

Augmentacja danych obrazowych dla modeli głębokich

Maria Deka, Julia Girtler, Patrycja Piechocka

2 czerwca 2025

Plan prezentacji

- 1 Wprowadzenie
- 2 Zbiór danych i architektura sieci
- 3 Techniki augmentacji
- 4 Eksperymentalny i wyniki
- 5 Bibliografia

Wprowadzenie

Cel projektu

Implementacja technik augmentacji danych, aby zwiększyć różnorodność zbioru treningowego dla sieci głębokich.

Projekt został zaimplementowany w języku Python z wykorzystaniem biblioteki **PyTorch**. Do obsługi danych obrazowych użyto pakietu **torchvision**, a wykresy generowano przy użyciu biblioteki **matplotlib**.

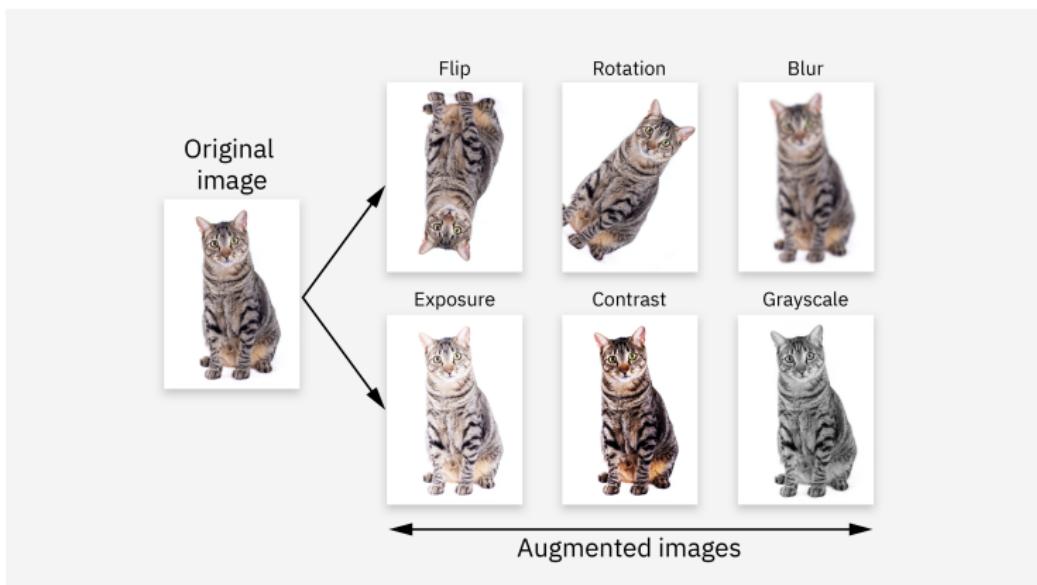
Augmentacja danych

Czym jest augmentacja danych?

Jedna z najczęściej stosowanych technik zwiększania zdolności generalizacji modeli uczenia maszynowego, szczególnie w kontekście sieci głębokich.

- Polega na generowaniu sztucznych przykładów na podstawie danych treningowych.
- Pomaga ograniczyć zjawisko przeuczenia (*overfitting*) oraz zwiększa stabilność modelu.
- W danych obrazowych obejmuje m.in.:
 - transformacje geometryczne (obrót, przeskalowanie, odbicie lustrzane),
 - zmiany kolorów,
 - metody mieszania obrazów, np. *Mixup* i *CutMix*.

Augmentacja danych



Zbiór danych CIFAR-10

Opis zbioru

W projekcie wykorzystano popularny zbiór danych obrazowych *CIFAR-10*, zawierający łącznie 60000 kolorowych obrazów o rozdzielczości 32×32 piksele. Dane są podzielone na 10 klas odpowiadających różnym obiektom, takim jak: **samolot, samochód, ptak, kot, jeleń, pies, żaba, koń, statek, ciężarówka**

Podział danych:

- 50000 obrazów treningowych,
- 10000 obrazów testowych.

