Sieć Kohonena do kompresji obrazów

Inteligentne przetwarzanie danych

Mateusz Szczęsny, 233266 Dawid Wójcik, 233271

Łódź, 2020 r.

1 Cel zadania

Celem zadania jest utworzenie programu implementującego sieć neuronową Kohonena, a następnie wykorzystanie go do przeprowadzenia badania polegającego na kompresji obrazów. Aplikacja powinna być dostosowana do kompresji obrazów w 8-bitowej skali szarości (256 odcieni szarości) oraz dla uproszczenia powinny być brane pod uwagę następujące parametry startowe:

- rozmiar ramki kompresji w pikselach, która również determinują liczbę neuronów w warstwie wejściowej,
- liczba neuronów w warstwie wejściowej,
- krok uczący,
- liczba losowych ramek obrazu, które posłużą do wytrenowania zaimplementowanej sieci,
- liczba epok przez które sieć będzie uczona.

Dla każdorazowej kompresji program powinien działać na pojedynczym obrazie w kali szarości oraz raportować wartości następujących parametrów:

- osiągnięty współczynnik kompresji, równy stosunkowi całkowitej liczby bitw wymaganych do zakodowania oryginalnego obrazu do całkowitej liczby bitów niezbędnych do zakodowania obrazu skompresowanego,
- wartość miary *PSNR* (szczytowy stosunek sygnału do szumu) dla obrazu oryginalnego oraz skompresowanego

Powyższe wartości mogą zostać wyliczone za pomocą następujących wzorów:

$$PSNR = 10\log_{10}(\frac{255^2}{MSE}); MSE = \frac{1}{N^2}\sum_{i=0}^{N}\sum_{j=0}^{N}(x_{i,j} - \hat{x}_{i,j})^2$$

2 Wstęp teoretyczny

Sieci Kohonena są jednymi z podstawowych typów samoorganizujących się sieci. Sieci Kohonena stanowią synonim całej grupy sieci, w których uczenie odbywa się metodą samoorganizującą typu konkurencyjnego. Polega ona na podawaniu na wejścia sieci sygnałów, a następnie wybraniu w drodze konkurencji zwycięskiego neuronu, który najlepiej odpowiada wektorowi wejściowemu. Dokładny schemat konkurencji i późniejszej modyfikacji wag może mieć różną postać. Wyróżnia się wiele podtypów sieci opartych na konkurencji, które różnią się dokładnym algorytmem samoorganizacji.

3 Implementacja

W celu implementacji zakładanego programu wykorzystano język programowania Python oraz bibliotek *OpenCV* do obsługi plików graficznych oraz textitnumpy do wykonywania operacji na kolekcjach danych. Dodatkowo wykorzystano techniki programowania obiektowego, aby kolejne etapu działania sieci były uwidocznione na tle innych operacji. Klasa *Neuron* wyróżnia 5 głównych metod, których zadaniem kolejno jest: przygotowanie danych oraz inicjalizacja parametrów wejściowych, wykonanie procesu nauczania, wykonanie kompresji obrazu, wykonanie dekompresji obrazu, utworzenie raportu z wykonanej pracy.

Klasa *Neuron* pozwala również na zdefiniowanie niezbędnych parametrów wejściowych opisanych szczegółowo w rozdziale pierwszym. Poniższy kod prezentuje przykładową inicjalizację sieci.

```
network = Network(
    epochs=150,
    learning_step=0.2,
    neurons_count=8,
    image="img/lena.png",
    image_width=512,
    image_height=512,
    frame_size=4,
    how_many_training_frames=512,
    logger=Logger(path="log/"),
    new_image_name=f"log/compressed_{int(time.time())}.png",
    )
```

Etap *przygotowania* polega na wczytaniu zdjęcia do kompresji, oraz przerobienia go na tablicę ramek, których wielkość określiliśmy w trakcie inicjalizacji sieci.

```
def prepare(self):
    self.image = Image(
        file=self.image_file,
        target_width=self.image_width,
        target_height=self.image_height,
    )
    self.image.convert_to_frames(self.frame_size)
    self.generate_frames_queue() # wygenerowanie tablicy
    indeksow ramek treningowych
```

