

---

# Metody wyjaśnialne w uczeniu głębokim

---

Kemal Erdem\*, Piotr Mazurek†, Piotr Rarus‡

Politechnika Wrocławskiego

{183705, 237407, 192179}@student.pwr.edu.pl

## Abstract

Wyjaśnialna sztuczna inteligencja (XAI, ang. *Explainable artificial intelligence*) oznacza zestaw metod które pozwalają na zrozumienie przyczyny podjęcia danej decyzji przez teoretycznie niezrozumiały model uczenia maszynowego. W tym artykule przedstawiamy istniejące metody, pokazujemy dostępne narzędzia pozwalające wyjaśniać własne modele oraz prezentujemy wyzwania i problemy przed którymi stoją obecnie istniejące rozwiązania.

## Spis treści

<b>1 Wprowadzenie</b>	<b>3</b>
1.1 Regulacje . . . . .	3
1.2 Modele interpretowalne i nieinterpretowalne . . . . .	3
<b>2 Podejścia do XAI</b>	<b>3</b>
2.1 Surrogate Based . . . . .	3
2.1.1 Global Surrogate . . . . .	3
2.1.2 Local Surrogate . . . . .	5
2.2 Attribution Based . . . . .	7
2.2.1 Integrated Gradients . . . . .	7
2.2.2 IG przykłady . . . . .	8
2.3 Contrastive Explanation . . . . .	12
2.4 Counterfactual Explanation . . . . .	13
2.5 Example similarity . . . . .	14
<b>3 Narzędzia</b>	<b>15</b>
3.1 Captum . . . . .	15
3.1.1 Dostępne metody . . . . .	15
3.1.2 Wykorzystanie . . . . .	15
3.1.3 Captum Insights . . . . .	16

\*1. Wprowadzenie, 2. Podejścia do XAI

†4. Wyzwania i problemy z metodami XAI

‡3. Narzędzia

3.2	tf-explain . . . . .	17
3.2.1	Dostępne metody[26] . . . . .	17
3.2.2	Wykorzystanie . . . . .	17
3.3	Lime . . . . .	18
3.3.1	Wykorzystanie . . . . .	18
3.4	SHAP . . . . .	19
3.4.1	DeepShap - Wykorzystanie . . . . .	19
3.4.2	Gradient Explainer - Wykorzystanie . . . . .	20
<b>4</b>	<b>Wyzwania i problemy z metodami XAI</b>	<b>22</b>
4.1	Ewaluacja wyjaśnień . . . . .	22
4.1.1	Ewaluacje oparte na ludzkiej percepceji przedstawionych wyjaśnień . . . . .	23
4.1.2	Ewaluacje oparte na danych liczbowych . . . . .	23
4.2	Wyjaśnianie błędnych predykcji . . . . .	25
4.3	Skomplikowana przestrzeń hiperparametrów . . . . .	26
<b>5</b>	<b>Podsumowanie</b>	<b>28</b>

# 1 Wprowadzenie

Explainable artificial intelligence (XAI) jest to dział sztucznej inteligencji zajmujący się wyjaśnialnością modeli tak aby były one zrozumiałe przez człowieka. XAI staje się coraz bardziej istotna w ostatnich latach, gdy przemysł i biznes przechodzi tranzycję z prostych modeli wyjaśnialnych na modele głębokie, które uzyskują lepsze rezultaty. Dlaczego XAI jest ważna?

- **systemy krytyczne** - musimy być pewni czy modele używane do obsługi urządzeń mogących wyrządzić szkodę, działają poprawnie
- **poprawa modeli** - dzięki wiedzy o sposobie działania modeli możemy poprawiać ich strukturę lub proces uczenia
- **nauka** - możemy uczyć się nowych rzeczy bazując na wyjaśnialności modeli które przewyższają ludzi swoimi rezultatami (np. AlphaGo i nowe sposoby na grę w GO [2])

## 1.1 Regulacje

Prócz powodów związanych bezpośrednio z sposobem działania modeli, w ostatnich latach powstały regulacje dotyczące prawa do wyjaśnialności systemów podejmujących decyzje. Najważniejszą dla UE jest GDPR[8] (w Polsce znane pod nazwą RODO). Jednym z najważniejszych artykułów w tym dokumencie jest Artykuł 22[3] w którym możemy znaleźć prawo do "*obtain an explanation of the decision reached*".

Kolejną regulacją jest pochodzący z USA Algorithmic Accountability Act[1]. W przeciwieństwie do GDPR nie jest on skierowany wyłącznie na wyjaśnialność ale także na zachowanie obiektywności algorytmu i brak dyskryminacji.

## 1.2 Modele interpretowalne i nieinterpretowalne

W uczeniu maszynowym możemy wydzielić grupę modeli które nazwiemy **Modele interpretowalne**. Do takiej grupy należą na przykład:

- **Linear regression**
- **Logistic regression**
- **Decision trees**

Z drugiej strony mamy tzw. "*modele głębokie*" które mimo wysokiej skuteczności nie są interpretowalne dla człowieka.

# 2 Podejścia do XAI

Istnieje wiele podziałów metod wyjaśnialnej sztucznej inteligencji a ponieważ nie ma odgórnie ustalonego poprawnego podziału, postanowiliśmy wybrać jeden z nich.

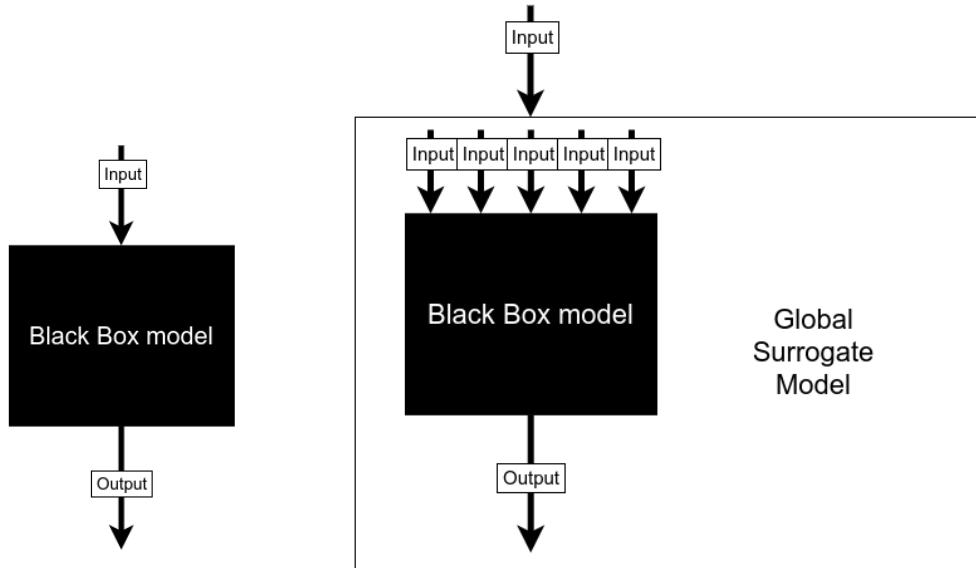
## 2.1 Surrogate Based

Podejście z wykorzystaniem modeli surogatów jest pierwszym sposobem jaki postanowiliśmy opisać. Idea podejścia jest prosta i polega na tworzeniu modeli interpretowalnych (np. podanych w sekcji 1.2) które mają naśladować modele nieinterpretowalne. Plusem takiego podejścia jest metoda niezależna od modelu który chcemy interpretować. Minusem natomiast jest spadek dokładności wyjściowego surogatu. Podejście z wykorzystaniem surogatów dzielimy na dwa sposoby:

- **Global Surrogate**
- **Local Surrogate**

### 2.1.1 Global Surrogate

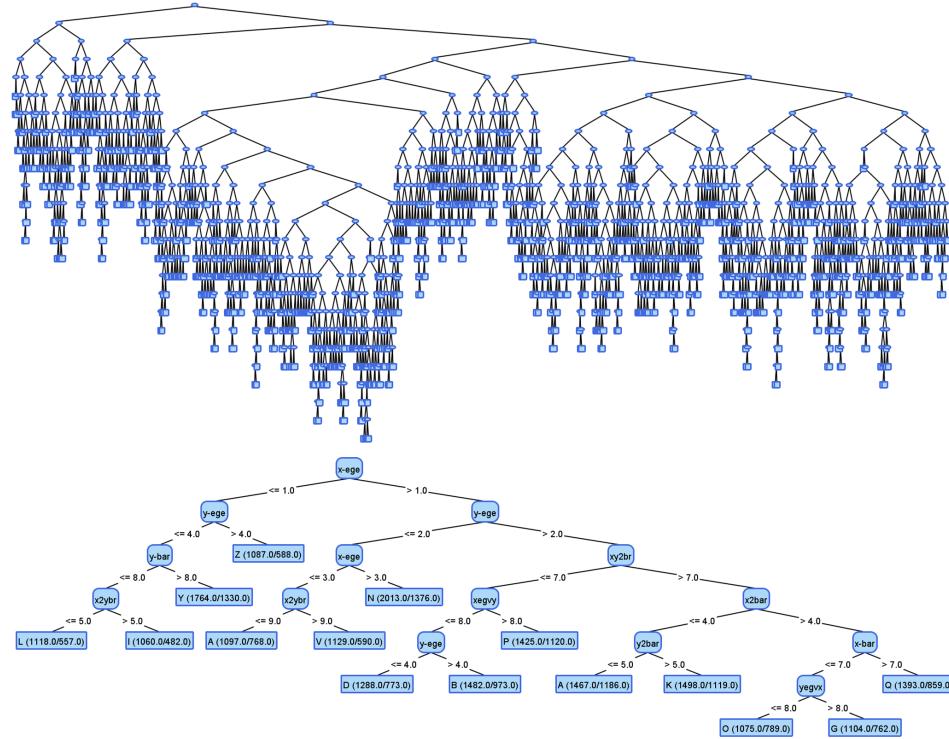
W przypadku globalnego surogatu naszym celem jest przybliżenie modelu głębokiego (Fig. 1) za pomocą jednego modelu interpretowalnego.



Rysunek 1: Black Box model

Rysunek 2: Global Surrogate training

Gdy uczymy taki model staramy się dokonywać perturbacji wejścia (Fig. 2) aby móc przybliżyć granicę decyzyjną modelu głębokiego. Oczywiście to rozwiązanie nie jest idealne ponieważ modele głębokie mogą posiadać bardzo skomplikowane granice i w ostateczności możemy otrzymywać bardzo złe surrogaty lub bardzo skomplikowane (patrz. Fig. 3).