Zbiór danych CIFAR-10

airplane**automobile****bird****cat****deer****dog****frog****horse****ship****truck**

Architektura sieci neuronowej

- W projekcie wykorzystano konwolucyjną sieć neuronową (CNN), składającą się z dwóch głównych bloków:
 - Każdy blok zawiera warstwę konwolucyjną, funkcję aktywacji ReLU oraz warstwę poolingującą (max pooling).
 - Po części konwolucyjnej występują dwie w pełni połączone warstwy, z funkcją ReLU oraz warstwą wyjściową typu softmax.
- Sieć przyjmuje obrazy w formacie $32 \times 32 \times 3$ i zwraca wektor o 10 elementach odpowiadających klasom.

Model bazowy bez augmentacji

Poniższy fragment kodu przedstawia sposób przygotowania danych i zbudowania podstawowej architektury sieci bez wykorzystania dodatkowych technik augmentacji:

```
1  transform = transforms.Compose([
2      transforms.ToTensor(),
3      transforms.Normalize((0.4914, 0.4822,
4          0.4465),
5          (0.247, 0.243,
6          0.261))])
```

Klasyczne techniki augmentacji

Klasyczne metody augmentacji polegają na stosowaniu prostych transformacji geometrycznych i kolorystycznych, które nie zmieniają etykiety obrazu.

Najczęściej stosowane metody:

- odbicia lustrzane (flip),
- obrót o losowy kąt,
- przycięcia (crop),
- zmiana jasności lub kontrastu,
- przesunięcia i skalowania.

Techniki te są łatwe do zaimplementowania i skutecznie redukują przeuczenie, jednak mają ograniczoną zdolność do generowania bardziej złożonych wariantów danych.

Klasyczne metody augmentacji

W tej wersji modelu obraz został zmodyfikowany klasycznymi technikami augmentacji – poprzez losowe przycięcie z marginesem oraz odbicie lustrzane:

```
1  transform_train = transforms.Compose([
2      transforms.RandomCrop(32, padding=4),
3      transforms.RandomHorizontalFlip(),
4      transforms.ToTensor(),
5      transforms.Normalize((0.4914, 0.4822,
6                          0.4465),
7                           (0.247, 0.243,
8                           0.261))])
```

Augmentacja przez mieszanie obrazów

W ostatnich latach dużą popularność zdobyły metody augmentacji oparte na łączeniu (mieszaniu) dwóch lub więcej obrazów.

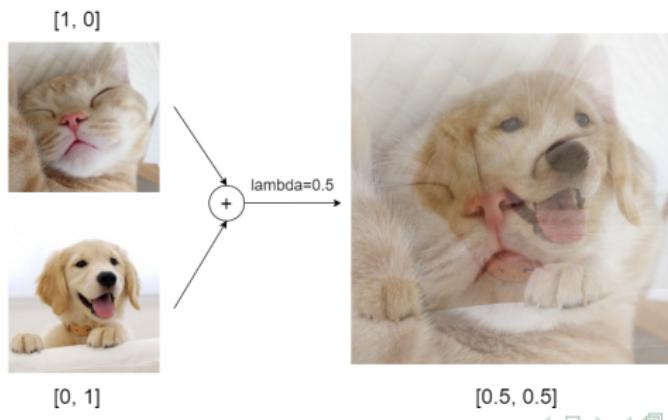
- Techniki te polegają na tworzeniu nowych przykładów poprzez:
 - kombinację pikseli,
 - łączenie fragmentów obrazu,
 - mieszanie cech semantycznych.
- Dodatkowo etykiety są mieszane proporcjonalnie, co sprzyja lepszej generalizacji.

