная эмоция.

Примеры изображений из fer2013:

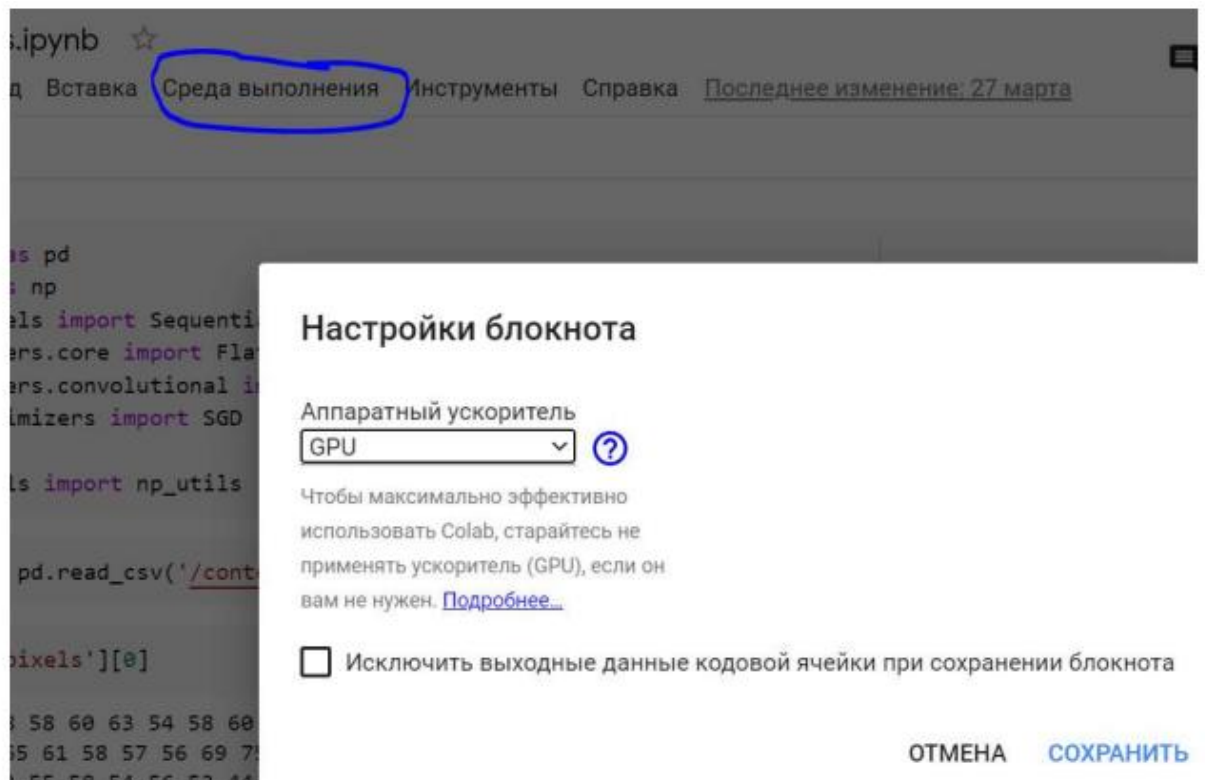


Данный датасет создавался для достижения следующих целей:

- 1) Применять сверточные нейронные сети (CNN) для распознавания мимики.
- 2) Правильно классифицировать каждое изображение лица по одной из семи категорий эмоций лица.

#### **Основные этапы:**

Так как обучение сверточных нейросетей довольно энергозатратное занятие, то для этих целей мы будем использовать Google Colab, с возможностью вычислений на облачном GPU (Graphics processing unit (Графический процессор)) от google. Чтобы включить графический процессор в Google Colab, для получения быстрой обработки данных. Нам нужно, перейти на вкладку «Среда выполнения», затем нажать «сменить среду выполнения» и выбрать GPU.