Rysunek 3: Przykład drzewa decyzyjnego J48 i VTJ48 [20]

Z powodów takich ograniczeń obecnie popularne jest inne podejście z wykorzystaniem surogatów.

### 2.1.2 Local Surrogate

Local Surrogate (*Lokalne Surogaty*) jest podejściem starającym się naprawić ograniczenia poprzedniego poprzez stworzenie wielu lokalnych interpretowalnych modeli zamiast jednego dużego. Tak samo jak w poprzednim przypadku podejście jest niezależne od rodzaju modelu który staramy się interpretować oraz nie wymaga znajomości jego struktury. Aby stworzyć lokalnych surogatów potrzebujemy tylko i wyłącznie wejścia i wyjścia.

Najpopularniejszym przykładem tego podejścia jest metoda **LIME** (*Local Interpretable Model agnostic Explanations* [17]). LIME w przeciwieństwie do globalnych surogatów nie próbuje przybliżać całego modelu lecz znaleźć granicę dezycyjną dla jednego przypadku testowego.

$$\text{explanation}(x) = \arg \min_{g \in G} L(f, g, \pi_x) + \Omega(g) \quad (1)$$

Gdzie:

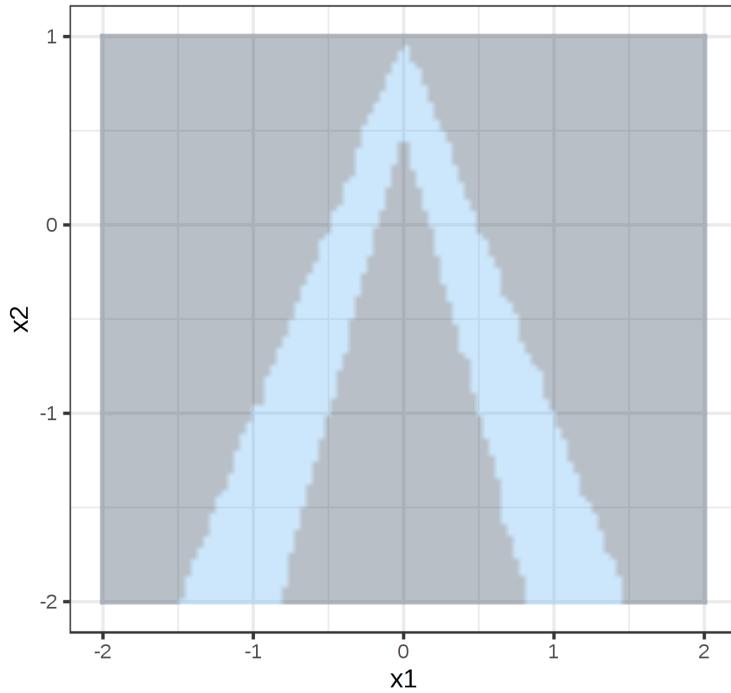
- **x** - przypadek testowy
- **g** - interpretowalny model
- **G** - rodzina wszystkich interpretowalnych modeli (np. wszystkie regresje liniowe)
- **L** - funkcja kosztu (np. MSE)
- **f** - oryginalny model
- $\pi_x$  - miara bliskości
- $\Omega(g)$  - miara złożoności modelu

Jeżeli chcielibyśmy przełożyć ten wzór na wyjaśnienie to możemy to wytlumaczyć w taki sposób: wyjaśnieniem (ang. *explanation*) dla danego przypadku testowego  $x$  jest model należący do rodziny modeli wyjaśnialnych  $G$  który minimalizuje funkcję kosztu, która mierzy jak bardzo oddalone są od siebie predykcje modelu  $g$  or oryginalnego modelu  $f$ . Jednocześnie złożoność modelu  $\Omega(g)$  powinna być na stosunkowo niskim poziomie. Miara bliskości  $\pi_x$  określa z jakiej odległości od przykładu  $x$  powinniśmy brać nasze perturbowane przykłady.

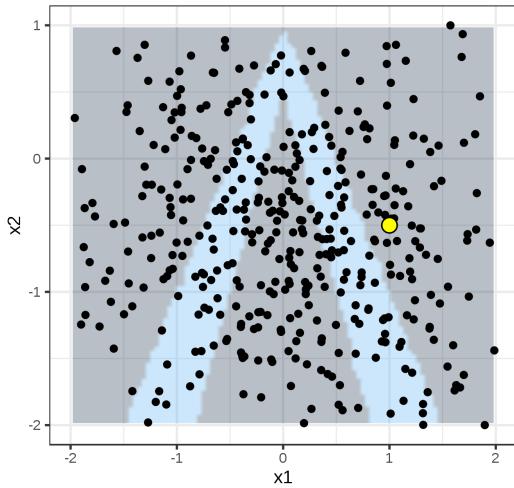
W praktyce  $\Omega(g)$  jest stała i z góry określona przed rozpoczęciem uczenia. Użytkownik określa maksymalną ilość cech którą liniowy model może użyć w celu przybliżenia.

Aby zobrazować działanie metody posłużymy się przykładem na danych tabelarycznych (LIME działa na różnych typach danych lecz dane tabelaryczne łatwo wizualizować i omawiać).

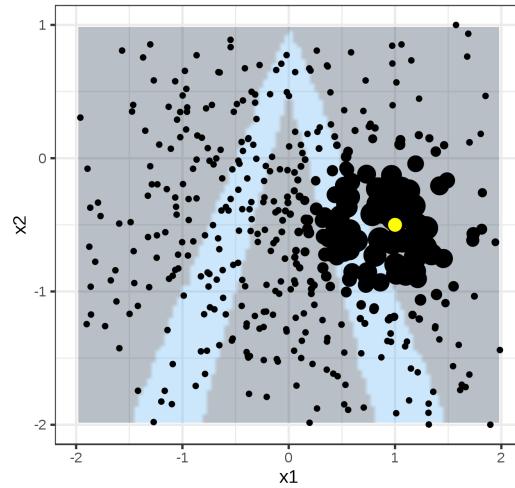
Na Rysunku 4 pokazana jest przestrzeń wygenerowana z wykorzystaniem dwóch cech **x1** i **x2**. W tej przestrzeni wydzielone są dwie klasy (*Niebieska* - 1, *Szara* - 0).



Rysunek 4: Rozpatrywana przestrzeń dwóch cech ( $x_1, x_2$ )<sup>1</sup>



Rysunek 5: Rozpatrywany przypadek i rozkład<sup>1</sup>

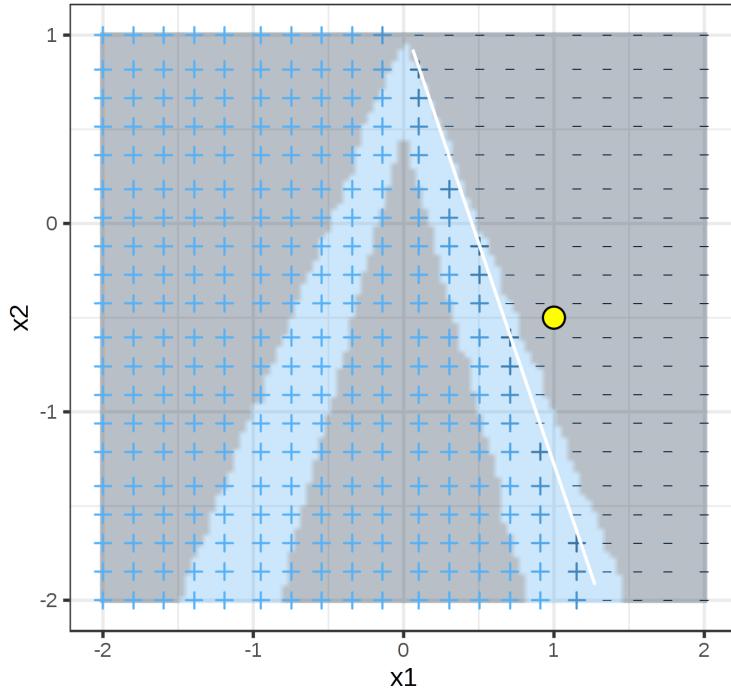


Rysunek 6: Waga punktów z uwzględnieniem miary bliskości<sup>1</sup>

Ponieważ LIME działa dla konkretnego przypadku testowego wybieramy jeden taki przypadek i nanosimy go na naszą przestrzeń (żółty punkt na Rys. 5). Następnie dobieramy punkty w tej przestrzeni z rozkładu normalnego (czarne punkty na Rys. 5). Wszystkie punkty klasyfikujemy z użyciem naszego oryginalnego modelu. Gdy punkty zostaną wybrane ustalamy ich wagę za pomocą miary bliskości zdefiniowanej wcześniej aby punkty bliżej naszego przykładu miały większy wpływ na model (Rys. 6).

---

<sup>1</sup>Interpretable Machine Learning rozdz.5.7- Book by Christoph Molnar



Rysunek 7: Liniowy model dla naszego przypadku testowego<sup>1</sup>

Na podstawie wagi oraz klasy przypisanej do punktów tworzymy model liniowy który pokazuje lokalną granicę decyzyjną (Rys. 7). Oczywiście trzeba tutaj zaznaczyć że ten przykład nie był trudnym przykładem i znaleziona granica ma sens. Można sobie wyobrazić jak wyglądałaby granica gdyby przypadek testowy znajdował się w punkcie  $\langle x_1=0, x_2=1 \rangle$ . Prawdopodobnie granica nie oddawałaby poprawnego zachowania oryginalnego modelu. Z tego też powodu metoda LIME doznała już wiele różnych modyfikacji które miały na celu zapobiec takim przypadkom.

## 2.2 Attribution Based

Metody atrybucyjne bazują na wyjaśnianiu predykcji modelu przy pomocy przypisywaniu jej do cech na wejściu w przypadku testowym. Wszystkie metody z tej grupy wymagają bezpośredniego dostępu do wnętrza modelu lecz tak samo jak w poprzednim przypadku mogą być stosowane do całej gamy różnych modeli. Ponieważ istnieje bardzo dużo metod z tej kategorii i wiele z nich jest popularna, postanowiliśmy się skupić i dokładniej opisać tylko jedną z nich zamiast opisywać większość bez wchodzenia w szczegóły.