Metoda Mixup

Mixup tworzy nowe dane przez liniową kombinację dwóch obrazów i ich etykiet.

$$x_{\text{mix}} = \lambda x_1 + (1 - \lambda) x_2$$

$$y_{\text{mix}} = \lambda y_1 + (1 - \lambda) y_2$$



Metoda Mixup

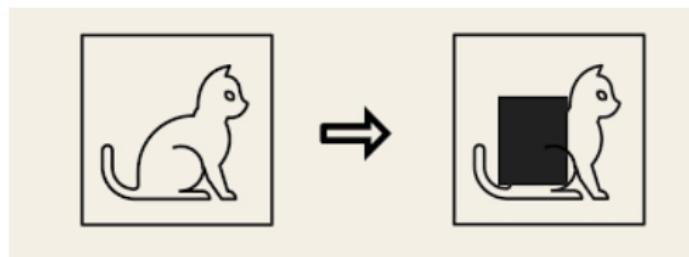
W tej wersji modelu zastosowano metodę Mixup, w której w każdej iteracji losowo łączenie dwóch przykładów (oraz ich etykiet) za pomocą współczynnika pozwala uzyskać sztuczne próbki treningowe o uśrednionych cechach i etykietach:

```
1 mixed_x = lam * x + (1 - lam) * x[index]
2 y_a, y_b = y, y[index]
3 loss = lam * criterion(output, y_a) + (1 -
    lam) * criterion(output, y_b)
```

Metoda Cutout

Cutout polega na zamaskowaniu losowego fragmentu obrazu — piksele w wybranym obszarze są zastępowane zerami lub szumem.

Etykieta pozostaje niezmieniona, co zmusza model do uczenia się z kontekstu i zapobiega przeuczeniu.



Przykład działania metody Cutout

Metoda Cutout

W tej odmianie modelu zastosowano augmentację Cutout, która polega na losowym zamaskowaniu fragmentu obrazu (nadpisaniu go zerami), zmuszając sieć do wychwytywania bardziej ogólnych i odpornych na brakujące regiony cech:

```
1     y1 = np.clip(y - self.size // 2, 0, h)
2     y2 = np.clip(y + self.size // 2, 0, h)
3     x1 = np.clip(x - self.size // 2, 0, w)
4     x2 = np.clip(x + self.size // 2, 0, w)
5
6     img[:, y1:y2, x1:x2] = 0.0
```

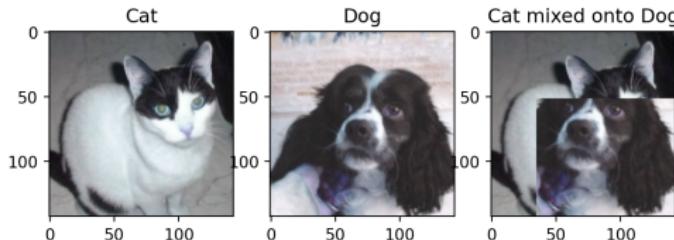
Metoda CutMix

CutMix polega na wycięciu fragmentu jednego obrazu i wklejeniu go w losowe miejsce drugiego.

Etykieta jest obliczana proporcjonalnie do powierzchni każdego z obrazów.

$$x_{\text{mix}} = x_1 \odot M + x_2 \odot (1 - M)$$

$$y_{\text{mix}} = \lambda y_1 + (1 - \lambda) y_2$$



Metoda CutMix

W tej odmianie modelu zastosowano augmentację CutMix, która polega na losowym wycięciu prostokątnego fragmentu obrazu i zastąpieniu go fragmentem z innego obrazu z tej samej partii (batcha).