Как только GPU будет включен, мы можем приступить к вычислениям. Первым делом импортируем необходимые библиотеки для построения сети. Код для импорта библиотек приведен ниже.

```
import pandas as pd
import numpy as np
from keras.models import Sequential
from keras.layers.core import Flatten, Dense, Dropout
from keras.layers.convolutional import Convolution2D, MaxPooling2D, ZeroPadding2D
from keras.optimizers import SGD
from keras.utils import np_utils
```

Как видно, помимо стандартных библиотек, таких как pandas и numpy там потребуется довольно много модулей из библиотеки keras – библиотека для создания для быстрой и понятной работы с нейросетями глубинного обучения, при это спроектирована таким образом, чтобы быть модульной, компактной и расширяемой. Помимо всего, данная библиотека написана непосредственно на языке Python. После импорта библиотек нам необходимо загрузить сам датасет. Так как он довольно большой, советую сразу загрузить

его на гугл диск и потом каждый раз считывать оттуда, чтобы не ждать долго, пока файл прогрузится в колаб.

```
emotion_data = pd.read_csv('/content/drive/MyDrive/fer2013.csv')
```

```
emotion_data.head()
```

	emotion	pixels	Usage
0	0	70 80 82 72 58 58 60 63 54 58 60 48 89 115 121...	Training
1	0	151 150 147 155 148 133 111 140 170 174 182 15...	Training
2	2	231 212 156 164 174 138 161 173 182 200 106 38...	Training
3	4	24 32 36 30 32 23 19 20 30 41 21 22 32 34 21 1...	Training
4	6	4 0 0 0 0 0 0 0 0 0 0 3 15 23 28 48 50 58 84...	Training

Как видно, датасет, действительно, состоит из трех столбцов: эмоция, вектор пикселей, и метка набора (обучающий, тренировочный или тестовый). Далее мы создаем массивы тренировочной и тестовой выборки, и добавляем в них значения пикселей, в соответствии с их меткой:

```
X_train = []
y_train = []
X_test = []
y_test = []
for index, row in emotion_data.iterrows():
    k = row['pixels'].split(" ")
    kint = []
    for i in k:
        kint.append(int(i))
    if row['Usage'] == 'Training':
        X_train.append(np.array(kint))
        y_train.append(row['emotion'])
    elif row['Usage'] == 'PublicTest':
        X_test.append(np.array(kint))
        y_test.append(row['emotion'])
```

После того, как мы добавили векторы пикселей в списки, нам необходимо сконвертировать их в массивы NumPy и поменять размерность X\_train, X\_test. После этого преобразуем метки обучения и метки тестирования в категориальные, как это представлено на коде, приведенном ниже.

```
▶ X_train = np.array(X_train)
  y_train = np.array(y_train)
  X_test = np.array(X_test)
  y_test = np.array(y_test)

  X_train = X_train.reshape(X_train.shape[0], 48, 48, 1)
  X_test = X_test.reshape(X_test.shape[0], 48, 48, 1)
  |
  y_train = np_utils.to_categorical(y_train, num_classes=7)
  y_test = np_utils.to_categorical(y_test, num_classes=7)
```

Теперь, когда мы привели данные к нужному формату, можно начинать разрабатывать многослойную модель CNN для обнаружения эмоций . Мы начинаем с инициализации модели, за которой следует слой батч-нормализации, а затем различные уровни конвентов с функцией активации ReLu, а также макс-пулинг и дроп-аут слои для эффективного обучения.



```

model = Sequential()
model.add(ZeroPadding2D((1,1),input_shape=(48,48,1)))
model.add(Convolution2D(64, (3, 3), activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(128, (3, 3), activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(256, (3, 3), activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(256, (3, 3), activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(256, (3, 3), activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(512, (3, 3), activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(512, (3, 3), activation='relu'))

