Śledzenie obiektów z wykorzystaniem granulacji i głębokich sieci neuronowych

Kacper Majchrzak, Mateusz Mazur

24.05.2024

1 Artykuł bazowy

Granulated deep learning and Z-numbers in motion detection and object recognition

W artykule poruszono problematykę detekcji ruchu, rozpoznawania obiektów i opisu scen z wykorzystaniem głębokiego uczenia się w kontekście obliczeń granularnych. Ponieważ głębokie uczenie jest wymagające obliczeniowo, autorzy proponują zastosowanie obliczeń granularnych w celu zredukowania wymagań obliczeniowych.

2 Ładowanie modelu SSD

Nasze prace rozpoczęliśmy od załadowania modelu SSD, który jest wykorzystywany do detekcji obiektów, oraz zdefiniowania funkcji pomocniczych do przetwarzania obrazów.

Ten etap sfinalizowaliśmy przetestoaniem modelu na przykładowym obrazie.

[5]:
```
saved_model = load_model(MODEL_NAME)
```

```
INFO:tensorflow:Saver not created because there are no variables in the graph to
restore
```

[6]:
```
# Loading default model signature.
model = saved_model.signatures['serving_default']
```

[8]:
```
labels = load_labels(LABELS_NAME)

print(f'Labels ({len(labels)}):')
for i in range(1,6):
    print(f'{i}: {labels[i]}')
print('...')
```

```
Labels (80):
1: person
2: bicycle
3: car
4: motorcycle
```