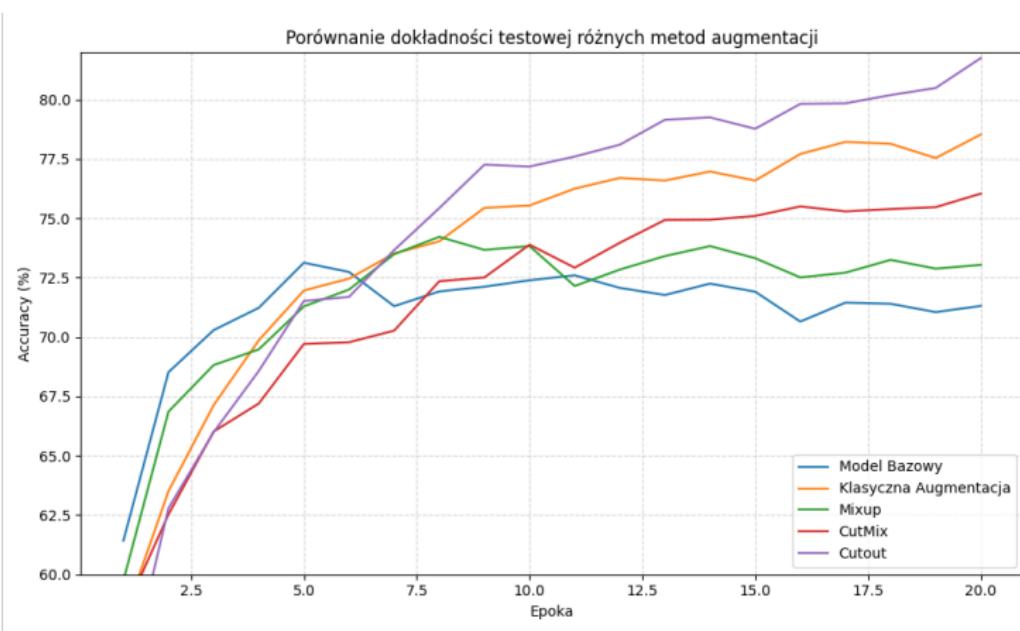
```
1 lam = np.random.beta(alpha, alpha)
2 rand_index =
3     torch.randperm(inputs.size(0)).to(device)
4 target_a, target_b = targets,
5     targets[rand_index]
6 bbx1, bby1, bbx2, bby2 =
7     rand_bbox(inputs.size(), lam)
8 inputs[:, :, bbx1:bbx2, bby1:bby2] =
9     inputs[rand_index, :, bbx1:bbx2,
10         bby1:bby2]
11 lam = 1 - ((bbx2 - bbx1)*(bby2 - bby1) / (H *
12             W))
13 loss = lam * criterion(output, target_a) + (1 -
14             lam) * criterion(output, target_b)
```

Ustawienia treningu

Wszystkie modele trenowane na zbiorze **CIFAR-10** z wykorzystaniem tej samej architektury sieci konwolucyjnej opisanej wcześniej. Parametry treningu były stałe dla każdego eksperimentu:

- **Liczba epok:** 20
- **Batch size:** 128
- **Optymalizator:** Adam
- **Współczynnik uczenia:** 0.001
- **Funkcja straty:** CrossEntropyLoss

Porównanie testowej dokładności dla różnych technik augmentacji



Rysunek: Porównanie test accuracy dla wszystkich metod

Porównanie wyników modeli

Tabela: Porównanie skuteczności modeli

Model	Test Accuracy [%]	Test Loss	Epoka
Model bazowy	73.13	0.8162	5
Klasyczna augmentacja	78.53	0.6355	20
Mixup	74.22	0.7805	8
CutMix	76.04	0.7488	20
Cutout	81.75	0.5456	20

Wnioski

- Najlepsze wyniki daje metoda *Cutout*.
- Modele, w których została zastosowana augmentacja są bardziej odpowne na przeuczenie.
- Model bazowy okazuje się najgorszy przy wyższej liczbie epok.

Bibliografia

-  WolframAlphaV1.0, *Easy Way to Improve Image Classifier Performance: Preprocessing Techniques*, Medium, 15 lipca 2020. <https://medium.com/@wolframalphav1.0/easy-way-to-improve-image-classifier-performance-p>
-  Politechnika Warszawska, *Analiza wydajności i optymalizacji systemów ML*, Repozytorium Politechniki Warszawskiej, 2019. <https://repo.pw.edu.pl/info/article/WUTb3002690e731469db98e22d006f73e35/>