```

В конце добавляются 2 дроп-аут слоя с коэффициентами 0.5, что означает, что во время обучения этих слоев 50% данных случайным образом выкидываются и набора, что делает нашу сеть более устойчивой к разнородным данным и выбросам. А также в последнем слое необходимо выбрать функцию активации softmax – это обобщенный случай логистической регрессии для многомерных данных. В нашем случае классов 7, поэтому мы применяем именно softmax.

```

model.add(Flatten())
model.add(Dense(4096, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(4096, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(7, activation='softmax'))

```

После этого мы компилируем модель, используя в качестве оптимизатора стохастический градиентный бустинг, как функцию потерь выберем категориальную кросс-энтропию, а в качестве метрики выберем ассурасу:

```

model.compile(optimizer='sgd', loss='categorical_crossentropy', metrics=['accuracy'])

```

После того, как модель скомпилирована можем начать обучение нашей сети. Выбрали 30 эпох обучения и размер батчей равным 32, данные параметры были подобраны опытным путем, их всегда можно поменять.

```

model.fit(X_train, y_train, batch_size=32, epochs=30, verbose=1, validation_data=(X_test, y_test))

```

```

Epoch 1/30
898/898 [=====] - 124s 104ms/step - loss: 1.8366 - accuracy: 0.2433 - val_loss: 1.8102 - val_accuracy: 0.2494
Epoch 2/30
898/898 [=====] - 92s 102ms/step - loss: 1.8052 - accuracy: 0.2530 - val_loss: 1.7743 - val_accuracy: 0.2678
Epoch 3/30
898/898 [=====] - 92s 102ms/step - loss: 1.6708 - accuracy: 0.3336 - val_loss: 1.5215 - val_accuracy: 0.4166
Epoch 4/30
898/898 [=====] - 92s 102ms/step - loss: 1.5052 - accuracy: 0.4084 - val_loss: 1.4452 - val_accuracy: 0.4472
Epoch 5/30
898/898 [=====] - 92s 102ms/step - loss: 1.3801 - accuracy: 0.4652 - val_loss: 1.3205 - val_accuracy: 0.4915
Epoch 6/30

```

После завершения обучения мы можем оценить модель и вычислить ее точность, используя приведенный ниже код.

```

loss_and_metrics = model.evaluate(X_test, y_test)

```

```

print(loss_and_metrics)

```

```

[2.1878998279571533, 0.6010030508041382]

```

Как видно, точность нашей модели по определению 7 эмоций составляет 60%, что довольно неплохо, если учесть, что научный Журнал IEEE Transactions on Affective Computing, выпускаемый Институтом инженеров электротехники и электроники (IEEE — международная некоммерческая ассоциация специалистов в области техники, мировой лидер в области разработки стандартов по радиоэлектронике, электротехнике и аппаратному обеспечению вычислительных систем и сетей) в марте 2020 года показал в своем обзоре, что наилучшая точность составляет всего 70% для данного набора данных с использованием чистой архитектуры CNN. Теперь нам необходимо форматировать нашу модель в json и записать веса нейросети в файл .h5, это нужно для того, чтобы каждый раз не обучать нашу нейросеть заново, а использовать уже готовые веса обученной нейросети для распознавания эмоций по изображению. Сделать это можно следующим образом:

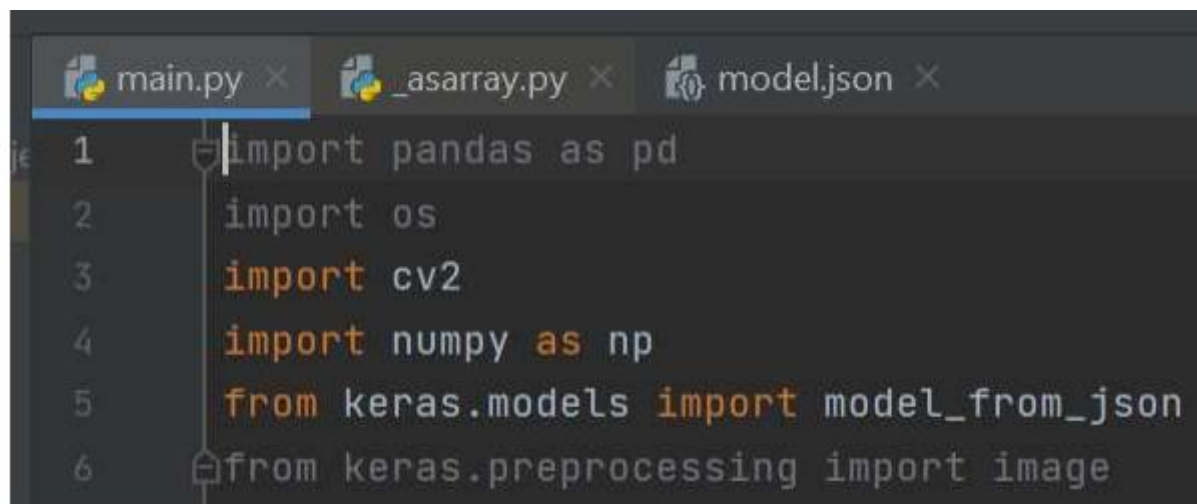
```
#сохраняем веса модели в json
model_json = model.to_json()
with open("model.json", "w") as json_file:
    json_file.write(model_json)
    model.save_weights("model.h5")
print("Saved model to disk")
```

Saved model to disk



## ЧАСТЬ 2. Запуск модели в реальном времени

Теперь мы запустим и протестируем созданную нами модель для обнаружения эмоций в реальном времени с помощью OpenCV и веб-камеры. Для этого мы напишем скрипт на Python. Нам необходимо будет писать скрипт в локальной IDE, чтобы использовать веб-камеру нашего компьютера. Выберем среду разработки PYcharm. Первым делом импортируем библиотеки, представленные на фрагменте кода ниже:



```
1 import pandas as pd
2 import os
3 import cv2
4 import numpy as np
5 from keras.models import model_from_json
6 from keras.preprocessing import image
```

как видно, мы импортировали все те же библиотеки, что и в прошлый раз, однако появилась одна новая – cv2, или полное ее название OpenCV (Open Source Computer Vision Library) - библиотека алгоритмов машинного обучения в части компьютерного зрения, обработки изображений и численных алгоритмов общего назначения, реализованная на C/C++. Примечательной чертой данной библиотеки является то, что это библиотека с открытым кодом, а также в ней есть уже много готовых алгоритмов, которыми я и воспользуюсь. Как мы помним, наша нейронная сеть обучалась на изображениях лиц следующего типа:



Отличительной чертой данных изображений является то, что на фото присутствует только лицо человека, и нет никаких посторонних предметов, однако, в жизни мы получаем изображения с камер, где помимо человеческих лиц содержится еще очень много лишней, в нашем случае, информации. Поэтому, для применения нашей нейронной сети в реальном мире нам необходимо «вырезать» лицо человека из видео потока. Для этого нам и понадобятся встроенные алгоритмы библиотеки OpenCV. В ней уже есть готовая нейросеть для распознавания человеческого лица на фотографии. С помощью приведенного ниже кода импортируем, так называемый каскад Хаара, который и будет отвечать за обнаружение лиц в видеопотоке в нашей программе, следует отметить, что данный файл также работает только с черно-белыми изображениями.

```
face_cascade_db = cv2.CascadeClassifier(cv2.data.harcascades+\
                                         'haarcascade_frontalface_default.xml')
```

Следующим шагом загружаем веса нашей нейронной сети и саму сеть, которые мы получили с помощью Google Colab и сохранили в файлы .h5 и .json.

```
model = model_from_json(open("model.json", "r").read())
model.load_weights('model.h5')
```

