Глубокое обучение в компьютерном зрении. Интерактивный курс

Урок 7. Обработка видео

Практическое задание 7

Обучить нейронную сеть для распознавания действий человека по видео на датасете KTH
Библиотеки: [Python, Tensorflow]

Выполнил ***Соковнин ИЛ***

За основу взят файл **7_Video.ipynb** из 7 занятия.

Переключение версии TensorFlow

В [1]:
```
%tensorflow_version 2.x
```

```
Colab only includes TensorFlow 2.x; %tensorflow_version has no effect.
```

В [2]:
```
import os
import glob
import random
import numpy as np
import matplotlib.pyplot as plt
import math

import tensorflow as tf
```

В [3]:
```
# Установка библиотеки scikit-video
# Будем использовать для чтения видео
if 0:
    !pip install scikit-video==1.1.11
import skvideo.io
```

```
Looking in indexes: https://pypi.org/simple, (https://pypi.org/simple,) https://us-python.pkg.dev/colab-wheels/public/simple/ (https://us-python.pkg.dev/colab-wheels/public/simple/)
Collecting scikit-video==1.1.11
  Downloading scikit_video-1.1.11-py2.py3-none-any.whl (2.3 MB)
     |████████████████████████████████| 2.3 MB 5.1 MB/s
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (from scikit-video==1.1.11) (1.21.6)
Requirement already satisfied: scipy in /usr/local/lib/python3.7/dist-packages (from scikit-video==1.1.11) (1.7.3)
Requirement already satisfied: pillow in /usr/local/lib/python3.7/dist-packages (from scikit-video==1.1.11) (7.1.2)
Installing collected packages: scikit-video
Successfully installed scikit-video-1.1.11
```

Загрузка и распаковка датасета KTH

Датасет KTH - набор видеофайлов в формате avi
. 6 классов:

- прогулка
- пробежка
- бег
- боксинг
- махание руками
- хлопанье руками

В [ ]:
```
if 0:
#     # По этим ссылке датасет недоступен
#     !wget http://www.nada.kth.se/cvap/actions/walking.zip
#     !wget http://www.nada.kth.se/cvap/actions/jogging.zip
#     !wget http://www.nada.kth.se/cvap/actions/running.zip
#     !wget http://www.nada.kth.se/cvap/actions/boxing.zip
#     !wget http://www.nada.kth.se/cvap/actions/handwaving.zip
#     !wget http://www.nada.kth.se/cvap/actions/handclapping.zip

      !wget http://www.csc.kth.se/cvap/actions/walking.zip
      !wget http://www.csc.kth.se/cvap/actions/jogging.zip
      !wget http://www.csc.kth.se/cvap/actions/running.zip
      !wget http://www.csc.kth.se/cvap/actions/boxing.zip
      !wget http://www.csc.kth.se/cvap/actions/handwaving.zip
      !wget http://www.csc.kth.se/cvap/actions/handclapping.zip
```

В [ ]:
```
# !rm -rf walking
# !rm -rf jogging
# !rm -rf running
# !rm -rf boxing
# !rm -rf handwaving
# !rm -rf handclapping
```

В [4]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

В [8]:
```
!ls /content/drive/MyDrive/CV
```

```
boxing.zip        jogging.zip                         running.zip
handclapping.zip  lesson_7_cv_hw_colab.ipynb          walking.zip
handwaving.zip    person01_handwaving_d3_uncomp.avi
```

В [12]:
```
if 0:
  !unzip /content/drive/MyDrive/CV/walking.zip -d walking > /dev/null
  !unzip /content/drive/MyDrive/CV/jogging.zip -d jogging > /dev/null
  !unzip /content/drive/MyDrive/CV/running.zip -d running > /dev/null
  !unzip /content/drive/MyDrive/CV/boxing.zip -d boxing > /dev/null
  !unzip /content/drive/MyDrive/CV/handwaving.zip -d handwaving > /dev/null
  !unzip /content/drive/MyDrive/CV/handclapping.zip -d handclapping > /dev/null
```

В [ ]:
```
!chcp 65001
```

```
Active code page: 65001
```

Подготовка датасета для классификации

Решение, предложенное на видеоуроке, дополняется следующим:

- делаем разбиение на тестовую и тренировочную выборку
- делаем пайплайн данных с помощью tf.data
- в модель введем регуляризацию Dropout
- уменьшим learning_rate в два раза, до 0.0005

B [13]:
```
classes = [
    'walking',
    'jogging',
    'running',
    'boxing',
    'handwaving',
    'handclapping',
]


# Создаём датасет, каждый элемент которого содержит путь к файлу и номер класса
dataset = []
data_root = './'  # Путь откуда берём файлы
# classes = os.listdir(data_root)

for cls in classes:
    # print('Processing class: {}'.format(cls))
    print(f'Processing class: {cls} - {len(os.listdir(cls))} samples')
    for fpath in glob.glob(os.path.join(data_root, cls, '*.avi')):  # Получаем путь к avi-файлам
        cls_idx = classes.index(cls)  # Индекс класса
        dataset.append((fpath, cls_idx))
```

```
Processing class: walking - 100 samples
Processing class: jogging - 100 samples
Processing class: running - 100 samples
Processing class: boxing - 100 samples
Processing class: handwaving - 100 samples
Processing class: handclapping - 99 samples
```

B [14]:
```
len(dataset)
```

Out[14]:
```
599
```

B [15]:
```
dataset[:5]
```

Out[15]:
```
[('./walking/person08_walking_d1_uncomp.avi', 0),
 ('./walking/person23_walking_d1_uncomp.avi', 0),
 ('./walking/person11_walking_d2_uncomp.avi', 0),
 ('./walking/person23_walking_d3_uncomp.avi', 0),
 ('./walking/person02_walking_d1_uncomp.avi', 0)]
```

B [16]:
```
type(dataset[0])
```

Out[16]:
```
tuple
```

Разделим датасет на тренировочную и тестовую выборки.

B [17]:
```
random.shuffle(dataset)  # Перемешиваем датасет, чтобы получить равномерно разные классы

test_count = math.floor(len(dataset)*0.4)

train_ds = dataset[:-test_count]
test_ds = dataset[-test_count:]
```

B [18]:
```
len(train_ds), len(test_ds)
```

Out[18]:
```
(360, 239)
```

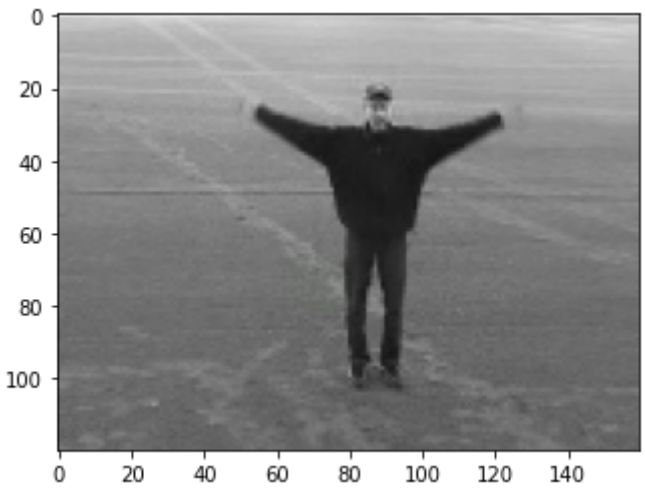
Визуализация кадра из видео

B [20]:
```
print(dataset[0][0])
print(dataset[0][1])
print(type(dataset[0][0]))
```

```
./handwaving/person23_handwaving_d2_uncomp.avi
4
<class 'str'>
```

B [21]:
```
# import skvideo.io

# Прочитаем наши данные. dataset[0][0] - путь к файлу.
path =  'person01_handwaving_d3_uncomp.avi'
videodata = skvideo.io.vread(dataset[0][0])
videodata = videodata.astype(np.float32) / 255.  # Приводим тензор к диапазону [0, 1]
                                                 # Изначально были значения int8 [0, 255]

# 3d-тензор первые измерения это кадр
print('videodata shape:', videodata.shape)
print('videodata shape:', videodata[50, ...].shape)
plt.imshow(videodata[50, ...]) # берем 50-й кадр.
plt.show()

# videodata shape: (477, 120, 160, 3)
# 3d-тензор (3d массив, где по одному измерению идут кадры)
# 477 - количество кадров
# 120, 160 - пространственные координаты
# 3 - количество каналов (3 несмотря на то, что видео чёрно-белое)
```

```
videodata shape: (658, 120, 160, 3)
videodata shape: (120, 160, 3)
```