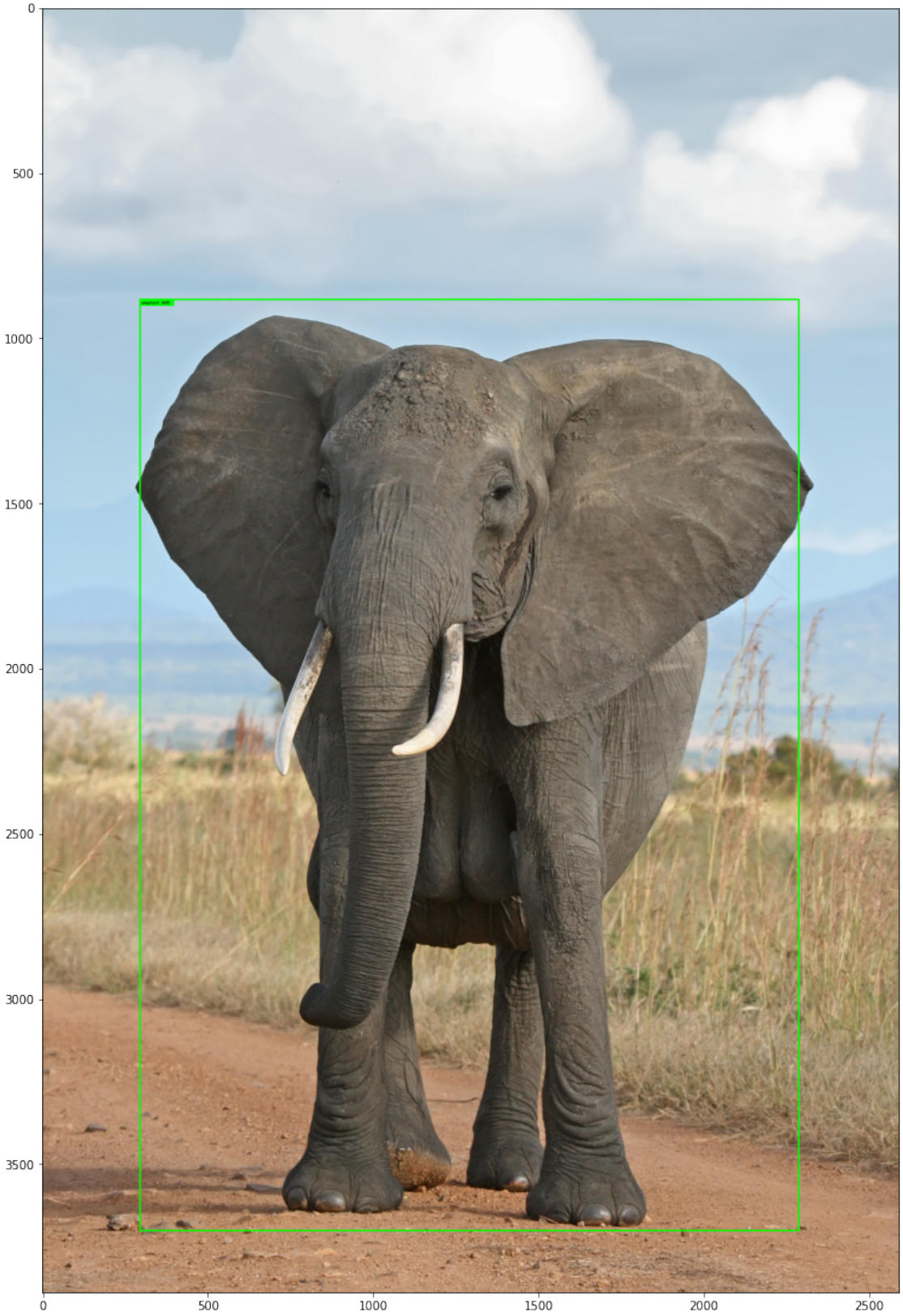
```
5: airplane
…
```

[9]:
```python
def detect_objects_on_image(image, model):
    image = np.asarray(image)
    input_tensor = tf.convert_to_tensor(image)
    # Adding one more dimension since model expect a batch of images.
    input_tensor = input_tensor[tf.newaxis, ...]

    output_dict = model(input_tensor)

    num_detections = int(output_dict['num_detections'])
    output_dict = {
        key:value[0, :num_detections].numpy()
        for key,value in output_dict.items()
        if key != 'num_detections'
    }
    output_dict['num_detections'] = num_detections
    output_dict['detection_classes'] = output_dict['detection_classes'].
 ↪astype(np.int64)

    return output_dict
```

[10]:
```python
def recognition_of_moving_objects(img, model=model):
    image_np = np.array(img)
    detections = detect_objects_on_image(image_np, model)
    return detections
```

[12]:
```python
# Test the model on the elephant image.
image_np = np.array(Image.open('data/elephant.jpg'))
detections = detect_objects_on_image(image_np, model)
image_with_detections = draw_detections_on_image(image_np, detections, labels)
plt.figure(figsize=(20, 20))
plt.imshow(image_with_detections)
```

[12]:
```
<matplotlib.image.AxesImage at 0x7ad4185fbf70>
```


0
500
1000
1500
2000
2500
3000
3500
0
500
1000
1500
2000
2500

[14]:
```python
def calculate_differencing(prev_f, f, vis=False):
    # Calculate the difference between the previous and the current frame.
    diff = cv2.absdiff(prev_f, f)
    if vis:
        plot([diff], ['diff'])
    return diff
```

3 Granulacja obrazów

W następnym etapie zaimplementowaliśmy dwie funkcje do granulacji obrazów: - `granulation` - obliczanie granulacji obrazu najprostszą metodą - `granulation_gt` - obliczanie granulacji obrazu metodą *quad tree decomposition*

3.1 Obliczanie granulacji najprostszą metodą

W tej metodzie obraz dzielimy na kwadraty o zadanym rozmiarze, a następnie dla każdego kwadratu obliczamy średnią wartość pikseli w nim zawartych.