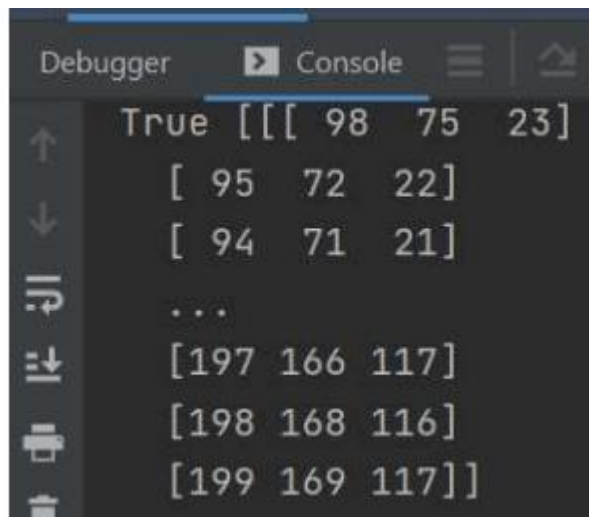
Важно, чтобы наши файлы лежали в одной директории с файлом скрипта, иначе придется в кавычках полностью прописывать путь к нашим файлам. Теперь нам необходимо подключиться к веб-камере компьютера, для этого воспользуемся модулем VideoCapture из библиотеки cv2, данный модуль позволяет осуществлять покадровую обработку поточного видеосигнала. В скобках указывается id камеры, которую мы хотим использовать. По умолчанию: 0 – фронтальная, 1 – тыловая, если такая имеется.

```
cap = cv2.VideoCapture(0)
```

После импорта всех необходимых модулей, мы наконец можем написать код для обнаружения лиц и классификации желаемых эмоций. Так как наша программа должна работать в режиме реального времени без остановок сделаем бесконечный цикл и уже внутри него будем реализовывать наш алгоритм.

```
14 while True:
15     sec, image = cap.read()
16     #print(sec, image)
17     converted_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
18     faces_detected = face_cascade_db.detectMultiScale(converted_image)
19
```

Действие cap.read() включает нашу веб-камеру и передает два значения: булеву переменную (True, если сигнал с камеры поступает и False в противном случае) и вектор набора пикселей кадра из видеопотока. Можем посмотреть эти значения в консоли:



Так как и наша нейросеть обучалась на черно-белых изображениях и каскады Хаара также работают лучше на черно-белых кадрах необходимо нашу картинку, запечатленную камерой конвертировать в ЧБ формат, для этого также уже есть готовый модуль у библиотеки OpenCV. Передаем в `cvtColor` наше изображение и параметр, указывающий, что мы из цветного изображения хотим получить черно-белое. И далее уже на конвертированной картинке применяем файл каскада Хаара для определения человеческого лица, который возвращает 4 параметра:

1. Координата X
2. Координата Y
3. Ширина w
4. Высота h

Это параметра квадрата с левым верхним углом в точке (x;y) и размером w\*h, внутри которого находится человеческое лицо. После получения данных параметров мы уже можем подготовить изображение с камеры для обработки нашей нейросетью, следующим образом:

```

20     for (x,y,w,h) in faces_detected:
21         cv2.rectangle(image,(x,y), (x+w,y+h), (255,0,0))
22
23         face_gray = converted_image[y:y+w,x:x+h]
24         face_gray = cv2.resize(face_gray,(48,48))
25         #print(face_gray)
26         image_pixels = np.array(face_gray)
27         image_pixels = np.expand_dims(image_pixels, axis=-1)
28         image_pixels = image_pixels.reshape(image_pixels.shape[0], 48, 48, 1)
29