### 2.2.1 Integrated Gradients

Metoda **IG** (*ang. Integrated Gradients* [22]) jest to metoda bazująca na liczeniu gradientów z interpolacją liniowej pomiędzy "baseline" (wartość bazowa np. w przypadku obrazów to może być jednolity czarny obraz) a wejściem docelowym (przykład testowy).

$$\text{IntegratedGrads}_i(x) := (x_i - x'_i) \times \int_{\alpha=0}^1 \frac{\partial F(x' + \alpha \times (x - x'))}{\partial x_i} d\alpha \quad (2)$$

Gdzie:

- **i** - wymiar z wektora wejściowego cech
- **x** - przypadek testowy
- **x'** - baseline
- $\alpha$  - wartość interpolacji

- $\frac{\partial F(x)}{\partial x_i}$  - gradient naszego modelu po danym wymiarze cech (" $i$ " wyżej)

Oczywiście obliczenie tej całki jest mało wykonalne więc stosowana jest aproksymacja wyglądająca następująco:

$$IntegratedGrads_i^{approx}(x) := (x_i - x'_i) \times \sum_{k=1}^m \frac{\partial F(x' + \frac{k}{m} \times (x - x'))}{\partial x_i} \times \frac{1}{m} \quad (3)$$

Gdzie:

- $\mathbf{i}$  - wymiar z wektora wejściowego cech
- $\mathbf{x}$  - przypadek testowy
- $\mathbf{x}'$  - baseline
- $\mathbf{k}$  - skalowana wartość interpolacji
- $\mathbf{m}$  - liczba kroków
- $\frac{\partial F(x)}{\partial x_i}$  - gradient naszego modelu po danym wymiarze cech (" $i$ " wyżej)

Pierwszą rzeczą jaką należy wyjaśnić jest "*baseline*". Baseline jest to reprezentacja wejścia która nie zawiera informacji użytecznych dla badanego modelu. W przypadku zdjęć można to wizualizować jako cały czarny obraz (macierz wypełniona zerami). Wybranie odpowiednich wartości cech dla baseline nie jest łatwe i może się różnić pomiędzy różnymi modelami a nawet poszczególnymi przypadkami testowymi. Więcej informacji można uzyskać sprawdzając pracę naukową na temat wizualizacji wpływu baseline na atrybucję cech [21]. Jednakże jeżeli chcemy znaleźć jeden dobry przykład na niepoprawność wypełniania wektora cech zerami warto rozważyć przykład klasyfikacji czy zdjęcie zostało zrobione w dzień lub w nocy. W takim przypadku klasyfikacji binarnej zdjęcia nocne (z reguły niższe wartości RGB) będą zbyt podobne do baseline i model może klasyfikować baseline (z dużą pewnością) jako zdjęcia nocne. W takim przypadku losowe wartości cech będą lepszym rozwiązaniem.

### 2.2.2 IG przykłady

Sposób działania metody można wyjaśnić na przykładzie. W naszym przypadku użyjemy sieci *Inception v1*[23] która posłuży do klasyfikacji obrazów. Jako wejście skorzystamy z dwóch obrazów reprezentujących kawie domowe, ubrane w różne stroje (Rys. 10).



guinea pig: 67.8% 339  
hamster: 10.7% 334  
wood rabbit: 0.9% 331

Rysunek 8: Kawia - pszczoła



guinea pig: 85.2% 339  
hare: 2.8% 332  
Angora: 1.8% 333

Rysunek 9: Kawia - królik

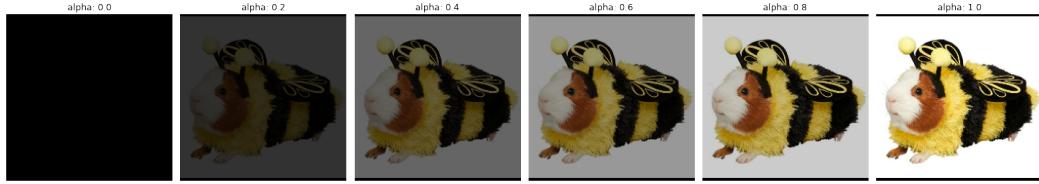
Rysunek 10: Rozpatrywane przypadki testowe (źródło: <https://www.petSMART.com/small-pet/toys-and-habitat-accessories/small-pet-costumes/>)

Przykłady testowe zostały sklasyfikowane poprawnie jako kawie domowe, mimo wielu cech odpowiadających innym zwierzętom. W drugim przypadku (Rys. 9). Z małą pewnością klasyfikator sądzi że jest to jednak królik i postaramy się odpowiedzieć na pytanie na co zwraca uwagę przy takiej klasyfikacji.

W naszym przypadku baseline będzie obraz wypełnionymi zerami (jednolity czarny) (Rys. 11) i będziemy dokonywać interpolacji pomiędzy nim a przykładem testowym dla zadanej liczby kroków (50).



Rysunek 11: Baseline i przykład testowy

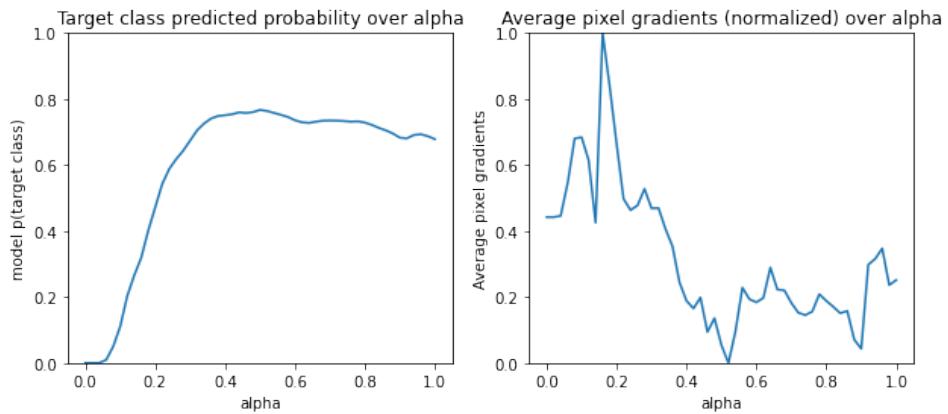


Rysunek 12: Przykład interpolacji dla 5 krokowego procesu

Każda z interpolacji pokazanych na Rys. 12 jest wykorzystywana w celu policzenia gradientu po każdym wymiarze wejścia (pixele) modelu (4).

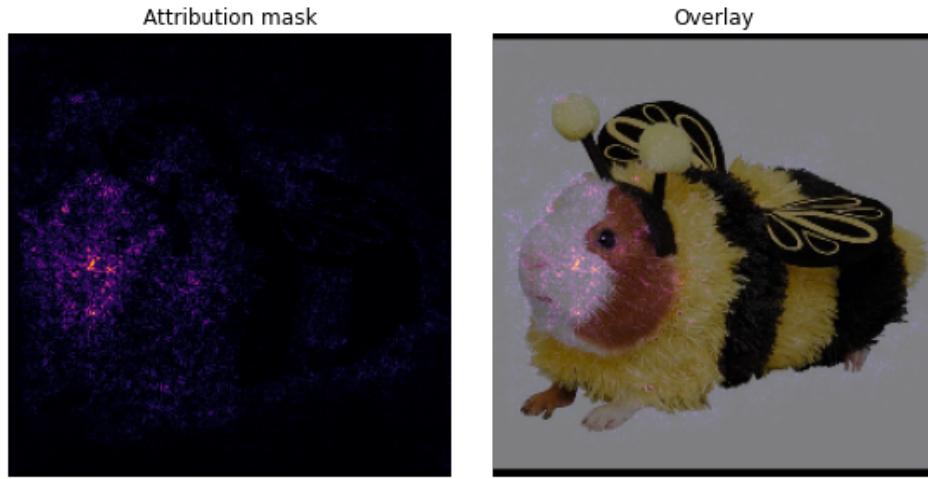
$$IntegratedGrads_i^{approx}(x) := (x_i - x'_i) \times \sum_{k=1}^m \frac{\partial F(\overbrace{x' + \frac{k}{m} \times (x - x')}^{\text{interpolate } m \text{ images at } k \text{ intervals}})}{\partial x_i} \times \frac{1}{m} \quad (4)$$

Mając interpolowane wejścia możemy policzyć wartości gradientów a następnie zobrazować jak zmienia się pewność modelu oraz średnie znormalizowane wartości gradientów dla naszych wymiarów (pixeli) (Rys. 13).



Rysunek 13: Wyniki dla Rys. 8

Sumując nasze gradienty i mnożąc je przez odpowiedni współczynnik (ang. *scale factor*  $(x_i - x'_i)$ ), otrzymujemy maskę którą możemy nazwać maską atrybucji (ang. *attribution mask*).



Rysunek 14: Klasa kawia domowa

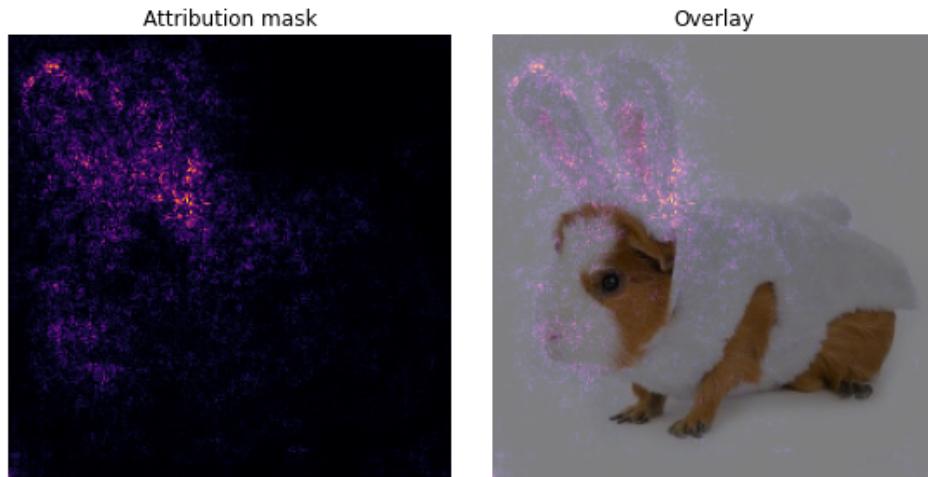


Rysunek 15: Klasa pszczoła

Gdy wygenerujemy sobie maski atrybucji dla pierwszego przykładu testowego i dwóch klas które nas interesują (*kawia domowa* i *pszczoła*) możemy zauważać interesujące rzeczy. W przypadku klasyfikacji jako kawia domowa, nasz model przywiązuje większą uwagę do cech wejścia zlokalizowanych bliżej pyszczka kawii (Rys. 14). Jednak w przypadku klasyfikacji jako pszczoła zwraca znaczenia nabierają elementy kostiumu (Rys. 15). Mimo że pewność klasyfikacji jako pszczoła jest zdecydowanie niższa i wynosi około 0.2% nadal możemy sprawdzić które cechy są istotne dla modelu.