[15]:
```python
def granulation(image, window_size=WINDOW_SIZE, vis=False):
    height, width = image.shape[:2]
    granules = np.zeros_like(image)
    for j in range(0, height, window_size):
        for i in range(0, width, window_size):
            granule = image[j : min(j+window_size, height), i:
 ↪min(i+window_size, width)]
            granules[j : min(j+window_size, height), i:min(i+window_size,␣
 ↪width)] = np.mean(granule, axis=(0, 1))

    if vis:
        plot([image, granules], ['Original', 'Granulated'],␣
 ↪subtitle='Granulation')

    return granules
```

3.2 Obliczanie granulacji metodą *quad tree decomposition*

W tej metodzie obraz dzielimy na 4 równe części, a następnie dla każdej z nich obliczamy średnią wartość koloru oraz błąd na podstawie histogramu pikseli w niej zawartych. Jeśli błąd w danej części jest większy niż zdefiniowany próg, to dzielimy tę część na kolejne cztery równe części, aż do osiągnięcia zdefiniowanej głębokości.

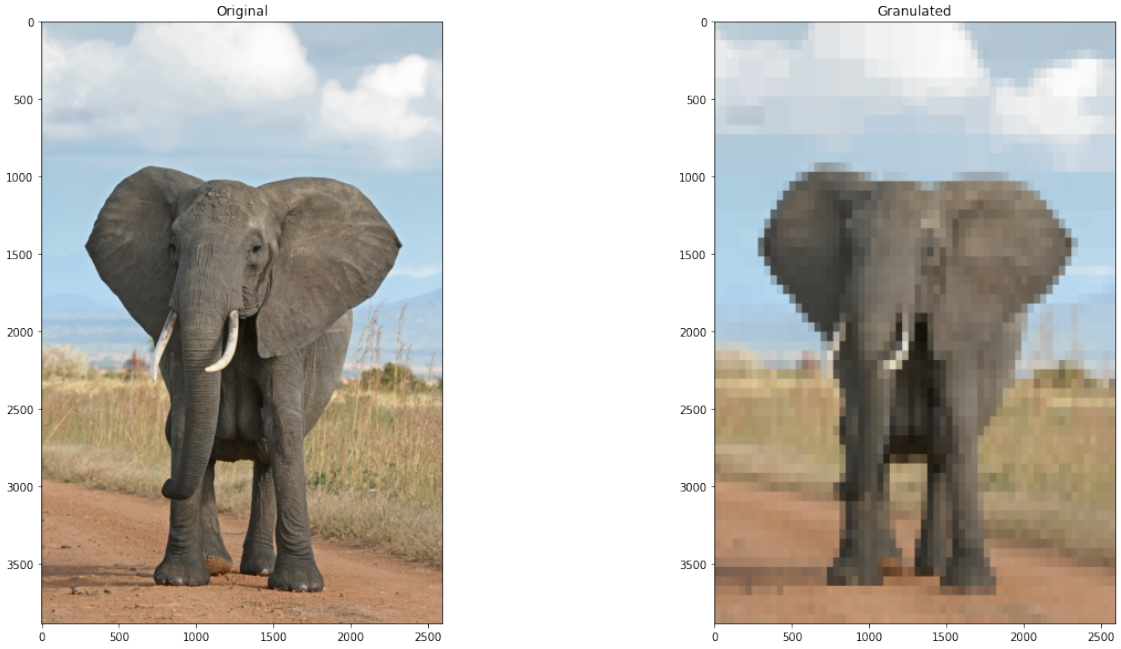
[17]:
```python
def granulation_gt(frame, visual=False, max_depth=6):
    frame_pil = Image.fromarray(frame)
    tree = Quadtree(frame_pil, max_depth=max_depth)
    image, granules = tree.render_at_depth(max_depth)
    image = np.array(image)
    if visual:
```

```
        plot([frame, image], ['Original', 'Granulated'], subtitle='Granulation')
    return image, granules
```

Przykład użycia:

[18]:
```
img = cv2.cvtColor(cv2.imread('data/elephant.jpg'), cv2.COLOR_BGR2RGB)
img_granulated, granules = granulation_gt(img)
fig, axs = plt.subplots(1, 2, figsize=(20, 10))
axs[0].imshow(img)
axs[0].set_title('Original')
axs[1].imshow(img_granulated)
axs[1].set_title('Granulated')
```

[18]:
```
Text(0.5, 1.0, 'Granulated')
```



4 Detekcja ruchomych obiektów

W ostatnim etapie chcieliśmy połączyć granulację obrazów z detekcją ruchu. Podczas eksperymentów zwróciliśmy uwagę na liczne problematyczne aspekty implementacyjne, które nie zostały poruszone w artykule.