```

Нарисуем квадрат вокруг человеческого лица, для этого передадим в cv2.rectangle наше изображение, где мы будем его рисовать, расположение и размеры квадрата (взятые из каскада Хаара), а также цвет рамки в виде вектора BGR (обратите внимание, не RGB как обычно) далее обрежем изображение, полученное с камеры ровно до размеров квадрата с лицом и трансформируем его в подходящий размер (48\*48, как раз тот, на котором мы обучали нашу нейронную сеть). Таким образом мы из цветной картинке с веб-камеры, получили ЧБ изображение только лица размером 48\*48, как раз такое, которое было в обучающем датасете. Далее преобразуем этот набор пикселей в массив нампай такой же формы, которая должна поступать на вход нашей обученной модели и сохраняем в переменную image\_pixels. Теперь пришло время предсказывать эмоции:

```

30     predictions = model.predict(image_pixels)
31     print(predictions)
32     max_index = np.argmax(predictions[0])
33     |
34     emotion_detection = ('angry', 'disgust', 'fear', 'happy', 'sad', 'surprise', 'neutral')
35     emotion_prediction = emotion_detection[max_index]
36

```

Просим нашу модель предсказать к какому из семи классов эмоций относится наше изображение, модель возвращает вероятности, с которой наше изображение относится к каждому из классов:

```

[[[1.3204028e-03 1.7537033e-08 1.2812833e-04 2.9862311e-05 4.6576193e-04
  6.5597042e-06 9.9804926e-01]]]

```



Затем выбираем класс с максимальной вероятностью – это и будет нашей эмоцией, которую в данный момент испытывает человек на картинке. Выведем необходимую нам эмоцию на экран:

```
37 cv2.putText(image, emotion_prediction, (int(x), int(y)),  
38 cv2.FONT_HERSHEY_SIMPLEX, 2, (0, 255, 0), 3)
```

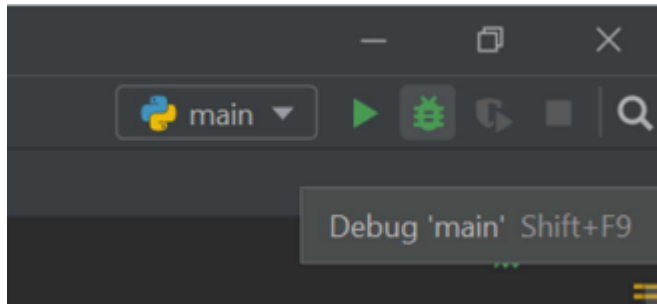
Для этого используем метод `putText`, в него необходимо передать изображение, на которое мы хотим вывести текст, в нашем случае это изначальное изображение с камеры, сам текст, у нас он записан в переменной `emotion_prediction` – равную текущей эмоции, предсказанной моделью. Координаты местоположения текста (можно указать прямо над рамкой с лицом), и параметры стиля, размера, цвета надписи и толщины текста. Осталось только вывести все это на экран в отдельное окно, сделаем это следующим образом:

```
39  
40 resize_image = cv2.resize(image, (1000, 600))  
41 cv2.imshow('Emotion by Grigorev', resize_image)
```

