ДОДАТОК А

Вихідний код програми

```
# Classes for effective generation of batches for iteration
# with possibility to use transforms for data augmentation
class Transform():
  def __init__(self, apply_flips, apply_crops, crop_coef=0.05):
    self.flip = apply_flips and bool(np.random.randint(2))
    self.apply_crops = apply_crops
    if apply_crops:
      total_coef = np.random.random() * crop_coef
      self.l1 = np.random.random() * total_coef
      self.r1 = 1 - (total_coef - self.l1)
      self.l2 = np.random.random() * total_coef
      self.r2 = 1 - (total_coef - self.l2)

  def apply(self, image, is_test=False):
    if self.flip:
      image = np.flip(image, axis=2)
    if self.apply_crops:
      image = np.rollaxis(image, 0, 3)
      h, w = image.shape[:2]
      image = cv2.resize(image[int(self.l1 * h):int(self.r1 * h),
int(self.l2 * w):int(self.r2 * w), :], (w, h))
      image = np.rollaxis(image, 2, 0)
      if is_test:
        image[image > 0.5] = 1
        image[image <= 0.5] = 0
    return image


class BatchGeneratorInternal():
  def __init__(self, batch_gen):
    self.batch_gen = batch_gen
    self.p = np.arange(len(batch_gen.image_files))
    self.transforms = [Transform(batch_gen.apply_flips,
batch_gen.apply_crops) for i in range(len(batch_gen.image_files))]
    if batch_gen.shuffle:
      self.p = np.random.permutation(len(batch_gen.image_files))

  def __getitem__(self, num):
    num = self.batch_gen.start_position + num * self.batch_gen.batch_size
    to = min(num + self.batch_gen.batch_size,
len(self.batch_gen.image_files))
    res_input, sizes = zip(*[self.batch_gen.get_input(self.p[i]) for i in
range(num, to)])
    res_output =
np.array([self.transforms[i].apply(self.batch_gen.get_output(self.p[i],
size), True) for i, size in zip(range(num, to), sizes)])
    res_input = np.array([self.transforms[i].apply(input) for i, input in
zip(range(num, to), res_input)])
    return res_input, res_output
```

```
  def __len__(self):
    return len(self.batch_gen)


class BatchGenerator(tf.keras.utils.Sequence):
  def __init__(self, dataset, dataset_path, batch_size, model_params,
         shift=0, shuffle=False, apply_flips=False, apply_crops=False):
    self.image_files = []
    self.dataset_path = dataset_path
    self.batch_size = batch_size
    self.model_params = model_params
    self.shift = shift
    self.shuffle = shuffle
    self.apply_flips = apply_flips
    self.apply_crops = apply_crops
    self.start_position = 0
    self.max_len = 10 ** 9
    def update_nan(x):
      from math import isnan
      if isnan(x):
        return -1
      return x
    for game_dir, clip_dir, cnt_frames in dataset:
      data = pd.read_csv(f'{dataset_path}/{game_dir}/{clip_dir}/Label.csv',
sep=',')
      rows = [(f'{game_dir}/{clip_dir}/{row["file name"]}',
           row['visibility'],
           update_nan(row['x-coordinate']),
           update_nan(row['y-coordinate']),
           row['status']) for it, row in data.iterrows()]
      for frame_id in range(shift, len(rows) -
model_params.n_consecutive_frames + 1, model_params.n_consecutive_frames):
        self.image_files.append(rows[frame_id:frame_id +
model_params.n_consecutive_frames])
    self.bgi = BatchGeneratorInternal(self)

  def get_image(self, filename):
    img = np.array(Image.open(f'{self.dataset_path}/{filename}'))
    h, w = img.shape[:2]
    img = cv2.resize(img, model_params.get_wh()) / 255
    return np.rollaxis(img, 2, 0), (h, w)

  def get_target(self, cx, cy, initial_w, initial_h, r):
    h, w = self.model_params.input_height, self.model_params.input_width
    if cx < 0 or cy < 0:
      return np.zeros((h, w))
    cx *= w / initial_w
    cy *= h / initial_h
    heatmap = np.zeros((h, w))
    for y in range(max(0, int(cy - r) - 1), min(h, int(cy + r) + 1)):
      for x in range(max(0, int(cx - r) - 1), min(w, int(cx + r) + 1)):
        if ((x - cx) ** 2 + (y - cy) ** 2 <= r ** 2):
          heatmap[y][x] = 1
    return heatmap
```

```
  def get_input(self, num):
    images = []
    sizes = []
    for row in self.image_files[num]:
      image, (h, w) = self.get_image(row[0])
      images.append(image)
      sizes.append((h, w))
    return np.concatenate(images, axis=0), sizes

  def get_output(self, num, sizes):
    return np.array([self.get_target(row[2], row[3], w, h, 2) for row, (h,
w) in zip(self.image_files[num], sizes)])

  def __len__(self):
    return min(self.max_len, (len(self.image_files) - self.start_position +
self.batch_size - 1) // self.batch_size)

  def on_epoch_end(self):
    self.start_position += self.max_len
    if self.start_position >= len(self.image_files):
      self.bgi = BatchGeneratorInternal(self)
      self.start_position = 0

  def __getitem__(self, num):
    return self.bgi.__getitem__(num)