Musimy zatemm stwierdzić, że nasze prace nie przyniosły oczekiwanych rezultatów, a granulacja nie spełniła swojej roli w procesie detekcji ruchu z wykorzystaniem głębokiego uczenia.

4.1 Dlaczego granulacja nie spełnia swojej roli w badanym przypadku?

Analizowana przez nas publikacja przedstawiała podejście, które opierało się na wykorzystaniu sieci neuronowej używającej granulacji obrazów oraz, docelowo, transfer-learningu. Przestawiona przez autorów warstwa granulacyjna miała za zadanie zastąpić (lub poprzedzać) warstwę konwolucyjną w sieci SSD, co miało pozwolić na wykorzystanie granulacji w procesie detekcji obiektów.

Niestety, w naszym przypadku, zastosowanie granulacji nie przyniosło oczekiwanych rezultatów. Z naszych eksperymentów wyciągamy następujące wnioski:

- **Brak możliwośći wykorzystania transfer-learningu** - Wynika to z faktu, że podstawy konwolucyjne sieci SSD zostały wytrenowane na obrazach, które nie były granulowane. W związku z tym, sieć nie jest w stanie wykryć obiektów na granulowanych obrazach. Możliwe jest oczywiście przetrenowanie sieci na granulowanych obrazach, jednak wymagałoby to ogromnej ilości czasu i zasobów, a przede wszystkim prawdoopodobnie nie przyniosłoby oczekiwanych rezultatów.
- **Granulacja powoduje utratę szczegółowości informacji** - Jedną z wad granulacji przedstawia poniższy przykład. Po lewej mamy obraz oryginalny, przeskalowany w dół, a po prawej obraz granulowany. Jak widać, granulacja powoduje utratę informacji (np. nie widzimy oka słonia), co w przypadku sieci neuronowych może prowadzić do znacznych problemów z uczeniem, a nauczone na granulowanych obrazach modele mogą być mniej dokładne.
- **Przekazywanie nieregularnych granulek do sieci neuronowych** - Warto również zauważyć, że granulek o nieregularnych kształtach, nie jesteśmy w stanie przekazać konwolucyjnym warstwom w sieci neuronowej (możliwe jest jedynie przekazanie kwadratów, prostokątów, itp. jako odpowiedniki pikseli). Wykorzystanie nieregularnych granulek miało wpłynąć na zmniejszenie ilości obliczeń, ale okazuje się być to prawdopodobnie niemożliwe.
- **Brak różniczkowalności** - Należy również dodać, że granulacja jest operacją nieliniową, co sprawia, że nie jest ona różniczkowalna. W związku z tym, nie jest możliwe stworzenie trenowalnych warstw granulacyjnych w sieciach neuronowych.