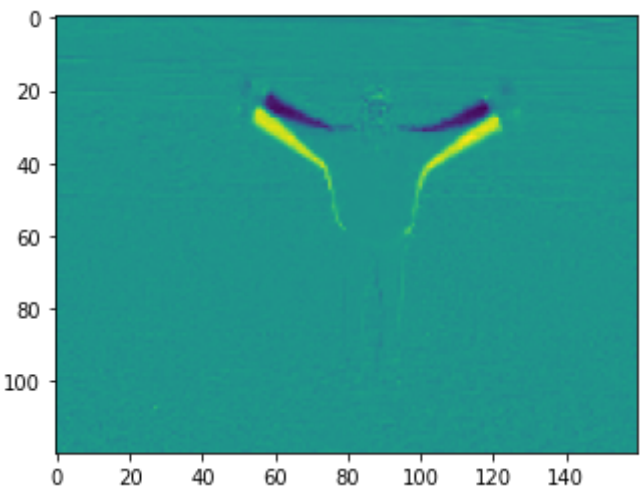
Визуализация "движения"

Предобработка.
Получаем движение пикселей. Например: из 3-го кадра вычитаем 2-й попиксельно, получаем движение в пиксельном пространстве.
Это не движение, а как поменялись цвета.

In [22]:
```
# Срез первого тензора, с 1-го кадра. 0-й не используем.
# Вычетаем из него тензор. Берём наоборот с 0-го по предпоследний. Не включая последний кадр.
# Получаем набор разностей: 1-0, 2-1, 3-2, ... .
# Сразу усредняем по 3-му измерению, т. к. у нас градация серого.
# keepdims=True - 3-е измерение превращается в 1 (вместо 3 бедет 1).
# Если keepdims=False - 3-е измерение исчезнет.
motion = np.mean(videodata[1:, ...] - videodata[:-1, ...], axis=3, keepdims=True)
print('motion shape:', motion.shape)
plt.imshow(motion[50, ..., 0])

# motion - тоже видеозапись, где на каждом кадре содержится движение
# Получаем в итоге тензор размерности (476, 120, 160, 1)
# 476 было 477
# 1 - цвета сократили до одного цвета.
```

```
motion shape: (657, 120, 160, 1)
```

Out[22]:
```
<matplotlib.image.AxesImage at 0x7f2e6b902e50>
```



В [23]:
```python
from tensorflow.keras.utils import Sequence

class KTHDataset(Sequence):
    # Подготовка датасета

    def __init__(self, data, max_frames, shuffle=False, batch_size=1, motion=True):
        """
        Конструктор

        Параметры:
        data - выборка [(path, label), ...]
        max_frames - количество кадров в видеоряде.
                     Чтобы видеоролики можно было объединить в batch,
                     к-во кадров у них должно быть одинаковым.
        shuffle - перемешать образцы.
        batch_size - размер batch-a.

        """
        self.data = data
        self.data_indices = list(range(len(data)))
        self.max_frames = max_frames
        self.shuffle = shuffle
        self.batch_size = batch_size
        if shuffle:
            random.shuffle(self.data_indices)

    def _get_videodata(self, path, motion=True):
        # Читаем данные
        videodata = skvideo.io.vread(path)
        videodata = videodata.astype(np.float32) / 255.

        # Количество повторений до необходимого размера
        n_repeats = (self.max_frames + 1) // videodata.shape[0]
        # Номер последнего кадра
        last_frame = (self.max_frames + 1) - (videodata.shape[0] * n_repeats)

        # приведение массивов к форме (max_frames, height, width, n_channels)
        # видео с меньшим кол-вом кадров - повторяется до необходимого размера
        # видео с большим - обрезается
        videodata = np.append(
            np.repeat(videodata, n_repeats, axis=0),  # Копировати n_repeats раз
            videodata[:last_frame],   # добавляемые значения
            axis=0
        )

        # "движение"
        if motion:
            videodata = np.mean(
                videodata[1:, ...] - videodata[:-1, ...],
                axis=3,
                keepdims=True
            )
        return videodata

    def __getitem__(self, idx, motion=True):
        # Поведение при обращении к объекту
        batch_indices = self.data_indices[idx * self.batch_size:(idx + 1) * self.batch_size]
        batch_X = []
        batch_y = []
        for i in batch_indices:
            path, label = self.data[i]
            videodata = self._get_videodata(path, motion)
            batch_X.append(videodata)
            batch_y.append(label)
        return np.array(batch_X), np.array(batch_y)

    def __len__(self):
        return math.ceil(len(self.data) / self.batch_size)
```