Rysunek 16: Klasa kawia domowa

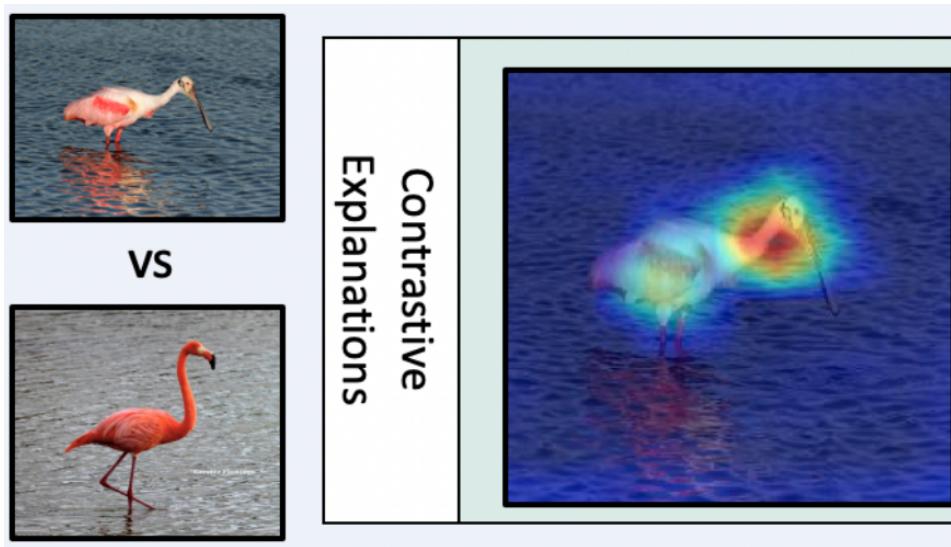


Rysunek 17: Klasa królik

Gdy spojrzymy na nasz drugi przypadek testowy (Rys. 9) tutaj kostium królika ma już zdecydowanie większą pewność (chocż nadal małą w porównaniu do kawii domowej). Maska dla kawii wygląda bardzo podobnie do poprzedniego przypadku i koncentruje się na okolicach pyszczka tego osobnika (Rys. 16) natomiast druga maska dla klasy "królik" bierze pod uwagę sztuczne uszy (Rys. 17).

### 2.3 Contrastive Explanation

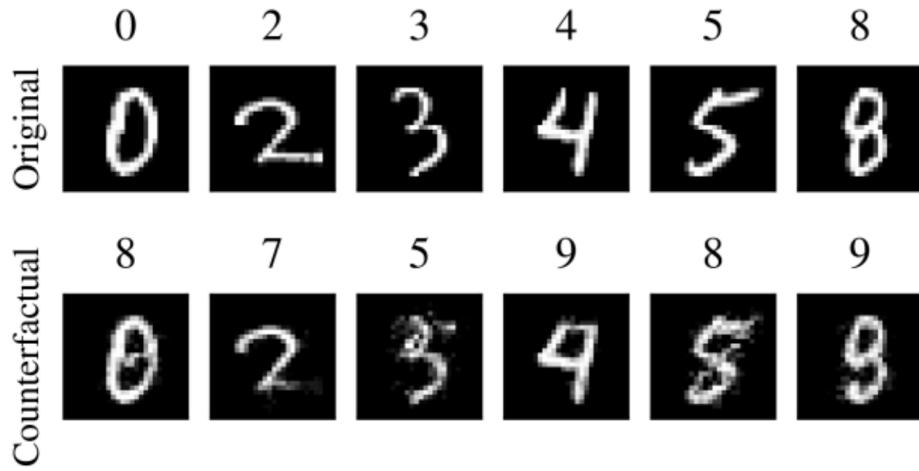
Metody bazujące na kontrastach polegają na znajdowaniu cech które powinny się znaleźć na wejściu, aby wyjście miało określony kształt lub cech które powinny zostać usunięte z wejścia aby wyjście miało określony kształt. Dobrym przykładem na zobrazowanie tego podejścia jest przykład z pracy *Contrastive Explanations in Neural Networks* [16] który wykorzystuje metodę Grad-CAM[18] aby wizualizować jakie cechy ma *Warzęcha zwyczajna* które odróżniają ją od *Flaminga* (Rys. 18).



Rysunek 18: Dlaczego Warzęcha a nie Flaming? [16]

#### 2.4 Counterfactual Explanation

Kontrfaktyczne (ang. Counterfactual) podejście jest bardzo podobne do podejścia bazującego na kontrastach. Główną różnicą jest to że zamiast podawać, które cechy odróżniają dany przykład od innego przykładu, to podejście stara się odpowiedzieć na pytanie "które wartości i jak bardzo należałoby zmienić aby z odpowiedzi  $X$  otrzymać  $Y$ ". Innymi słowy, szukamy minimalnego zestawu cech które z klasy  $X$  zrobią nam  $Y$  przy danym wejściu do modelu. W pracy *Counterfactual Explanations & Adversarial Examples – Common Grounds, Essential Differences, and Potential Transfers* [7], znajduje się przykład tego podejścia (Rys. 19) na zbiorze MNIST i wygląda on następująco:



Rysunek 19: Przykład metody "Counterfactual Explanation" [7]

Jak widzimy, w wektorach wejściowych zostały wprowadzone zmiany które przyczyniły się do zmiany klasyfikacji tych wektorów. Bardzo często ta metoda jest mylona z Adversarial Examples i podana wyżej praca dokonuje rozróżnienia pomiędzy nimi dwoma.

## 2.5 Example similarity

Stosunkowo prostą metodą służącą do wyjaśniania modeli głębokich jest metoda bazująca na znajdowaniu przykładów podobnych. Metoda wymaga dostępu do implementacji modelu i polega na porównywaniu odległości pomiędzy reprezentacjami różnych przypadków testowych w przestrzeni n-tej warstwy sieci (np. przedostatnia warstwa sieci CNN). Odległość może być liczona na wiele sposobów, z czego najprostszym z nich jest prosta odległość euklidesowa pomiędzy dwoma wektormi. Bardzo często do tego typu metod wykorzystuje się algorytm T-SNE[14] aby nanieść wiele przykładów na przestrzeń 2D lub 3D. Dobrym przykładem takiego rozwiązania jest wizualizacja embeddingów 7 warstwy w pełni połączonej z sieci CNN (AlexNet [11]) (Rys. 20).



Rysunek 20: Wizualizacja przykładów w przestrzeni embeddingów AlexNet [24]

Możemy tutaj zauważyc, że obrazy znajdujące się bliżej siebie w przestrzeni 2D są ze sobą spokrewnione tematycznie a nie tylko kolorystycznie. Nie jest to idealny sposób reprezentacji czy zrozumienia sposobu działania sieci lecz bardziej sposób na określenie co sieć uważa za obiekty należące do tej samej klasy.

### 3 Narzędzia

#### 3.1 Captum

Captum[4] jest biblioteką przeznaczoną do interpretowania modeli sieci neuronowych zaimplementowanych przy wykorzystaniu biblioteki PyTorch. Captum dostarcza programistom szereg metod umożliwiających sprawdzenie, które cechy składają się na ostateczną decyzję modelu. Może być wykorzystywany do ulepszania oraz poprawiania modeli przez identyfikację cech, które wpływają na nieoczekiwane odpowiedzi modelu. Ułatwia również implementację własnych metod interpretujących zachowanie modeli zaimplementowanych w PyTorchu. Choć głównym odbiorcą biblioteki są programiści oraz naukowcy tworzący modele, Captum może być również wykorzystywany w środowisku produkcyjnym, aby pomóc użytkownikom zrozumieć decyzje modelu. Captum jest zaimplementowany w języku Python, a jego kolejne wydania są dostępne w galerii bibliotek pipi, co znacznie ułatwia instalację. Biblioteka jest rozwijana przez firmę Facebook w ramach Facebook Open Source. Jest popularna, bogata w metody oraz posiada należyté wsparcie.

##### 3.1.1 Dostępne metody

Metody interpretacji modelu[6], zaimplementowane w bibliotece Captum, dzielą się na trzy grupy: atrybucje podstawowe, atrybucje warstwy, atrybucje neuronu. Podejemy również przykłady dostępnych metod.

- Atrybucja podstawowa - pozwalają sprawdzić w jakim stopniu cechy wejściowe modelu wpływają na ostateczny wynik.
  - Integrated Gradients
  - Gradient SHAP
  - Saliency
  - Guided Backpropagation, Deconvolution
  - Guided GradCAM
  - Lime
- Atrybucja warstwy - pozwalają sprawdzić wpływ każdego neuronu w danej warstwie na wynik modelu.
  - Layer Conductance
  - Internal Influence
  - Layer Activation
  - GradCAM
- Atrybucja neuronu - pozwalają sprawdzić wpływ każdej z cech wejściowych na aktywację danego neuronu.
  - Neuron Conductance
  - Neuron Integrated Gradients
  - Neuron GradientSHAP

##### 3.1.2 Wykorzystanie

Przykład wykorzystania biblioteki do interpretacji predykcji modelu sieci konwolucyjnej na zbiorze *ImageNet*:

```
from captum.attr import GuidedGradCam
from torchvision import models

# Ładowanie pretrenowanego modelu AlexNet
alexnet = models.alexnet(pretrained=True)
alexnet.eval()

# Stworzenie obiektu interpretującego model
# Podczas tworzenia wskazujemy ostatnią warstwę konwolucyjną w sieci
```

```

guided_gc = GuidedGradCam(alexnet, alexnet.features[10])

# Przeprowadzamy predykcję sieci na danych
out = alexnet(batch)

# Wskazujemy klasę z najwyższym wynikiem
score, index = out.max(1)

# Obliczamy wpływ warstwy na predykcję otrzymanych klas
attributions = guided_gc.attribute(batch_t, index)

```

### 3.1.3 Captum Insights

Interpretacja decyzji modelu, bez właściwych wizualizacji, może być kłopotliwa. Tym, co wyróżnia bibliotekę na tle innych, jest dodatkowy moduł Captum Insights[5] pozwalający użytkownikowi w prosty sposób zwizualizować predykcje modeli. Posiada on również integrację z Jupyter Notebook.