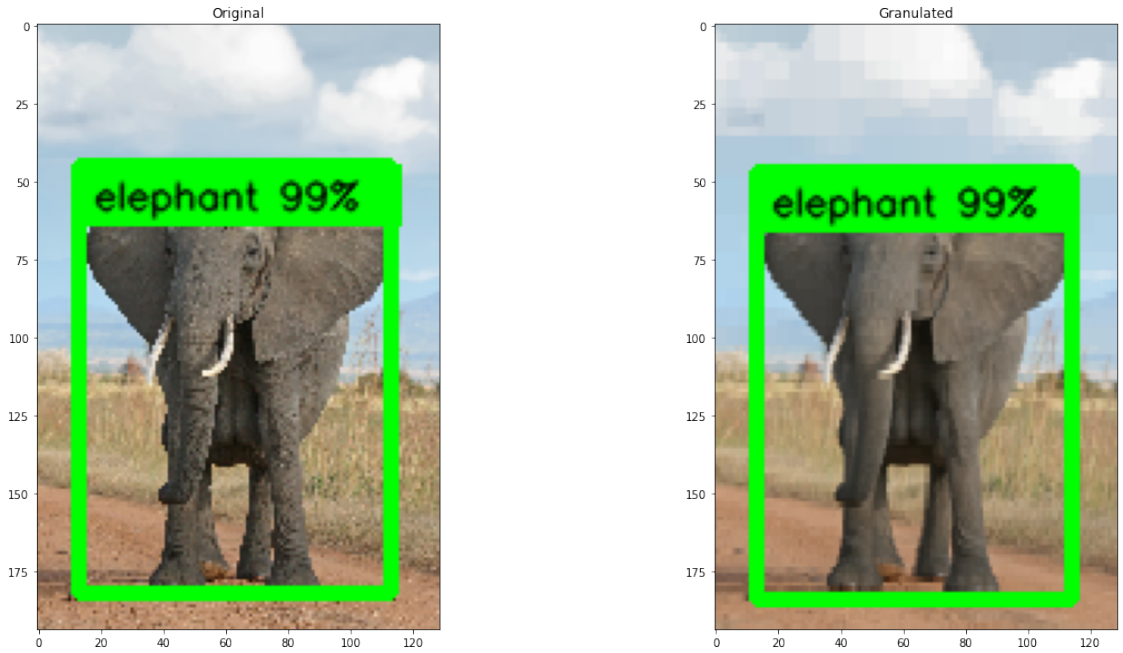
$$g(a) + g(b) \neq g(a + b)$$

[19]:
```
d = recognition_of_moving_objects(img)
d_g = recognition_of_moving_objects(img_granulated)
```

[20]:
```
factor = 0.05
img_resized = cv2.resize(img, (int(img.shape[1]* factor), int(img.shape[0]*␣
 ↪factor)))
d_resized = recognition_of_moving_objects(img_resized)
img_resized_granuled, granules_resized = granulation_gt(img, max_depth=7)
img_resized_granuled = cv2.resize(img_resized_granuled,␣
 ↪(int(img_resized_granuled.shape[1]* factor), int(img_resized_granuled.
 ↪shape[0]* factor)))
d_resized_granuled = recognition_of_moving_objects(img_resized_granuled)
```

[21]:
```
fig, axs = plt.subplots(1, 2, figsize=(20, 10))
axs[0].imshow(draw_detections_on_image(img_resized, d_resized, labels))
axs[0].set_title('Original')
axs[1].imshow(draw_detections_on_image(img_resized_granuled,␣
 ↪d_resized_granuled, labels))
axs[1].set_title('Granulated')
```

[21]:
```
Text(0.5, 1.0, 'Granulated')
```



4.2 Alternatywne podejście z wykorzystaniem granulacji

Aby wykorzystać granulację w procesie śledzenia obiektów, zdecydowaliśmy się na zastosowanie granulacji obrazów w celu wykrycia ruchu na obrazie. Granulacja obrazów pozwala na zredukowanie ilości informacji w obrazie, co pozwala na szybsze przetwarzanie obrazu.

Zaimplementowaliśmy następujący algorytm:

- obliczamy granulację ramki obecnej,
- obliczamy różnicę między granulacją ramki obecnej a granulacją ramki poprzedniej,
- wykrywamy ruch na obrazie na podstawie różnicy granulacji,
- obszary, w których wykryto ruch, są wycinane z obecnej ramki, a następnie są przekazywane do sieci neuronowej, która dokona klasyfikacji obiektów