Kolejnym etapem jest uczenie sieci za pomocą załadowanych danych. Poniższy kod przedstawia główną pętlę programu odpowiadającą za uczenie sieci. Zadaniem tej części programu jest przeanalizowanie każdej ramki treningowej oraz każdego neuronu oraz wybranie tego, którego wagi są najbardziej dopasowane do sygnału wejściowego (best matching unit - bmu). Następnie program dokonuje korekcji wag zwycięskiego neuronu tak by najlepiej odwzorować wejściową strukturę danych. Poniższy kod prezentuje instrukcje odprawiające za etap *uczenia*.

```
def learn(self):
    for _ in range(self.epochs):
        for training_frame_index in self.
    training_frames_queue:
            matching_unit = []
            for neuron_index, neuron in enumerate(self.
    neurons):
                distance = vector_distance(
                    self.image.get_normal_frame(
    training_frame_index),
                    normalize_vector(neuron.weights),
                )
                matching_unit.append(
                    [training_frame_index, neuron_index,
    distance,]
                )
        bmu = min(matching_unit, key=lambda x: x[2])
        self.neurons[bmu[1]].weights = normalize_vector(
            self.neurons[bmu[1]].weights
        ) + self.learning_step * (
            self.image.get_normal_frame(bmu[0])
            - normalize_vector(self.neurons[bmu[1]].weights)
        )
```

Po uprzednim wyuczeniu sieci program może przystąpić do etapu *kompresji*. Wynikiem tej części jest uzyskanie tablicy neuronów, w której każdy kolejny odpowiada odpowiedniej ramce (jest do niej najbardziej dopasowany). Dzięki temu uzyskujemy prototyp złożony z neuronów najbardziej odpowiadających naszemu wejściowemu obrazowi. Poniższy kod prezentuje instrukcje odpowiedzialne za ten etap.

```
def compress(self):
    self._prototypes = []
    for frame_index, normalized_frame in enumerate(
        self.image.normalized_framed_image
    ):
        potential_winners = []
        for neuron in self.neurons:
            potential_winners.append(
                (
                    frame_index,
                    neuron,
                    vector_similarity(normalized_frame,
    neuron.normalized_weights,),
                )
            )
        winner = max(potential_winners, key=lambda x: x[2],)
        self._prototypes.append(winner[1])
```

Ostatnim etapem odpowiedzialnym za przetworzenie obrazu jest *dekompresja*. Jej efektem jest otrzymanie skompresowanego obrazu. Program na podstawie utworzonej w poprzedniej części listy wzorcowych neuronów oblicza nowe ramki, z których następnie zostanie złożony docelowy obraz. Poniższy kod jest reprezentacją instrukcji odpowiedzialnych za dekompresję.

```
def decompress(self):
    new_frames = []
    for i, prototype_neuron in enumerate(self._prototypes):
        decompressed_frame = (
            prototype_neuron.normalized_weights * self.image.
    image_brightness[i]
        ) # calculate new frame based on
        new_frames.append(
            np.array(
                decompressed_frame.reshape(self.frame_size,
    self.frame_size),
                dtype=np.uint8,
            )
        )
    new_frames = np.concatenate(new_frames, axis=1)

    sorted_frames = []
    # for every image row
    for x in range(0, new_frames.shape[1], self.frame_size *
    self.image.col_count):
        # reshape calculated frames to match original image
        sorted_frames.append(
            new_frames[:, x : x + (self.frame_size * self.
    image.col_count)]
        )

    self.final_image = np.concatenate(sorted_frames)
    cv2.imwrite(self.new_image_name, self.final_image)
```

Ostatnim zadaniem programu jest obliczenie wszystkich założonych parametrów, ich opis widnieje w rozdziale *Cel zadania*. Wyliczone wartości zostają zapisane do pliku w celu dalszej analizy. Szczegółowe instrukcje dostępne są w plikach źródłowych załączonych do sprawozdania. Plik *network.py* -> klasa *Network* -> metoda *report*.

4 Przebieg eksperymentu

W celu przetestowania zaimplementowanej sieci poddaliśmy ją badaniu polegającemu na kompresji 2 obrazów. Obrazy w skali szarości zostałyp rzedstawione na Rysunku 1.

(a) lena.png

(b) boat.png

Rysunek 1: Obrazy poddane analizie

Każdy z obrazów został poddany kompresji przy użyciu tego samego zestawu parametrów, które prezentują się następująco.