  def set_max_len_as_percent(self, percent):
    self.max_len = len(self.image_files) // percent + 1


# Algorithm for linking separate results to trajectory
class Triplet:
  def __init__(self, trajectory_x, trajectory_y, center_time, found_balls,
window_size, d_error):
    self.trajectory_x = trajectory_x
    self.trajectory_y = trajectory_y
    self.center_time = center_time
    self.found_balls = found_balls
    self.window_size = window_size
    self.d_error = d_error
    self.supports = self.get_supports(center_time, found_balls,
window_size, d_error)

  def get_ball_coordinates(self, t):
    def evaluate(trajectory, t):
      return trajectory[0] * t ** 2 + trajectory[1] * t + trajectory[2]
    return (evaluate(self.trajectory_x, t), evaluate(self.trajectory_y, t))

  def update_by_supports(self, found_balls):
    ts = [p[0] for p in self.supports if p[1] != -1]
    xs = [found_balls[p[0]][p[1]][0] for p in self.supports if p[1] != -1]
    ys = [found_balls[p[0]][p[1]][1] for p in self.supports if p[1] != -1]
    return Triplet(np.polyfit(ts, xs, 2), np.polyfit(ts, ys, 2),
self.center_time, self.found_balls, self.window_size, self.d_error)
```

```
  def get_support(self, t, found_balls, d_error):
    p = self.get_ball_coordinates(t)
    support = (d_error, -1)
    for i, c in enumerate(found_balls[t]):
      d = get_distance(p, c)
      if d <= d_error:
        support = min(support, (d, i))
    return support[1]

  def get_supports(self, center_time, found_balls, window_size, d_error):
    supports = []
    for i in range(max(0, center_time - window_size), min(len(found_balls),
center_time + window_size + 1)):
      j = self.get_support(i, found_balls, d_error)
      supports.append((i, j))
    return supports

  def get_score(self, found_balls, d_error):
    score = 0
    for i, j in self.supports:
      if j == -1:
        score += d_error ** 2
      else:
        score += get_distance(self.get_ball_coordinates(i),
found_balls[i][j]) ** 2
    return score

  def get_start_time(self):
    return min([support[0] for support in self.supports if support[1] != -
1])

  def get_finish_time(self):
    return max([support[0] for support in self.supports if support[1] != -
1])

  def get_cnt_supports(self):
    return sum([support[1] != -1 for support in self.supports])


def create_triplet(i1, j1, i2, j2, i3, j3, center_time, found_balls,
window_size, d_error):
  return Triplet(my_polyfit([i1, i2, i3], [found_balls[i1][j1][0],
found_balls[i2][j2][0], found_balls[i3][j3][0]]),
           my_polyfit([i1, i2, i3], [found_balls[i1][j1][1],
found_balls[i2][j2][1], found_balls[i3][j3][1]]),
           center_time, found_balls, window_size, d_error)


def get_triplet_distance(triplet1, triplet2):
  l1, r1 = triplet1.get_start_time(), triplet1.get_finish_time()
  l2, r2 = triplet2.get_start_time(), triplet2.get_finish_time()
  if r2 <= r1:
    for i in range(r2, r1 + 1):
      if triplet1.supports[i] != triplet2.supports[i]:
        return math.inf
```

```
    return 0
  return min([get_distance(triplet1.get_ball_coordinates(i),
        triplet2.get_ball_coordinates(i)) for i in range(r1, r2 + 1)])


class Trajectory:
  def __init__(self, triplets, n_frames, n_extend=0):
    self.triplets = triplets
    self.ids_by_frame = [[] for i in range(n_frames)]
    for i, triplet in enumerate(triplets):
      for j in range(triplet.get_start_time(), triplet.get_finish_time() +
1):
        self.ids_by_frame[j].append(i)
    for (id1, triplet1), triplet2 in zip(enumerate(triplets[:-1]),
triplets[1:]):
      finish1 = triplet1.get_finish_time()
      start2 = triplet2.get_start_time()
      for i in range(finish1 + 1, min(start2, finish1 + 1 + n_extend)):
        self.ids_by_frame[i].append(id1)
      for i in range(max(finish1 + 1, start2 - n_extend), start2):
        self.ids_by_frame[i].append(id1 + 1)
    for i in range(len(self.ids_by_frame)):
      self.ids_by_frame[i].sort()

  def get_ball_coordinates(self, time):
    frame_id = int(time + 0.5)
    if frame_id >= len(self.ids_by_frame) or
len(self.ids_by_frame[frame_id]) == 0:
      return False, (0, 0, 0)
    x = 0
    y = 0
    for i in self.ids_by_frame[frame_id]:
      p = self.triplets[i].get_ball_coordinates(time)
      x += p[0]
      y += p[1]
    return True, (x / len(self.ids_by_frame[frame_id]),
            y / len(self.ids_by_frame[frame_id]),
            self.ids_by_frame[frame_id])


def get_trajectory(found_balls, max_dist_by_frame, d_error, window_size=15,
max_size_misses=10, minimum_trajectory_size=10, triplet_th=4,
update_by_supports=False):
  def get_triplets(found_balls):
    triplets = []
    for i in range(1, len(found_balls) - 1):
      for prev_index in range(len(found_balls[i - 1])):
        for current_index in range(len(found_balls[i])):