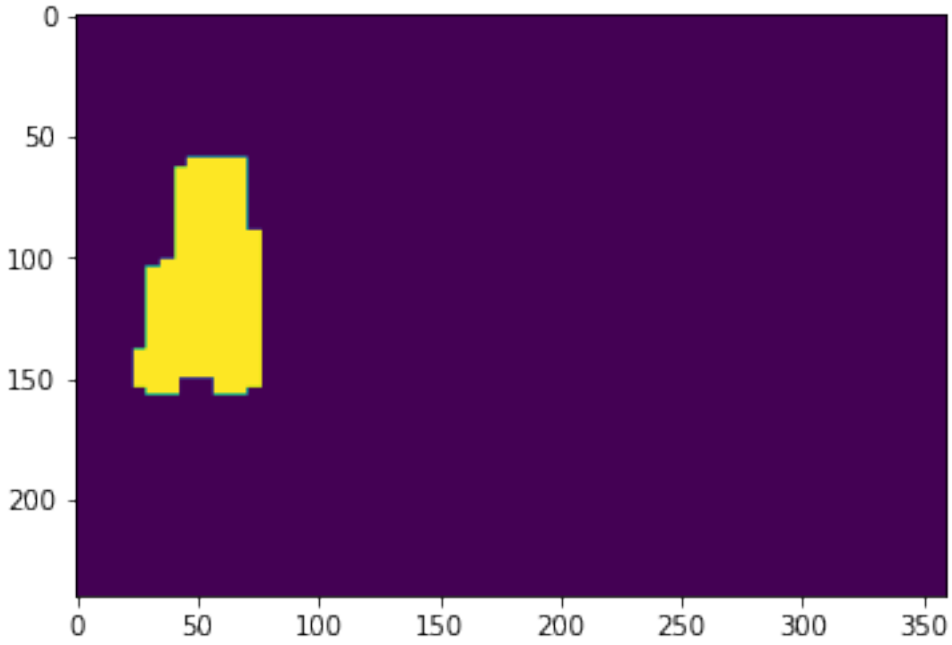
Podejście to pozwala na zwiększenie szybkości przetwarzania obrazu, a także na zredukowanie ilości informacji przekazywanej do sieci neuronowej, co pozwala na zwiększenie szybkości działania sieci oraz poprawę jakości klasyfikacji.

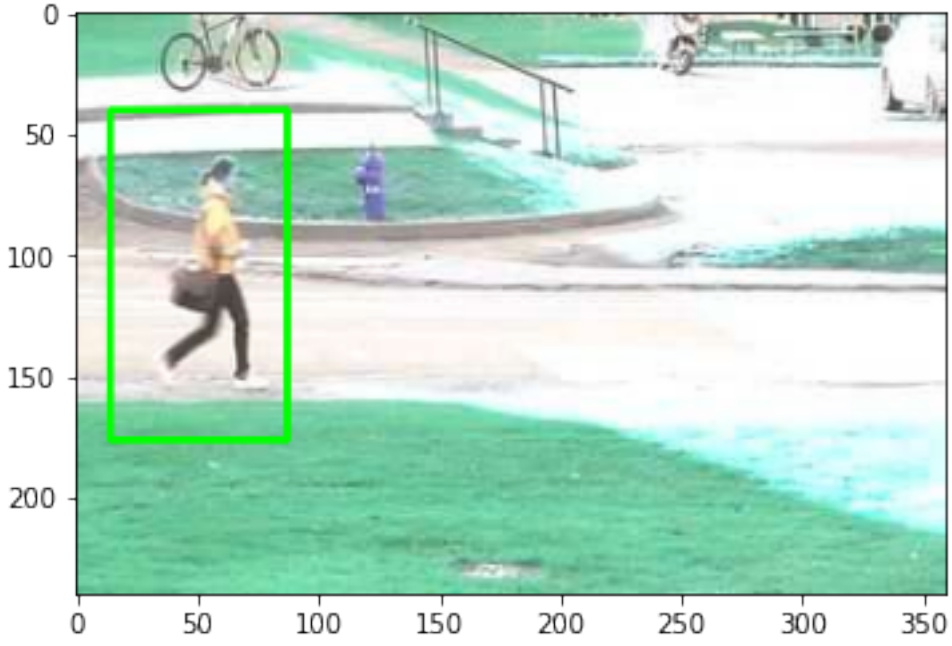
[24]:
```
PATH = 'data/pedestrian'

f1 = cv2.imread(f'{PATH}/input/in%06d.jpg' % (roi_start+30))
f2 = cv2.imread(f'{PATH}/input/in%06d.jpg' % (roi_start+30+1))
```