- **4** neurony w warstwie wyjściowej; rozmiary ramek: **4, 8, 16, 32, 64**
- **8** neuronów w warstwie wyjściowej; rozmiary ramek: **4, 8, 16, 32, 64**
- **16** neuronów w warstwie wyjściowej; rozmiary ramek: **4, 8, 16, 32, 64**
- **32** neurony w warstwie wyjściowej; rozmiary ramek: **4, 8, 16, 32, 64**
- **74** neuronów w warstwie wyjściowej; rozmiary ramek: **4, 8, 16, 32, 64**

Reszta parametrów startowych była stała. Jedynym wyjątkiem była ilość ramek treningowych, która w przypadku rozmiarów ramek 32 i 64 wynosiła kolejno 200 i 10. Wartości pozostałych parametrów prezentują się następująco.

$$liczba\ epok = 150$$

$$krok\ uczacy = 0.05$$

Po każdej operacji program wygeneruje podsumowanie zawierający wszystkie parametry startowe oraz parametry potrzebne do dalszej analizy zagadnienia tj. błąd średnio-kwadratowy, współczynnik kompresji oraz wartość PSNR. Przykładowy raport prezentuje poniższy kod.

```
==================================================
Image: img/lena.png
Frame: 8px x 8px
Neurons count: 4
Learning step: 0.05
Epochs count: 150
Number of training frames: 512
Int bits: 24
Int bytes: 3
Float bits: 24
Float bytes: 3
Source image size in bits: 2097152
Source image size in bytes: 262144
Compression ratio: 98.6988
Bits count of original image: 2097152
Bits count of compressed image: 21248
Mean Square Error: 66.6988
Peak Signal-to-Noise Ratio: 29.8896 dB
==================================================
```

5 Wyniki

Wszystkie wyniki są dołączone do kodów źródłowych w folderze *log*. Poniższe zestawienie prezentuje wyniki otrzymane dla kompresji zdjęcia *boat.png* dla różnych parametrów startowych.

Compression ratio		Liczba nuronów w warstwie wyjściowej				
		4	8	16	32	64
Rozmiar ramki uczącej	2px x 2px	6,40	5,33	4,57	4,00	3,55
	4px x 4px	25,54	21,25	18,16	15,81	13,93
	8px x 8px	98,70	80,31	66,06	53,89	42,67
	16px x 16px	256,00	170,67	107,79	64,00	35,93
	32 px x 32px	154,57	80,31	41,17	20,90	10,54

Rysunek 2: Wynikowy współczynnik kompresji

Rysunek 2 przestawia uzyskane wartości współczynnika kompresji dla analizy wejściowego obrazu. Możemy zaobserwować zmniejszanie się wartości współczynnika wraz ze wzrostem liczby neuronów w warstwie wejściowej (niezależnie od wielkości ramki).

MSE		Liczba nuronów w warstwie wyjściowej				
		4	8	16	32	64
Rozmiar ramki uczącej	2px x 2px	45,12	42,60	40,50	35,72	68,33
	4px x 4px	99,54	56,37	61,19	71,89	73,11
	8px x 8px	103,28	96,19	70,63	73,89	101,12
	16px x 16px	103,05	102,87	103,59	87,80	104,94
	32 px x 32px	96,71	89,46	106,18	87,42	89,05

Rysunek 3: Wynikowy błąd średnio-kwadratowy

Rysunek 3 prezentuje wartość błędu średnio-kwadratowego, dla konkretnych parametrów startowych. Możemy tutaj zaobserwować wzrost błędu wraz ze wzrostem wielkości ramki treningowej. Korekcja innych parametrów wejściowych takich jak ilość ramek uczących mogłaby wpłynąć na zmniejszenie błędu.