          if get_distance(found_balls[i][current_index], found_balls[i -
1][prev_index]) > max_dist_by_frame:
            continue
          for next_index in range(len(found_balls[i + 1])):
            if get_distance(found_balls[i][current_index], found_balls[i +
1][next_index]) > max_dist_by_frame:
              continue
```

```
            triplet = create_triplet(i - 1, prev_index, i, current_index, i
+ 1, next_index, i, found_balls, window_size, d_error)
            while True:
              score = triplet.get_score(found_balls, d_error)
              l = min([p[0] for p in triplet.supports if p[1] != -1])
              c = min([(abs(p[0] - i), p[0]) for p in triplet.supports if
p[1] != -1])[1]
              r = max([p[0] for p in triplet.supports if p[1] != -1])
              shift = triplet.supports[0][0]
              extended_triplet = create_triplet(*triplet.supports[l -
shift], *triplet.supports[c - shift], *triplet.supports[r - shift], i,
found_balls, window_size, d_error)
              if update_by_supports:
                new_extended_triplet =
extended_triplet.update_by_supports(found_balls)
                extended_triplet = new_extended_triplet
              if score <= extended_triplet.get_score(found_balls, d_error):
                break
              triplet = extended_triplet
            if update_by_supports:
              triplet = triplet.update_by_supports(found_balls)
            if triplet.get_finish_time() - triplet.get_start_time() + 1 >=
triplet_th:
              triplets.append(triplet)
    return triplets

  def get_best_trajectory(start, finish, triplets, maximum_delay=2):
    triplets = [triplet for triplet in triplets if start <=
triplet.get_start_time() and triplet.get_finish_time() <= finish]
    triplets.sort(key=lambda triplet: triplet.center_time)
    best_start = (-1, -1)
    for i, triplet in enumerate(triplets):
      if triplet.get_start_time() <= start + maximum_delay:
        best_start = max(best_start, (triplet.get_cnt_supports(), i))
    start_id = best_start[1]
    parents = [-1 for i in range(len(triplets))]
    distances = [math.inf for i in range(len(triplets))]
    distances[start_id] = 0
    for i, triplet in enumerate(triplets):
      current_start_time = triplet.get_start_time()
      current_finish_time = triplet.get_finish_time()
      for j in range(i - 1, -1, -1):
        if triplets[j].center_time + window_size + max_size_misses + 1 <
current_start_time:
          break
        if triplets[j].get_finish_time() + max_size_misses + 1 <
current_start_time:
          continue
        if triplets[j].get_finish_time() >= current_finish_time:
          continue
        if triplets[j].get_start_time() >= current_start_time:
          continue
        d = get_triplet_distance(triplets[j], triplet)
        if distances[j] + d < distances[i]:
          distances[i] = distances[j] + d
```

```
            parents[i] = j
      best_finish = (math.inf, -1)
      for i, triplet in enumerate(triplets):
        if triplet.get_finish_time() + maximum_delay >= finish:
          best_finish = min(best_finish, (distances[i], i))
      finish_id = best_finish[1]
      result = []
      while finish_id != -1:
        result.append(triplets[finish_id])
        finish_id = parents[finish_id]
      result.reverse()
      return result
    triplets = get_triplets(found_balls)
    cnt = [0 for i in range(len(found_balls))]
    for triplet in triplets:
      for i in range(triplet.get_start_time(), triplet.get_finish_time() +
1):
        cnt[i] += 1
    start = 0
    trajectory = []
    while True:
      while start < len(found_balls) and cnt[start] == 0:
        start += 1
      if start == len(found_balls):
        break
      finish = start
      total = 0
      while finish < len(found_balls):
        total += cnt[finish]
        if finish - max_size_misses >= start:
          total -= cnt[finish - max_size_misses]
        if total == 0:
          break
        finish += 1
      while finish >= len(found_balls) or cnt[finish] == 0:
        finish -= 1
      if finish - start + 1 >= minimum_trajectory_size:
        for triplet in get_best_trajectory(start, finish, triplets):
          trajectory.append(triplet)
      start = finish + 1
    trajectory = Trajectory(trajectory, len(found_balls))
    return trajectory
```

ДОДАТОК Б

Відомість кваліфікаційної роботи

Позначення	Найменування	Дод. відомості
	Текстові документи	
1.	Пояснювальна записка	68 с.
	Інші документи	
2.	Презентаційні матеріали	19 плакатів

.	.	Прізвище та ініціали.	Підп.	Дата			
Розробив		Асландуков М.М.			Використання глибинної нейронної мережі для покращення якості відео тенісних матчів	Шифр групи	Код напр./спец.
Перевірив		Бодянський Є.В.				СШІм-20-2	122
Н.контр.		Малєєва І.А.				ХНУРЕ кафедра ШІ	
Затв.		Філатов В.О.					