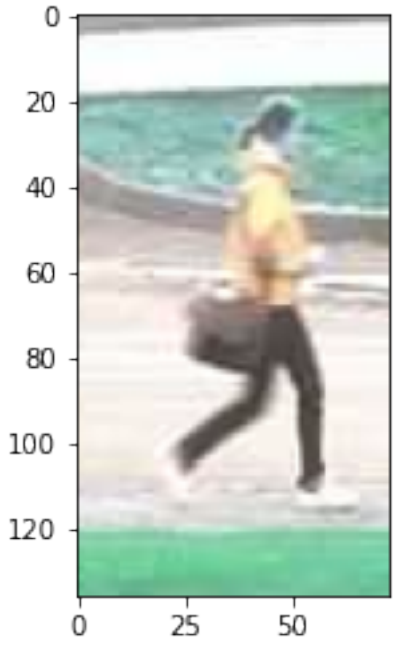
[26]:
```
g1, _ = granulation_gt(f1)
g2, _ = granulation_gt(f2)
```

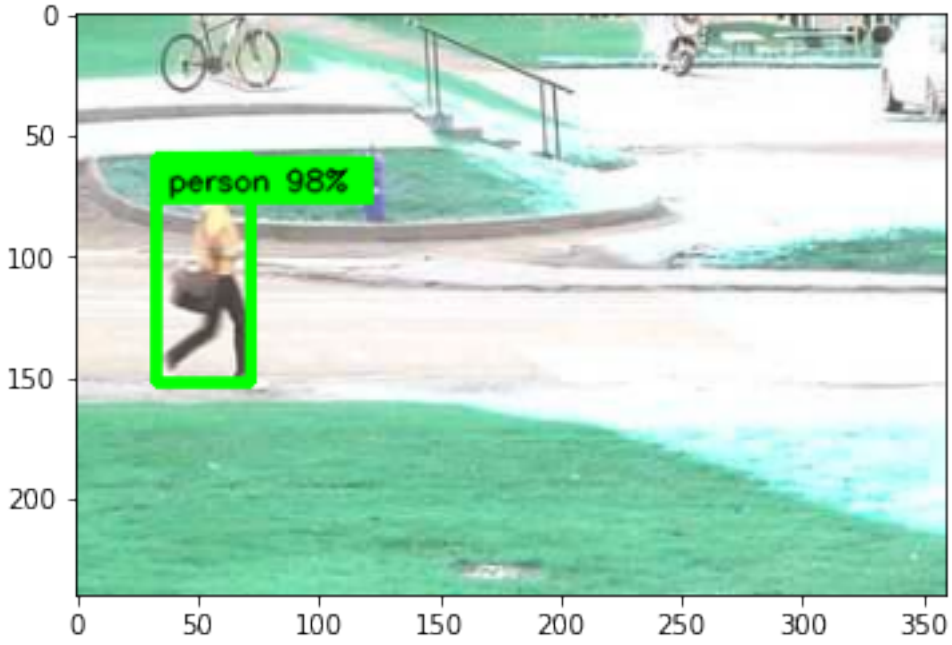
Poniższa komórka pokazuje kolejne etapy działania algorytmu.

[27]:
```
f1_out, dets = extract_moving_objects(f2, g1, g2, tresh=10, vis=True)
```


0
50
100
150
200
0
50
100
150
200
250
300
350


0
50
100
150
200
0
50
100
150
200
250
300
350





Następnie zaimplementowaliśmy funkcję `track`, która wykorzystuje granulację obrazów do detekcji ruchu na obrazie, oraz `track_base`, która nie wykorzystuje granulacji obrazów.