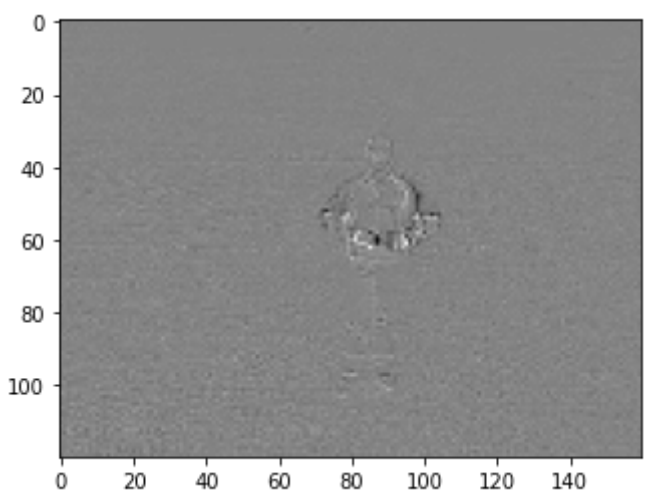
В [24]:
```python
import math

# len(train_ds),list(range(len(train_ds)))
# videodata_shape = 392
videodata_shape = 572
max_frames = 4000
batch_size = 400
n_repeats = (max_frames + 1)//videodata_shape
print(videodata_shape, max_frames, n_repeats, (max_frames+1) - videodata_shape*n_repeats)
print(math.ceil(videodata_shape / batch_size), videodata_shape / batch_size)
# 392 400 1 9
# 572 400 0 401
# 572 4000 6 569
```

```
572 4000 6 569
2 1.43
```

B [52]:
```
train = KTHDataset(train_ds, max_frames=400, shuffle=True, batch_size=4)
test = KTHDataset(test_ds, max_frames=400, shuffle=False, batch_size=4)
```

B [56]:
```
idx = np.random.randint(len(train))

batch_X, batch_y = train[idx]

# plt.imshow(batch_X[0][-200, ..., 0], cmap='gray')
plt.imshow(batch_X[0][-200, ..., 0], cmap='gray')
print(classes[batch_y[0]])
plt.show()
```

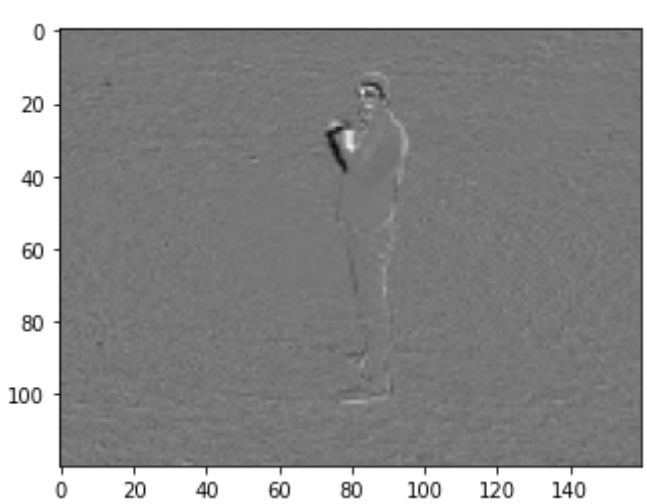
handclapping



B [57]:
```
test_without_motion = KTHDataset(test_ds, max_frames=400, shuffle=False, batch_size=1, motion=False)
```