Przykład wykorzystania Captum Insights:

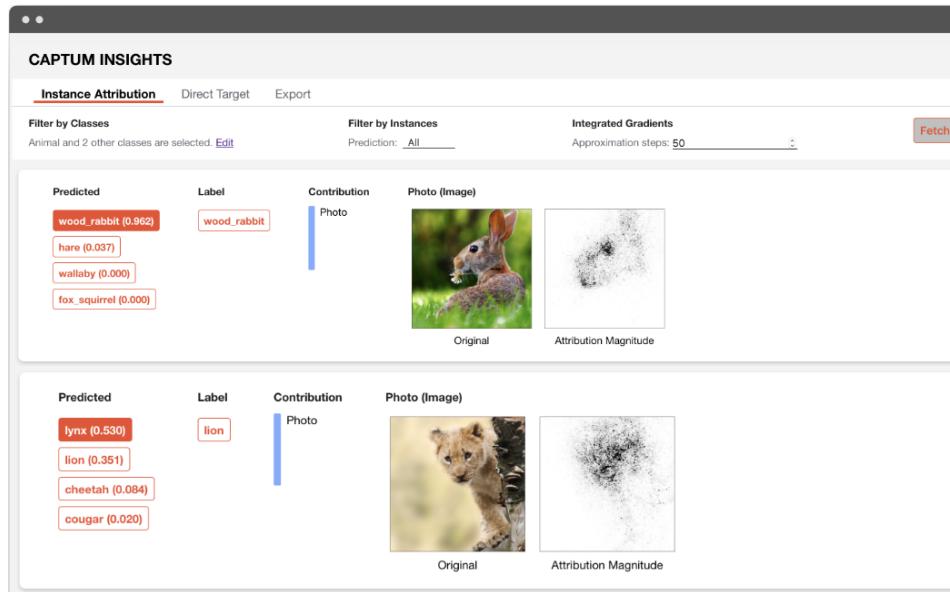
```

import torch
import torch.nn.functional as F
from captum.insights import AttributionVisualizer
from captum.insights.features import ImageFeature
from torchvision import models

# Ładowanie pretrenowanego modelu AlexNet
alexnet = models.alexnet(pretrained=True)
alexnet.eval()

# Uruchomienie wizualizacji we wątku notebook'a
visualizer = AttributionVisualizer(
    models=[alexnet],
    score_func=lambda o: F.softmax(o, 1),
    classes=[...], # nazwy klas
    features=[
        ImageFeature(
            "Photo",
            baseline_transforms=[...],
            input_transforms=[...],
        )
    ],
    dataset=...,
)
visualizer.render()

```



Rysunek 21: Przykład wykorzystania Captum Insights[5]

### 3.2 tf-explain

Kolejną z bibliotek przeznaczonych do interpretacji modeli sieci neuronowych jest tf-explain[25]. Narzędzie to jest dedykowane do interpretacji modeli zaimplementowanych w oparciu o bibliotekę TensorFlow. W porównaniu do Captum, biblioteka ta nie posiada oficjalnego wsparcia żadnej z dużych korporacji, jest mniej popularna oraz mniej bogata w metody interpretacji.

#### 3.2.1 Dostępne metody[26]

- Activations Visualization
- Vanilla Gradients
- Gradients\*Inputs
- Occlusion Sensitivity
- Grad CAM
- SmoothGrad
- Integrated Gradients

#### 3.2.2 Wykorzystanie

Z powyższych metod można korzystać zarówno podczas trenowania modelu wykorzystując wywołania zwrotne, jak również na wyuczonym modelu. Wszystkie metody zaimplementowane w bibliotece obsługują następujący interfejs:

- *explain* - metoda interpretująca model
- *save* - metoda zapisująca interpretację

#### Core API

Przykładowy kod interpretujący model:

```
# Importowanie klasy interpretującej model
from tf_explain.core.grad_cam import GradCAM

# Tworzenie instancji interpretera
explainer = GradCAM()
```

```

# Obliczanie interpretacji
output = explainer.explain(*explainer_args)

# Zapisanie wyniku
explainer.save(output, output_dir, output_name)

```

## Callbacks

Przykład zastosowania wywołań zwrotnych podczas uczenia modelu:

```

# Importowanie klasy interpretującej model
from tf_explain.callbacks.grad_cam import GradCAMCallback

# Dodanie interpretacji do wywołań zwrotnych podczas uczenia
callbacks = [
    GradCAMCallback(
        validation_data=(x_val, y_val),
        layer_name="activation_1",
        class_index=0,
        output_dir=output_dir,
    )
]

# Uruchomienie uczenia modelu wraz z interpretacją
model.fit(x_train, y_train, batch_size=2, epochs=2, callbacks=callbacks)

```

Rezultaty interpretacji są zapisywane w logach procesu uczenia i mogą być wyświetlane przy wykorzystaniu modułu Tensorboard.

## 3.3 Lime

Powyżej przedstawiliśmy narzędzia dedykowane dla sieci neuronowych. Istnieją również narzędzia przeznaczone do interpretacji szerszej klasy modeli uczenia maszynowego. Jedną z takich metod jest Lime[12]. Biblioteka jest zaimplementowana w języku Python i dostępna w galerii paczek pip. Biblioteka pozwala interpretować pojedyncze predykcje modeli uczonych na danych tekstowych, tabelarycznych lub na zdjęciach. Lime jest w stanie interpretować każdy model klasyfikatora typu black-box.

### 3.3.1 Wykorzystanie

Przykład wykorzystania Lime do interpretacji klasyfikacji danych obrazowych i wyznaczenia wpływu poszczególnych pikseli na ostateczną predykcję.

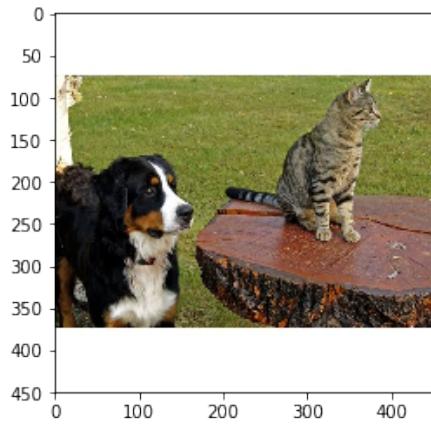
```

from lime import lime_image

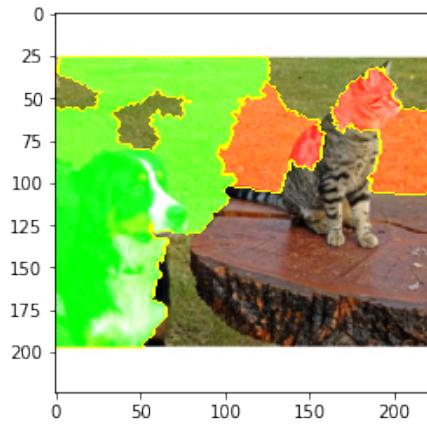
# Tworzenie instancji obiektu interpretującego model
explainer = lime_image.LimeImageExplainer()

explanation = explainer.explain_instance(
    image, # obraz wejściowy
    predict, # funkcja predykcyjna interpretowanego klasyfikatora
)

```



(a) Obraz wejściowy



(b) Interpretacja predykcji klasy pies

Rysunek 22: Zielony - obszar wpływający dodatnio na predykcję, Czerwony - obszar wpływający negatywnie na predykcję[13]

### 3.4 SHAP

Kolejną z metod, które również można wykorzystać do interpretacji sieci neuronowych jest SHAP[19]. Biblioteka również jest zaimplementowana w języku Python i dostępna w galerii paczek pipi. Biblioteka dostarcza dedykowaną metodę *Deep Shap*, która jest zoptymalizowana do analizy modeli sieci neuronowych i pozwala szybko obliczyć przybliżone wartości SHAP.

#### 3.4.1 DeepShap - Wykorzystanie

```

import numpy as np
import shap

model = ... # model klasyfikatora
images = ... # zbiór danych obrazowych
background = images[:100]
test_images = images[100:103]

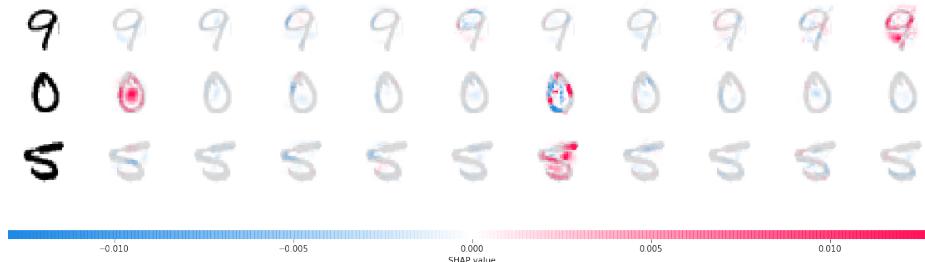
# Tworzenie instancji interpretera
explainer = shap.DeepExplainer(model, background)

# Obliczanie wartości shap
shap_values = e.shap_values(test_images)

shap_numpy = [np.swapaxes(np.swapaxes(s, 1, -1), 1, 2) for s in shap_values]
test_numpy = np.swapaxes(np.swapaxes(test_images.numpy(), 1, -1), 1, 2)

# Wyświetlenie wpływu cech na predykcję
shap.image_plot(shap_numpy, -test_numpy)

```



Rysunek 23: Wykres prezentuje wpływ cech na predykcję każdej z klas uszeregowanych do 0 do 9 w kolejnych kolumnach.[19]