Przykładowe wyniki

[36]:
```
track_base('data/pedestrian', roi_start=roi_start, roi_end=roi_end) # tracking␣
↪bez granulacji, z SSD analizującym cały obraz
```

[37]:
```
track('data/pedestrian', roi_start=roi_start, roi_end=roi_end) # tracking z␣
↪granulacją, z SSD analizującym podobszary, w których wykryto ruch
```

[38]:
```
track_base('data/highway', roi_start=roi_start_hw, roi_end=roi_end_hw)
```

[39]:
```
track('data/highway', roi_start=roi_start_hw, roi_end=roi_end_hw, pad_factor=1)
```

Klasyfikowane obiekty są wyświetlane na obrazie wraz z informacją o ich klasie oraz prawdopodobieństwie przynależności do danej klasy. Na pierwszy rzut oka widać, że klasyfikacja obiektów jest poprawna, a granulacja pozwala na zwiększenie dokładności klasyfikacji. Niestety, widzimy również, że wraz z pojawieniem się większej ilości obiektów na obrazie, FPS spada, co jest związane z koniecznością przetworzenia większej ilości obiektów przez sieć neuronową.

Jak widzimy, granulacja obrazów może być użyteczna w procesie detekcji obiektów, jednak nie w sposób, w jaki zostało to zaprezentowane w analizowanej publikacji.

4.3 Podsumowanie

Nasze eksperymenty z połączeniem granulacji obrazów i detekcji ruchu przy użyciu głębokiego uczenia nie przyniosły oczekiwanych rezultatów. Przedstawione w analizowanej publikacji podejście, które zakładało zastosowanie granulacji przed lub zamiast warstwy konwolucyjnej w sieci SSD, okazało się problematyczne. Kluczowe wnioski z naszych badań to:

- **Problemy z transfer-learningiem**: Sieci SSD wytrenowane na standardowych obrazach nie potrafią efektywnie działać na obrazach granulowanych. Przetrenowanie sieci na granulowanych obrazach wymagałoby znacznych zasobów i czasu, a uzyskane wyniki prawdopodobnie byłyby niezadowalające.
- **Utrata szczegółowości**: Granulacja powoduje utratę istotnych szczegółów na obrazach, co utrudnia naukę i dokładność sieci neuronowych. Przykładem jest niewidoczne oko słonia na granulowanym obrazie.
- **Problemy z nieregularnymi granulkami**: Nieregularne granulki nie mogą być łatwo przetwarzane przez warstwy konwolucyjne, co ogranicza możliwości optymalizacji obliczeniowej.
- **Brak różniczkowalności**: Granulacja jest operacją nieliniową i nieróżniczkowalną, co uniemożliwia tworzenie trenowalnych warstw granulacyjnych w sieciach neuronowych.

4.3.1 Alternatywne podejście

Podjęliśmy próbę zastosowania granulacji do wykrywania ruchu na obrazach w inny sposób. Nasze podejście polegało na:

- Obliczaniu granulacji obecnej i poprzedniej ramki,
- Porównywaniu granulacji między ramkami w celu wykrycia ruchu,
- Przekazywaniu wykrytych obszarów ruchu do sieci neuronowej do klasyfikacji obiektów.

To podejście okazało się skuteczniejsze, pozwalając na szybsze przetwarzanie obrazów i zwiększenie dokładności klasyfikacji. Jednak wraz ze wzrostem liczby obiektów na obrazie, wydajność (FPS)

spadała, co wynikało z konieczności przetwarzania większej liczby obiektów przez sieć neuronową.

4.4 Wnioski

Granulacja obrazów może być przydatna w procesie detekcji ruchu, jednak nie w formie zaprezentowanej w analizowanej publikacji. Alternatywne podejście, które łączy granulację z wykrywaniem ruchu, a następnie klasyfikacją obiektów, pokazuje pewne korzyści, ale również ujawnia ograniczenia związane z wydajnością przy dużej liczbie obiektów.