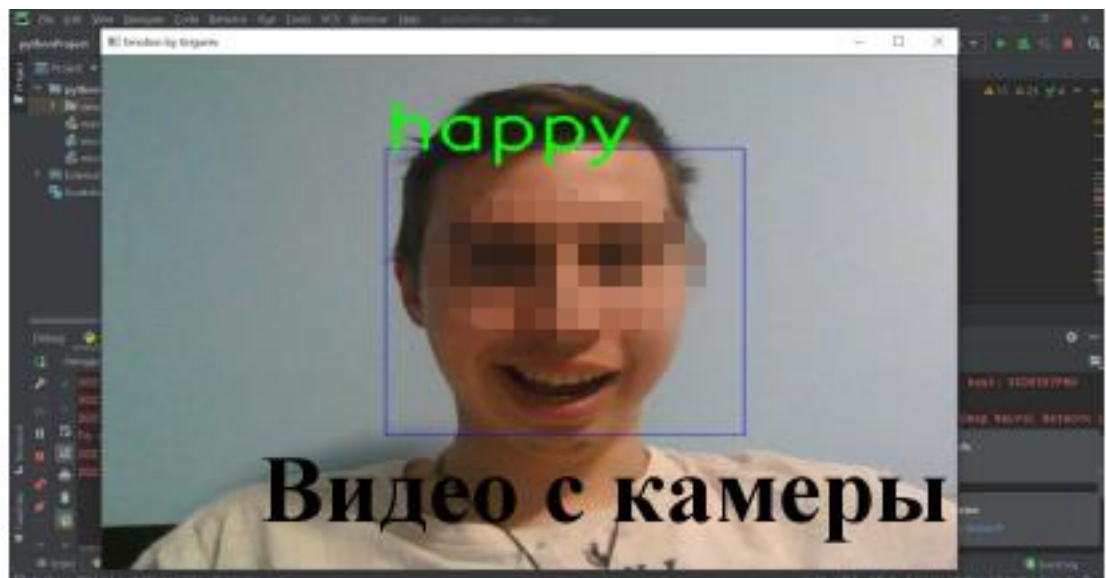
Форматируем наше изображение в нужный нам размер (в моем случае это 1000\*600 пикселей) и с помощью метода `imshow` создадим окно с нашим изображением, в качестве первого параметра нужно также передать заголовок окна. Осталась одна незначительная деталь. Поскольку изначально мы делали бесконечный цикл, надо теперь сделать путь для выхода из него:

```
42 if cv2.waitKey(10) == ord('b'):  
43     break  
44  
45 cap.release()  
46 cv2.destroyAllWindows  
47 print('success')
```

Для завершения работы программы необходимо нажать клавишу «b», после чего цикл прекратится, и все созданные окна уничтожатся, а мы увидим приятную надпись «sucsess» в нашей консоли. Протестируем нашу программу. Для запуска программы необходимо нажать иконку debug в правом верхнем углу программы PyCharm



После запуска у нас сразу включится веб-камера и откроется окно программы, для распознавания эмоций в режиме реального времени.



Чтобы распознавать эмоции на нескольких лицах одновременно, вам нужно установить face\_recognition. Ниже приведен код программы для распознавания эмоций нескольких человек в режиме реального времени.:

```
import cv2
import face_recognition
import numpy as np
from keras.models import model_from_json

video_capture = cv2.VideoCapture(0)
face_locations = []
```

```

model = model_from_json(open("model.json", "r").read())
model.load_weights('model.h5')
emotion_values = ['angry', "disgust", "fear", "happy", "sad",
"surprise", "neutral"]

while True: # Grab a single frame of video
    ret, frame = video_capture.read()# Convert the image from
BGR color (which OpenCV uses) to RGB color (which
face_recognition uses)
    rgb_frame = frame[:, :, ::-1]# Find all the faces in the current
frame of video
    face_locations = face_recognition.face_locations(rgb_frame)#
Display the results

    for top, right, bottom, left in face_locations:
        # Draw a box around the face
        cv2.rectangle(frame, (left, top), (right, bottom), (0, 0, 255),
2)# Display the resulting image
        face = frame[top:bottom, left:right]
        if len(face) > 0:
            face_gray = cv2.cvtColor(face, cv2.COLOR_BGR2GRAY)
            face_gray48 = cv2.resize(face_gray, (48, 48))

            image_pixels = np.array(face_gray48)
            image_pixels = np.expand_dims(image_pixels, axis=0)
            image_pixels =
image_pixels.reshape(image_pixels.shape[0], 48, 48, 1)

            predictions = model.predict(image_pixels)
            max_index = np.argmax(predictions)

            emotion_prediction = emotion_values[max_index]

            cv2.putText(frame, 'face', (left, top - 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, (36, 255, 12), 1)
            cv2.putText(frame, emotion_values[max_index], (left,
top - 25), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (36, 255, 12), 1)
            cv2.imshow('Video', frame)# Hit 'q' on the keyboard to
quit!
            if cv2.waitKey(1) & 0xFF == ord('q'):
                break

```