### 3.4.2 Gradient Explainer - Wykorzystanie

Biblioteka posiada również zaimplementowaną metodę Expected Gradients łączącą cechy Integrated Gradients, Shap oraz SmoothGrad w jedną metodę interpretacji modeli. Pozwala to na użycie całego zbioru danych jako rozkładu tła i lokalne wygładzanie. Metoda polega na aproksymacji modelu funkcją liniową pomiędzy każdą próbką tła a próbką, którą chcemy wyjaśnić, przy założeniu, że cechy wejściowe są niezależne, wtedy Expected Gradients będą aproksymacją wartości SHAP.

```

from keras.applications.vgg16 import VGG16
from keras.applications.vgg16 import preprocess_input
import keras.backend as K
import numpy as np
import json
import shap

# ładowanie pretrenowanego modelu klasyfikatora
model = VGG16(weights='imagenet', include_top=True)
X, y = shap.datasets.imagenet50()

# wybór dwóch obrazów ze zbioru do interpretacji
to_explain = X[[39, 41]]

# ładowanie nazw etykiet zbioru ImageNet
url = "https://s3.amazonaws.com/deep-learning-models/image-models/imagenet_class_index.json"
fname = shap.datasets.cache(url)
with open(fname) as f:
    class_names = json.load(f)

# funkcja mapująca obraz wejściowy na podaną warstwę
def map2layer(x, layer):
    feed_dict = dict(zip([model.layers[0].input], [preprocess_input(x.copy())]))
    return K.get_session().run(model.layers[layer].input, feed_dict)

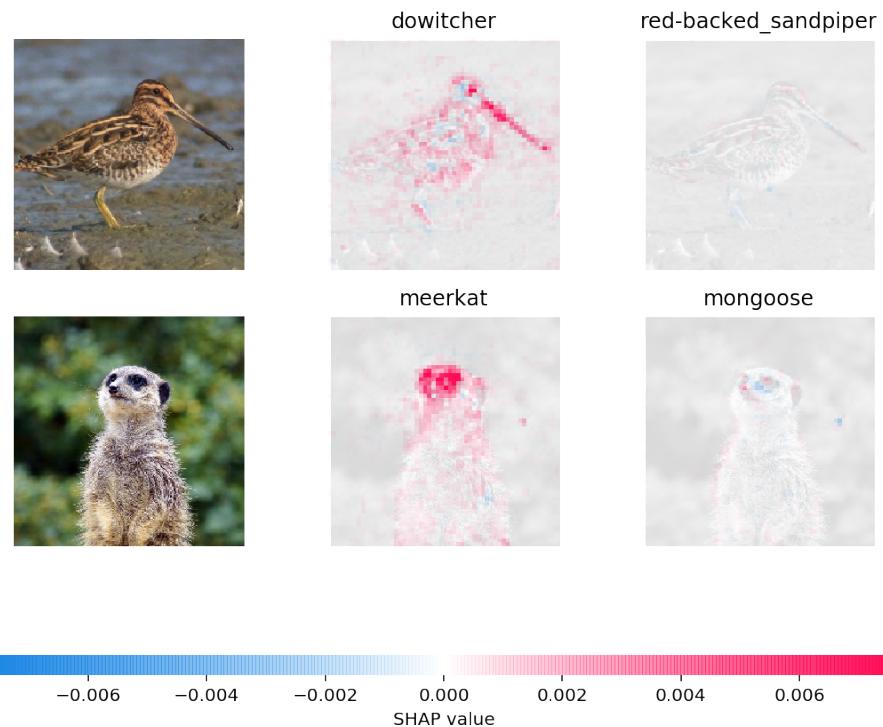
# interpretacja wpływu wejścia do siódmej warstwy modelu na klasyfikację
e = shap.GradientExplainer(
    (model.layers[7].input, model.layers[-1].output),
    map2layer(X, 7),
    local_smoothing=0
)

shap_values, indexes = e.shap_values(map2layer(to_explain, 7), ranked_outputs=2)

# określenie nazw wypredykowanych klas
index_names = np.vectorize(lambda x: class_names[str(x)][1])(indexes)

```

```
# uruchomienie wizualizacji  
shap.image_plot(shap_values, to_explain, index_names)
```



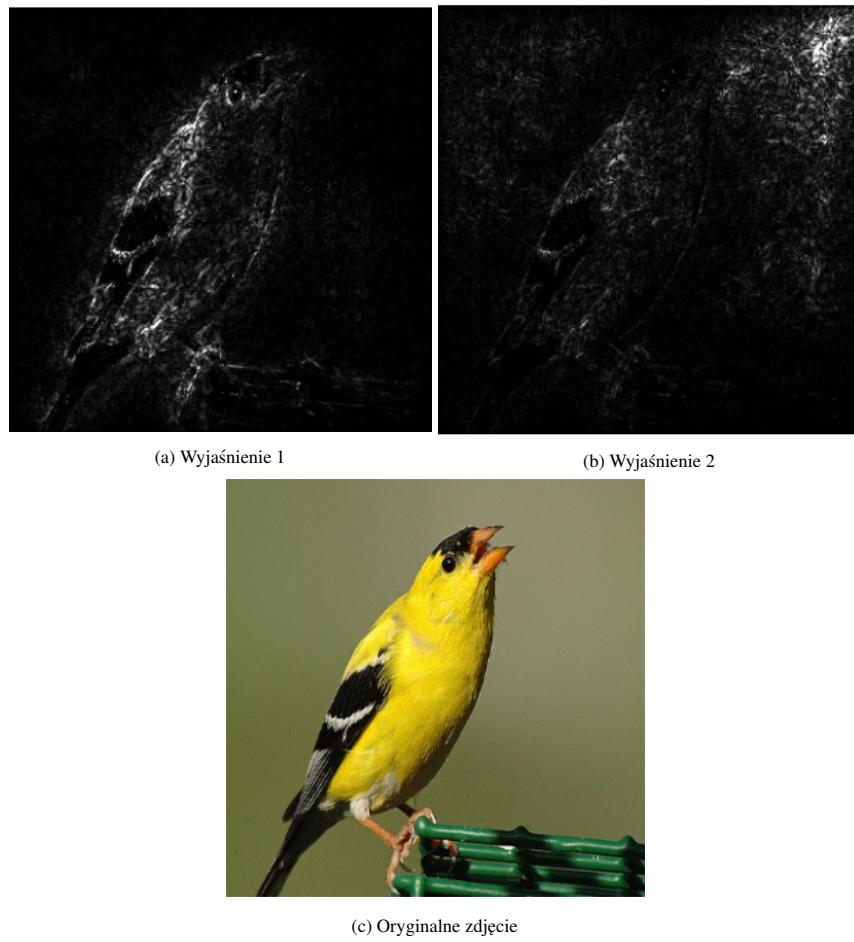
Rysunek 24: Wykres prezentuje wpływ cech na predykcję dwóch najbardziej prawdopodobnych klas.[19]

## 4 Wyzwania i problemy z metodami XAI

Pomimo dynamicznego rozwoju w ostatnich latach metody wyjaśnialne są wciąż dalekie bycia idealnymi. Pomimo silnych podstaw teoretycznych prezentowane wyjaśnienia są często nie lepsze od losowych atrybucji[10]. Większość metod skupia się jedynie na pokazaniu prostych korelacji pomiędzy zmiennymi zamiast stosować podejście przyczynowe, a przestrzeń hiperparametrów związana z niektórymi metodami może w praktyce uniemożliwić znalezienie uniwersalnej metody do wyjaśniania modeli.

### 4.1 Ewaluacja wyjaśnień

Dlaczego jedno wyjaśnienie jest lepsze od drugiego? Wbrew pozorom jest to bardzo skomplikowane pytanie. Jako ludzie intuicyjnie czujemy że Wyjaśnienie 1 (Rys. 25c) jest lepsze od Wyjaśnienia 2 (Rys. 25b), aczkolwiek wykazanie tego w sposób formalny jest wciąż otwartym nieroziązonym problemem.



Rysunek 25: Wizualizacja wyjaśnień dla różnych wartości hiperparametru  $\alpha$  w Integrated Gradients<sup>4</sup>

Do podejścia ewaluacji autorzy zasadniczo podchodzą na dwa sposoby:

- Ewaluacje oparte na ludzkiej percepcji przedstawionych wyjaśnień
- Ewaluacje oparte na danych liczbowych

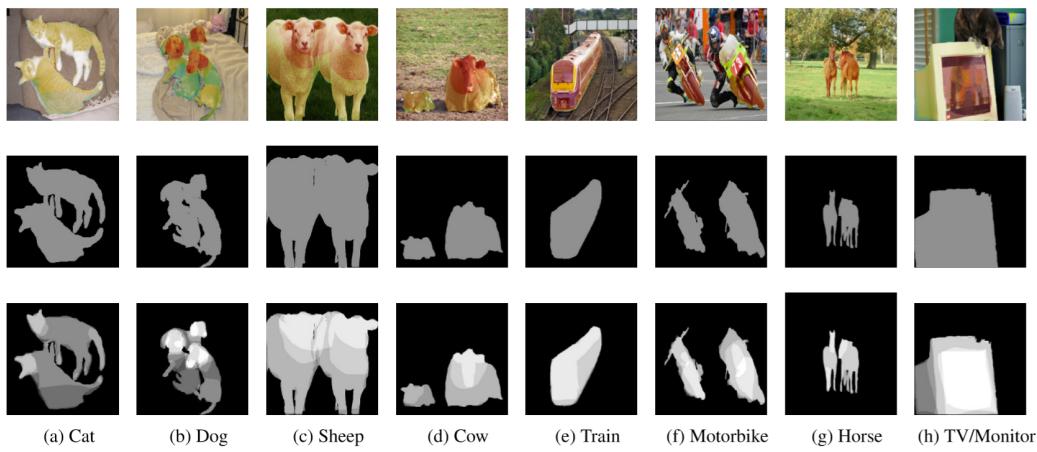
<sup>4</sup><https://distill.pub/2020/attribution-baselines/>

#### 4.1.1 Ewaluacje oparte na ludzkiej percepcji przedstawionych wyjaśnień

Tego typu ewaluacje to zwykle proste porównanie tego co wg. ludzi jest cechą wpływającą na predykcje z tymi cechami które pokazuje model.

Ciekawym przykładem jest tutaj praca **A Human-Grounded Evaluation Benchmark for Local Explanations of Machine Learning**[15]. Autorzy zaproponowali wykorzystanie rankingu tworzonego przez 10 losowo wybranych użytkowników do ewaluacji wyjaśnień dawanych przez metody XAI. Badania przeprowadzane były na obrazach oraz tekstach (z powodu ludzkich ograniczeń nie da się w efektywny sposób przeprowadzić takich badań dla danych tabelarycznych).