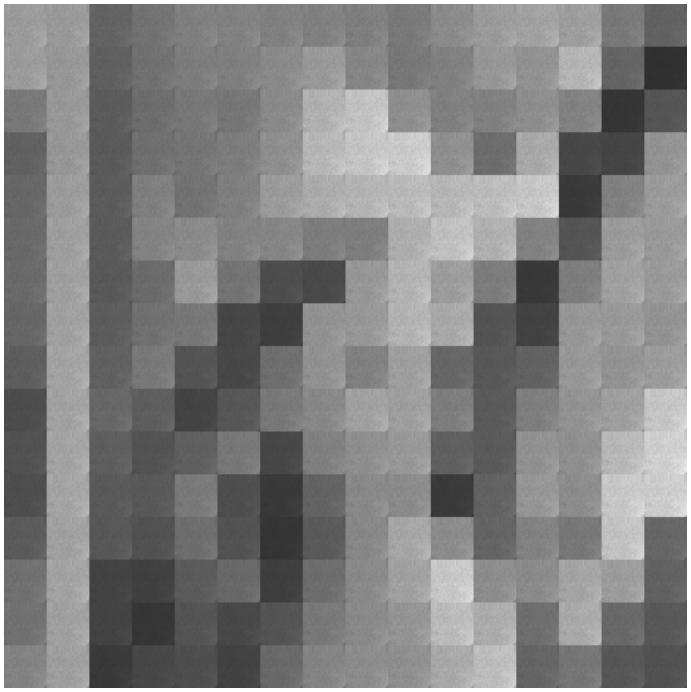
PSNR [dB]		Liczba nuronów w warstwie wyjściowej				
		4	8	16	32	64
Rozmiar ramki uczącej	2px x 2px	31,59	31,84	32,06	32,60	29,78
	4px x 4px	28,15	30,62	30,26	29,56	29,49
	8px x 8px	27,99	28,30	29,64	29,45	28,08
	16px x 16px	28,00	28,01	27,98	28,70	27,92
	32 px x 32px	28,28	28,61	27,87	28,71	28,63

Rysunek 4: Wynikowa wartość PSNR

Rysunek 4 przedstawia wartości dla szczytowego stosunku sygnału do szumu (PSNR). Uzyskane wyniki zawierają się w przedziale $< 27.87, 32.6 >$.

Rysunek 5: lena.png; ramka 4px; 32 neurony wyjściowe

Rysunek 5, Rysunek 6a oraz Rysunek 6b przedstawiają skompresowane obrazy otrzymane przy pomocy zaimplementowanej sieci. Na potrzeby sprawozdania obrazy zostały dodatkowo przeskalowane do rozmiaru $[256\,x\,256]px$, aby mogły zmieścić się w dokumencie.



(a) lena.png; ramka 32px; 64 neurony wyjściowe

(b) boat.png; ramka 8px; 8 neuronów wyjściowych

Rysunek 6: Obrazy skompresowane

6 Wnioski

1. Wykorzystanie sieci Kohonena jest możliwe do rozwiązania problemu kompresji obrazu, dzięki zdolności samoorganizacji sieci - czyli umiejętności adaptacji do nieznanych wcześniej danych wejściowych.

2. Wzrost ilość neuronów użytych w warstwie wyjściowej sieci, powoduje spadek współczynnika kompresji (dla badanej próby).

3. Sieć z uwagi na swoją architekturę wymaga przechowywania znacznej ilości danych. Poza oryginalnym sygnałem program musi przechowywać dane przygotowane do analizy, neurony oraz tablice indeksów. Powoduje to znaczne wykorzystanie pamięci przez program. Niektóre dane potrzebne do analizy można przekształcać tylko na potrzeby obliczeń, tak aby nie przechowywać ich stanu, jednak to rozwiązanie spowoduję podniesienie i tak dużej liczby operacji matematycznych na danych.

4. Parametr PSNR nie pozwala na jednoznaczne określenie jakości wynikowego obrazu. Przykładem może być obraz boat.png, który dla 8 neuronów oraz ramek wielkości [2px x 2px] oraz [4px x 4px] przyjmuje kolejno wartości 31,84dB i 30,62dB. Pomimo niewielkiej różnicy (porównując z innymi wskazaniami), jakość odbieranego obrazu jest różna, ponieważ innym decydującym parametrem jest współczynnik kompresji analizowanego obrazu.

5. Ramki dużej wielkości w połączeniu z dużą ilością neuronów warty wyjściowej znacząco zwiększają złożoność obliczeń sieci. Dla bardzo dużych danych wejściowych i parametrów przydatne może okazać się wykorzystanie do obliczeń zasobów karty graficznej w celu zwiększenia wydajności metody.