B [58]:
```
idx = np.random.randint(len(train))

batch_X, batch_y = train[idx]

plt.imshow(batch_X[0][-200, ..., 0], cmap='gray')
print(classes[batch_y[0]])
plt.show()
```

boxing



Создание модели CNN

В [ ]:
```
# # CNN
# def build_model():
#     x = tf.keras.Input(shape=(None,120,160,1))

#     out = tf.keras.layers.Conv3D(32, (5, 5, 5), (1, 2, 2), padding='same', activation='relu')(x)
#     out = tf.keras.layers.MaxPool3D((1, 2, 2), padding='same')(out)

#     out = tf.keras.layers.Conv3D(64, (5, 5, 5), (1, 2, 2), padding='same', activation='relu')(out)
#     out = tf.keras.layers.MaxPool3D((1, 2, 2), padding='same')(out)

#     out = tf.keras.layers.Conv3D(64, (3, 3, 3), (1, 2, 2), padding='same', activation='relu')(out)
#     out = tf.keras.layers.MaxPool3D((1, 2, 2), padding='same')(out)

#     out = tf.keras.layers.Conv3D(64, (3, 3, 3), (1, 1, 1), padding='same', activation=None)(out)
#     out = tf.keras.layers.GlobalAveragePooling3D()(out)

#     out = tf.keras.layers.Dense(64, activation='relu')(out)
#     out = tf.keras.layers.Dropout(0.5)(out)  # to do добавляем Dropout
#     out = tf.keras.layers.Dense(6, activation='softmax')(out)

#     return tf.keras.Model(inputs=x,outputs=out,name='func_model')

# model = build_model()

# model.summary()
```

В [59]:
```
# Используем 3d-свёртки
# (5, 5, 5) = 5x(5x5) в окно попадает 5 кадров и пространственный кусочек 5x5
# Этим окном пробегаемся по всему 3d массиву
# (1, 2, 2) - страйд. 1 - не уменьшаем разрешение по временному измерению.
# 2, 2 - понижаем пространственное разрешение в 2 раза
model = tf.keras.Sequential([
    # на выходе 32 канала, ядро
    tf.keras.layers.Conv3D(32, (5, 5, 5), (1, 2, 2), padding='same', activation='relu'),
    # 3d-pooling
    tf.keras.layers.MaxPool3D((1, 2, 2), padding='same'),
    tf.keras.layers.Conv3D(64, (5, 5, 5), (1, 2, 2), padding='same', activation='relu'),
    tf.keras.layers.MaxPool3D((1, 2, 2), padding='same'),
    tf.keras.layers.Conv3D(64, (3, 3, 3), (1, 2, 2), padding='same', activation='relu'),
    tf.keras.layers.MaxPool3D((1, 2, 2), padding='same'),
    tf.keras.layers.Conv3D(64, (3, 3, 3), (1, 1, 1), padding='same', activation=None),

    # GlobalAveragePooling3D - все карты признаков агрегирует в одно число!
    # GlobalAveragePooling3D - делает усреднение (reduce)
    # по пространственным и временным измерениям (см. ниже)
    # Получится 64 числа (64d-вектор).
    tf.keras.layers.GlobalAveragePooling3D(),

    tf.keras.layers.Dense(64, activation='relu'),

    # tf.keras.layers.Dropout(0.5),  # Добавляем Dropout

    tf.keras.layers.Dense(6, activation=None),
])
```

В [60]:
```
# # Вход в модель. Добавляем batch измерение (Свёрточные сети любят batch).
# inp = motion[None, ...]
# out = model(inp)

# # Смотрим на размерность выходного тензора
# print('Input shape:', inp.shape)
# print('Output shape:', out.shape)
```

Подготовка к обучению

В [61]:
```
NUM_EPOCHS = 3
LEARNING_RATE = 0.001  # 5e-4

# SparseCategoricalCrossentropy - разреженная категориальная кроссэнтропия
# from_logits=True - так как мы не поставили функцию активации softmax,
# loss сам должен накрутить softmax сверху
model.compile(
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    # loss='categorical_crossentropy',
    optimizer=tf.keras.optimizers.Adam(LEARNING_RATE))

# Используем tenzorbord и пишем summary в дирректорию 'logs/exp1'
writer = tf.summary.create_file_writer('logs/model')

# Не указываем размер batch-а так как он равен 1.
# Так как у нас простая имплементация.
```