Dla zwróconych wyjaśnień użytkownicy przypisywali wynik punktowy (z zakresu 0 do 1) tego na ile wg. nich dana część danych kontrybuje do finalowej predykcji (uproszczony przykład dla segmentacji na Rys. 26). Następnie obliczano korelację pomiędzy wynikami zwracanymi przez metody XAI (badano LIME[17] i GradCAM[18]) a ewaluacją ludzi.



Rysunek 26: Przykładowe wyjaśnienie błędnej predykcji[15]

Autorzy pokazali że istnieje pewna korelacja pomiędzy tymi wynikami, aczkolwiek sami przyznają że błędy poznawcze annotatorów (ang *bias*) mają znaczący wpływ na rezultaty eksperymentów co sprawia że zaproponowany przez nich benchmark jest daleki od obiektywnego.

Temu problemowi popróbowią zaradzić metody oparte o obiektywne dane matematyczne.

#### 4.1.2 Ewaluacje oparte na danych liczbowych

W metodach opartych na danych liczbowych liczni autorzy starali się zaproponować obiektywne metody mierzenia jakości wyjaśnień. Dwa ciekawsze naszym zdaniem podejścia to praca *A Benchmark for Interpretability Methods in Deep Neural Networks*[10] oraz *On the (In)fidelity and Sensitivity for Explanations*[27].

Sara Hooker i inni autorzy w 2018 roku wykazali że duża część wyjaśnialnych metod atrybucyjnych nie różni się za bardzo od losowego przypisania wartości istotności do poszczególnych części danych wejściowych.

Aby to wykazać autorzy zaproponowali podejście które sami nazwali **ROAR (RemOve And Retrain)**

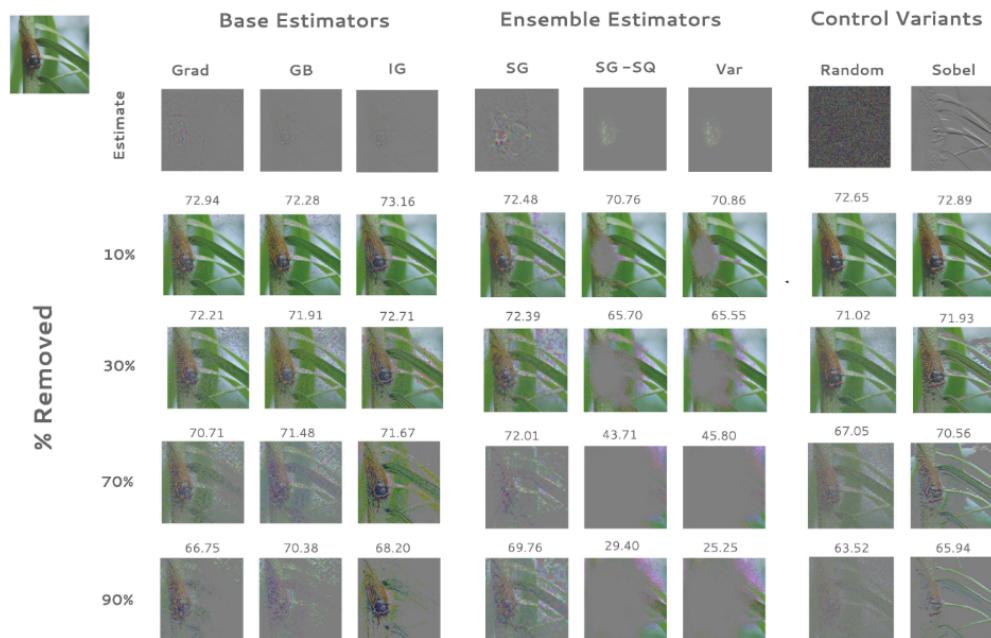
Zgodnie z nazwą podejście sprowadza się do dwóch kroków usunięcia części danych i ponownego przetrenowania modeli. Metoda w teorii powinna zadziałać dla dowolnego zestawu danych, ale autorzy prezentują jej działanie na obrazach.

Autorzy zakładają że poprawnie działająca metoda wyjaśnialna wskaże najważniejsze elementy danych wejściowych. Zgodnie z tym założeniem jeżeli z każdego obrazu usuniemy te najważniejsze dane to teoretycznie skuteczność klasyfikacji obiektów powinna znacząco spadać (w końcu usunięto tę część która była najistotniejsza dla poprawnej predykcji).

Dokładnie to zrobili autorzy. Dla każdego zdjęcia ze zbioru treningowego i testowego (ze zbioru ImageNet) wykorzystując 6 metod atrybucyjnych (Gradient heatmap, Integrated Gradients, Guided Backprop, SmoothGrad Integrated Gradients, SmoothGrad-Squared Integrated Gradients, VarGrad Integrated Gradients) określili najważniejsze piksele. Następnie przygotowano 4 nowe zbioru danych. W tych zbiorach usunięto odpowiednio 10, 30, 70 oraz 90 procent teoretycznie najważniejszych pikseli. Następnie dla każdego nowo utworzonego zbioru wytrenowano niezależnie od siebie 5 modeli Resnet50[9] i zbadano średnią skuteczność klasyfikacji obrazów.

Jako zbiór kontrolny utworzono atrybucje z losowo przypisany poziomami istotności. Teoretycznie wraz z usuwaniem coraz większej części istotnych (określanych przez metody XAI jako istotne) pikseli skuteczność powinna spadać dużo szybciej niż w losowym przypisaniu istotności do pikseli. Tak się jednak nie dzieje (Rys. 27). Dla większości metod XAI testowanych przez autorów spadek skuteczności klasyfikacji jest taki sam jak dla losowego przypisania pikseli.

Można stąd wyciągnąć wniosek że testowane przez autorów metody nie radzą sobie ze znajdowaniem istotnych pikseli lepiej niż losowa atrybucja. Jedynie metody SmoothGrad-Squared Integrated Gradients oraz VarGrad Integrated Gradients zachowały się tak jak powinny, tj. prawdopodobnie poprawnie przypisały istotność pikseli.



Rysunek 27: Zmiany skuteczności rozpoznawania poprawnej klasy wraz z usuwaniem pikseli zgodnie z metodą ROAR. Liczba nad obrazem to średnia skuteczność (ang. accuracy) na zbiorze testowym dla 5 niezależnie trenowanych modeli Resnet50. Modele były trenowane na zmodyfikowanych danych zgodnie z metodą ROAR.

Od lewej do prawej (gradient heatmap (GRAD), Integrated Gradients (IG), Guided Backprop (GB)), derivative approaches that ensemble a set of estimates (SmoothGrad Integrated Gradients (SG-SQ-IG), SmoothGrad-Squared Integrated Gradients (SG-SQ-IG), VarGrad Integrated Gradients (Var-IG))[10]

W pracy **On the (In)fidelity and Sensitivity for Explanations**[27] autorzy zaproponowali podejście alternatywne. Zakładają oni że dobrą metodą ewaluacji wyjaśnialności jest zbadanie stopnia w jakim metoda wyjaśnialna jest w stanie wykryć jak model  $f$  zmienia predykcje po dodaniu perturbacji  $I$  do oryginalnego wejścia  $x$ .

Aby obliczyć ten wpływ autorzy zaproponowali policzenie wartości oczekiwanej różnicę średnio-kwadratowej między wyjaśnieniem  $\Phi$  pomnożonym przez perturbację  $I$  a różnicą między predykcją

dla danych oryginalnych  $f(x)$  a predykcją dla danych zmienionych o perturbację  $f(x - I)$  (Równanie 5).

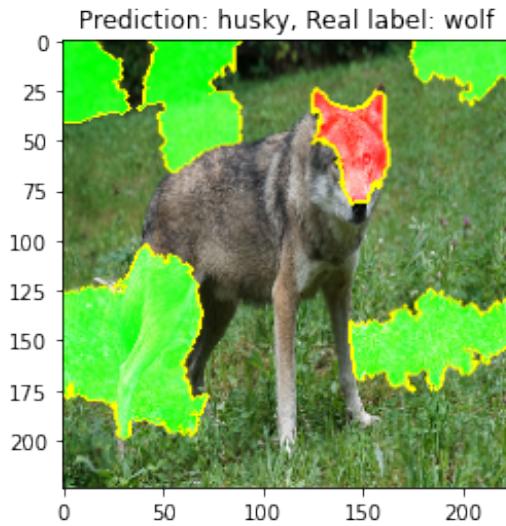
$$INFD(\Phi, f, x) = \mathbb{E} \left[ I^T \Phi(f, x) - (f(x) - f(x - I))^2 \right] \quad (5)$$

Wybór odpowiedniej perturbacji  $I$  jest hiperparametrem tej metody.

## 4.2 Wyjaśnianie błędnych predykcji

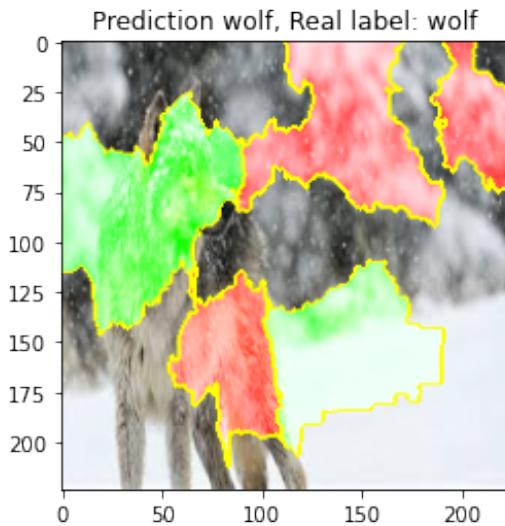
Jednym z częstych przykładów jakie pokazywane są na prezentacjach dotyczących LIME [17] jest wykorzystanie metod XAI to wykazania dlaczego model podjął błędą decyzję.

W przykładzie z LIME pokazane jest że model klasyfikujący psy na obrazach na wilki oraz husky tak naprawdę klasyfikuje to czy na zdjęciu widać znajduje się śnieg czy trawa (wilki w zbiorze danych były zawsze prezentowane w otoczeniu zimowym, a husky w trawie, Rys. 28).



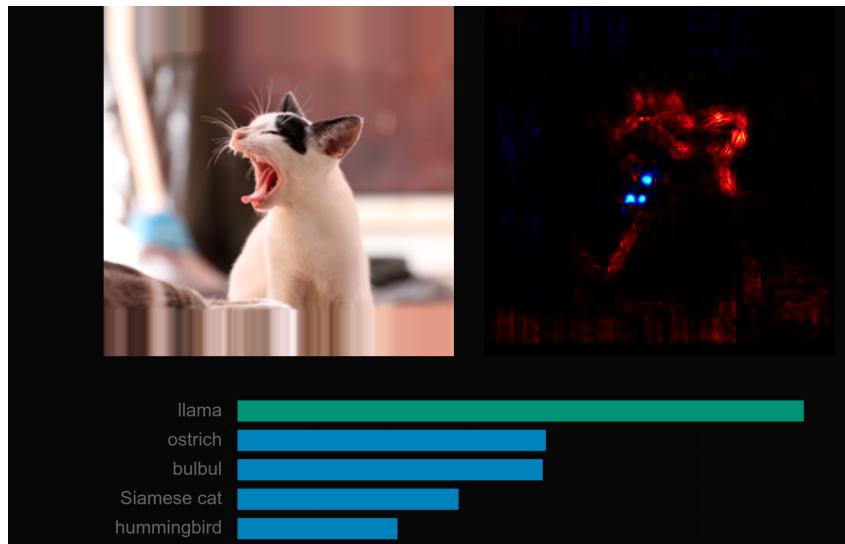
Rysunek 28: Przykładowe wyjaśnienie błędnej predykcji

Problemem jest to że w praktyce dawane wyjaśnienia często są sprzeczne ze sobą. Dla omawianego problemu np w teorii śnieżne tło powinno kontrybuować pozytywnie dla predykcji wilk, a kontrybuuje negatywnie (Rys. 29). Takie sprzeczne sygnały powodują że trudno zaufać takiej metodzie.



Rysunek 29: Wyjaśnienie predykcji "wilk"<sup>5</sup>

Wyjaśnianie błędnych predykcji staje się prawdziwym problemem kiedy wyjdziemy poza prosty przykład z wilkami i dwoma klasami do problemu typu ImageNet z 1000 klas. W takich przypadkach metody wyjaśnialne podają często nieintuicyjne wyjaśnienia które w wielu przypadkach będą bezużyteczne (Rys. 30.). Problem ten nie jest zbytnio opisany w literaturze, ale dużo osób natyka się na niego w praktyce korzystając z metod typu XAI.



Rysunek 30: Przykład wyjaśniania błędnej predykcji przez Integrated Gradients<sup>6</sup>

#### 4.3 Skomplikowana przestrzeń hiperparametrów

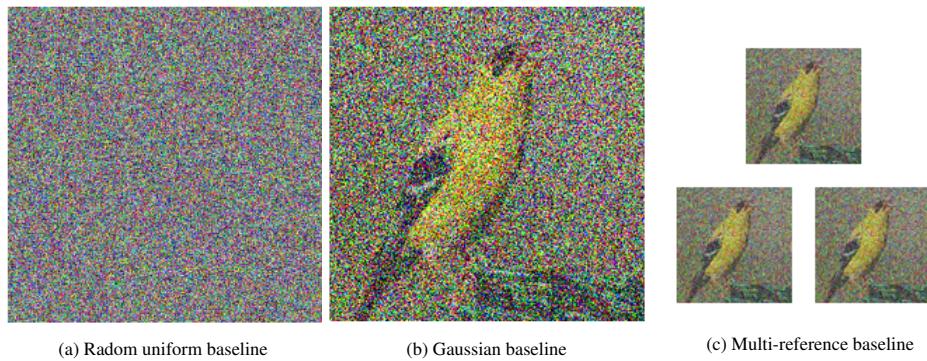
Problemem z wykorzystaniem metod XAI w praktyce jest konieczność określenia wielu hiperparametrów które mogą mieć znaczny wpływ na finałowe wyjaśnienie.

Dla omawianego wcześniej IG odpowiedni wybór baseline-u (Rys. 31) może mieć fundamentalne znaczenie dla zwracanego wyjaśnienia, podobnie jak wybór parametru  $\alpha$ . (Rys. 25) Ta liczba nieza-

<sup>5</sup><https://tugot17.github.io/data-science-blog/xai/lime/2020/09/09/LIME-biased-dataset.html>

<sup>6</sup><http://heatmapping.org/>

leżnych hiperparametrów jest dużą przeszkodą w wykorzystywaniu metod XAI w praktyce, bo nigdy nie wiadomo czy wybrano odpowiedni zestaw. Dodatkowo problemem jest fakt że nie ma żadnej gwarancji że zestaw hiperparametrów metody XAI poprawnie działający dla jednego obrazu będzie również dobrze działał dla innego obrazu.



Rysunek 31: Wizualizacja wyjaśnień dla różnych wartości hiperparametru  $\alpha$  w Integrated Gradients<sup>7</sup>

---

<sup>7</sup><https://distill.pub/2020/attribution-baselines/>

## **5 Podsumowanie**

Metody XAI to dynamicznie rozwijająca się w ostatnich latach gałąź sztucznej inteligencji. Spowodowane jest to chęcią stosowania systemów krytycznych opartych o skomplikowane modele a także nowo-powstającymi regulacjami. Podejść jest wiele, ale żadne nie jest uniwersalnie lepsze od innych. Intensywnie rozwijają się także są liczne otwarto-źródłowe implementacje tych algorytmów kompatybilne z popularnymi frameworkami do uczenia głębokiego.

Pomimo rosnącego zainteresowania tematyką XAI w ostatnich latach wciąż nie istnieją ogólnie przyjęte standardy ewaluacji relevantności zwracanych wyjaśnień, a same metody wciąż potrafią zwracać wyjaśnienia które są bezużyteczne dla ludzi.

XAI jako dziedzina prawdopodobnie w przyszłości będzie się coraz dynamicznej rozwijać i może być ciekawą ścieżką specjalizacji np. w ramach studiów doktoranckich.

## Literatura

- [1] *Algorithmic Accountability Act of 2019*. URL: <https://www.congress.gov/bill/116th-congress/house-bill/2231>.
- [2] *AlphaGo - DeepMind*. URL: <https://deepmind.com/research/case-studies/alphago-the-story-so-far>.
- [3] *Article 22 GDPR - Automated individual decision-making, including profiling*. URL: <https://www.privacy-regulation.eu/en/article-22-automated-individual-decision-making-including-profiling-GDPR.htm>.
- [4] *Captum*. URL: <https://captum.ai/>.
- [5] *Captum Insights*. URL: [https://captum.ai/docs/captum\\_insights](https://captum.ai/docs/captum_insights).
- [6] *Captum Methods*. URL: <https://captum.ai/docs/algorithms>.
- [7] Timo Freiesleben. *Counterfactual Explanations Adversarial Examples – Common Grounds, Essential Differences, and Potential Transfers*. 2020. arXiv: 2009.05487 [cs.AI].
- [8] *General Data Protection Regulation*. URL: <https://www.privacy-regulation.eu/en/index.htm>.
- [9] Kaiming He **and others**. *Deep Residual Learning for Image Recognition*. 2015. arXiv: arXiv: 1512.03385 [cs.CV].
- [10] Sara Hooker **and others**. *A Benchmark for Interpretability Methods in Deep Neural Networks*. 2018.
- [11] Alex Krizhevsky, Ilya Sutskever **and** Geoffrey Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. in: *Neural Information Processing Systems 25 (january 2012)*. DOI: 10.1145/3065386.
- [12] *Lime*. URL: <https://github.com/marcotcr/lime>.
- [13] *Lime - Torch example*. URL: <https://github.com/marcotcr/lime/blob/master/doc/notebooks/Tutorial%20-%20images%20-%20Pytorch.ipynb>.
- [14] Laurens van der Maaten **and** Geoffrey Hinton. “Visualizing data using t-SNE”. in: *Journal of Machine Learning Research 9 (november 2008), pages 2579–2605*.
- [15] Sina Mohseni, Jeremy E. Block **and** Eric D. Ragan. *A Human-Grounded Evaluation Benchmark for Local Explanations of Machine Learning*. 2018. arXiv: 1801.05075 [cs.CV].
- [16] Mohit Prabhushankar **and others**. *Contrastive Explanations in Neural Networks*. 2020. arXiv: 2008.00178 [cs.CV].
- [17] Marco Tulio Ribeiro, Sameer Singh **and** Carlos Guestrin. “*Why Should I Trust You?*”: Explaining the Predictions of Any Classifier. 2016. arXiv: 1602.04938 [cs.LG].
- [18] Ramprasaath R. Selvaraju **and others**. “Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization”. in: *International Journal of Computer Vision 128.2 (october 2019), pages 336–359*. ISSN: 1573-1405. DOI: 10.1007/s11263-019-01228-7. URL: <http://dx.doi.org/10.1007/s11263-019-01228-7>.
- [19] *SHAP*. URL: <https://github.com/slundberg/shap>.
- [20] Gregor Stiglic **and others**. “Comprehensive Decision Tree Models in Bioinformatics”. in: *PLOS ONE 7.3 (march 2012), pages 1–13*. DOI: 10.1371/journal.pone.0033812. URL: <https://doi.org/10.1371/journal.pone.0033812>.
- [21] Pascal Sturmfels, Scott Lundberg **and** Su-In Lee. “Visualizing the Impact of Feature Attribution Baselines”. in: *Distill 5 (january 2020)*. DOI: 10.23915/distill.00022.
- [22] Mukund Sundararajan, Ankur Taly **and** Qiqi Yan. *Axiomatic Attribution for Deep Networks*. 2017. arXiv: 1703.01365 [cs.LG].
- [23] Christian Szegedy **and others**. *Going Deeper with Convolutions*. 2014. arXiv: 1409.4842 [cs.CV].
- [24] *t-SNE visualization of CNN codes - Andrej Karpathy*. URL: <https://cs.stanford.edu/people/karpathy/cnnembed/>.
- [25] *tf-explain*. URL: <https://tf-explain.readthedocs.io/en/latest/>.
- [26] *tf-explain methods*. URL: <https://tf-explain.readthedocs.io/en/latest/methods.html>.
- [27] Chih-Kuan Yeh **and others**. *On the (In)fidelity and Sensitivity for Explanations*. 2019. arXiv: arXiv:1901.09392 [cs.CV].