B [62]:
```
if 0:
    !mkdir models
```

B [63]:
```
checkpoint_filepath = './models/checkpoint'

model_checkpoint_callback = tf.keras.callbacks.ModelCheckpoint(
    filepath=checkpoint_filepath,
    save_weights_only=True,
    monitor='val_loss',
    mode='min',
    save_best_only=True)
```

Цикл обучения модели

B [64]:
```
model.fit(train, validation_data=test, epochs=NUM_EPOCHS, callbacks=[model_checkpoint_callback])
```

```
Epoch 1/3
90/90 [==============================] - 377s 4s/step - loss: 1.6028 - val_loss: 1.1945
Epoch 2/3
90/90 [==============================] - 374s 4s/step - loss: 1.1108 - val_loss: 0.8032
Epoch 3/3
90/90 [==============================] - 373s 4s/step - loss: 0.8920 - val_loss: 0.8119
```

Out[64]:
```
<keras.callbacks.History at 0x7f2d1c33fc90>
```

B [36]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv3d (Conv3D)             (None, None, None, None,  4032
                              32)

 max_pooling3d (MaxPooling3D  (None, None, None, None,  0
 )                            32)

 conv3d_1 (Conv3D)           (None, None, None, None,  256064
                              64)

 max_pooling3d_1 (MaxPooling  (None, None, None, None,  0
 3D)                          64)

 conv3d_2 (Conv3D)           (None, None, None, None,  110656
                              64)

 max_pooling3d_2 (MaxPooling  (None, None, None, None,  0
 3D)                          64)

 conv3d_3 (Conv3D)           (None, None, None, None,  110656
                              64)

 global_average_pooling3d (G  (None, 64)               0
 lobalAveragePooling3D)

 dense (Dense)               (None, 64)                4160

 dropout (Dropout)           (None, 64)                0

 dense_1 (Dense)             (None, 6)                 390

=================================================================
Total params: 485,958
Trainable params: 485,958
Non-trainable params: 0
_________________________________________________________________
```

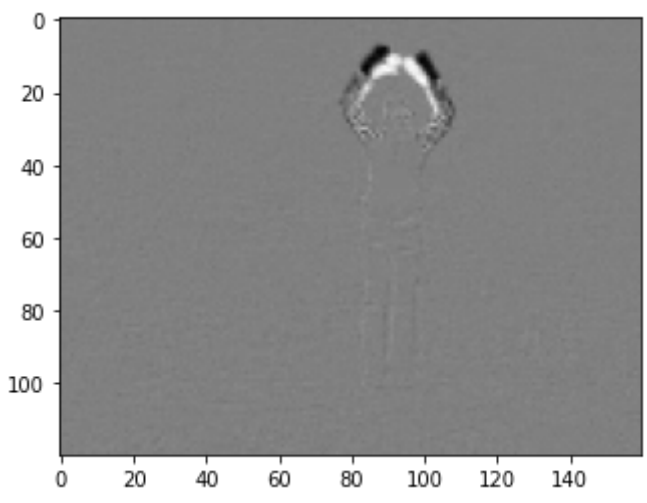
B [65]:
```
import pickle

filename = 'finalized_model_1.sav'
pickle.dump(model, open(filename, 'wb'))
```

```
INFO:tensorflow:Assets written to: ram://a5cfa810-716b-46e6-939d-f479b4023736/assets
```

Тестирование обученной модели

B [71]:
```
idx = np.random.randint(len(test))

batch_X, batch_y = test[idx]

plt.imshow(batch_X[0][200, ..., 0], cmap='gray')

out = model(batch_X)[0]
cls_pred = np.argmax(out.numpy())

print('True class:', classes[batch_y[0]])
print('Predicted class:', classes[cls_pred])
plt.show()
```

```
True class: handwaving
Predicted class: handwaving
```



B [69]:
```

```

B [73]:
```
idx = np.random.randint(len(test))

batch_X, batch_y = test[idx]

plt.imshow(batch_X[0][200, ..., 0], cmap='gray')

out = model(batch_X)[0]
cls_pred = np.argmax(out.numpy())

print('True class:', classes[batch_y[0]])
print('Predicted class:', classes[cls_pred])
plt.show()
```

```
True class: jogging
Predicted class: walking
```