



AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE
Wydział Elektrotechniki, Automatyki, Informatyki i Inżynierii Biomedycznej

Projekt dyplomowy

Analiza obszarów istotności w ocenie działania głębokich sieci neuronowych
Saliency maps in the analysis of deep learning methods

Autor: *Kacper Motyka*
Kierunek studiów: *Automatyka i Robotyka*
Opiekun pracy: *dr hab. inż. Wojciech Chmiel*

Kraków, 2023

Serdecznie dziękuję mojemu promotorowi dr hab. inż. Wojciechowi Chmielowi za poświęcony czas oraz merytoryczne wsparcie podczas pisania pracy.

Spis treści

1. Wstęp.....	7
1.1. Cel pracy	8
1.2. Zakres pracy.....	8
2. Sieci Neuronowe	9
2.1. Budowa i zasada działania perceptronu.....	9
2.1.1. Trenowanie perceptronu.....	10
2.2. Omówienie działania głębszych sieci neuronowych	11
2.2.1. Trenowanie głębszej sieci neuronowej.....	12
2.2.2. Funkcje aktywacji	14
2.2.3. Przetwarzanie zdjęć - wady modelu MLP	15
2.3. Konwolucyjne sieci neuronowe	15
2.4. Transfer Learning	19
2.5. Wybrane architektury sieci neuronowych.....	20
2.5.1. <i>VGG</i>	20
2.5.2. <i>ResNet</i>	22
2.5.3. <i>EfficientNet</i>	24
3. Analiza obszarów istotności	29
3.1. Uzasadnienie potrzeby stosowania metod analizy istotności	29
3.2. Dostępne metody analizy istotności	30
3.2.1. Sieci dekonwolucyjne – Zeiler i Fergus.....	30
3.2.2. Algorytm nadzorowanej propagacji wstecznej – Jost Springenberg	33
3.2.3. Algorytm mapowania aktywacji klas <i>CAM</i> – Bolei Zhou	36
3.2.4. <i>Grad-CAM</i> – Ramprasaath R. Selvaraju.....	38
4. Sposób prowadzenia testów.....	45
4.1. Środowisko programistyczne i wykorzystane biblioteki	45
4.2. Zbiór danych.....	45

4.3. Przygotowanie danych.....	46
4.4. Wybrane architektury i metody wyznaczania map istotności.....	47
4.4.1. Model bazujący na sieci <i>VGG-16</i>	47
4.4.2. Model bazujący na sieci <i>ResNet50</i>	49
4.4.3. Model bazujący na sieci <i>EfficientNet</i>	51
4.4.4. Prosta sieć konwolucyjna.....	53
4.4.5. Porównanie wybranych architektur	55
4.5. Metody analizy obszarów istotności.....	56
4.6. Metodologia testów	58
4.6.1. Testy manualne	58
4.6.2. Porównanie statystyczne	58
5. Przedstawienie wyników	61
5.1. Wyniki testów manualnych.....	61
5.2. Wyniki testów statystycznych.....	68
5.2.1. Izolowane testy metod dla różnych architektur sieci	68
5.2.2. Testy porównawcze metody <i>CAM</i> oraz <i>Grad-CAM</i>	74
6. Podsumowanie	77

1. Wstęp

Sieci neuronowe stanowią podstawę głębokiego uczenia. Nadają się one idealnie do realizowania skomplikowanych zadań uczenia maszynowego. Dzięki nim możemy w bardzo krótkim czasie, w celu klasyfikacji, przeanalizować ogromne ilości zdjęć, rozwijać usługi rozpoznawania mowy a także tworzyć złożone systemy rekomendacji bądź prognozowania cen.

Zdecydowanie jednym z najważniejszych sektorów, w którym wykorzystywane są sieci neuronowe jest medycyna. Pozwalają one między innymi na bardzo dokładną analizę obrazów medycznych, wspierając tym samym lekarzy w podejmowaniu niezwykle istotnych dla życia pacjentów decyzji dotyczących dalszego leczenia. Uważa się nawet, że sieci neuronowe czasami są w stanie wychwycić zniekształcenia, które są niezauważalne dla lekarza.

Niestety, nawet sieci neuronowe mają swoje słabości. W przeciwieństwie do prostych modeli, takich jak regresja liniowa czy drzewa regresyjne, których wyniki działania są łatwo interpretowalne bez stosowania dodatkowych metod analizy, w przypadku sieci neuronowych często ciężko jest stwierdzić, dlaczego wynik działania jest taki a nie inny. Jest to szczególnie ważne w przypadku takich dziedzin jak medycyna czy farmakologia, w których błędnie podjęta decyzja może mieć tragiczne skutki. Dlatego też, specjalisci często nie są w stanie zaufać wynikom działania sieci neuronowych i nawet jeśli sieć neuronowa ma bardzo dobrą skuteczność, szansa na jej uznanie jest mała.

Powstało jednak kilka metod pozwalających na interpretowanie działania sieci neuronowych. Jedną z możliwości jest stosowanie map, które pozwalają na analizę obszarów istotności. Za pomocą map istotności (z ang. *saliency maps*) możemy uwidoczyć, które fragmenty obrazu miały największy wpływ na podjęcie przez sieć neuronową decyzji, dzięki czemu możemy stwierdzić czy sieć neuronowa podczas podejmowania decyzji skupiała się na prawidłowym fragmencie obrazu, który zawiera najcenniejsze informacje.

Dlatego też, w niniejszej pracy zdecydowałem się przeprowadzić testy, które pozwolą porównać aktualnie dostępne metody tworzenia map istotności w celu sprawdzenia w jakim stopniu rozwiązują one problem interpretowalności w przypadku analizy obrazów medycznych. Przy wyborze tematu pracy kierowałem się także możliwością dogłębniego zrozumienia działania sieci neuronowych oraz poszerzeniem mojej wiedzy na temat interpretowalności ich wyników.

1.1. Cel pracy

Celem pracy jest zastosowanie metod oceny istotności w analizie wyników klasyfikacji obrazów medycznych przez głębokie sieci neuronowe. Dla wybranych architektur głębokich sieci neuronowych zamierzam zastosować różne metody analizy istotności. Uzyskane wyniki zostaną poddane analizie w celu oceny możliwości zastosowania map istotności w analizie obrazów medycznych.

1.2. Zakres pracy

Rozdział 2 poniższej pracy poświęciłem wprowadzeniu kilku pojęć, które są niezbędne, w celu zrozumienia tematyki, którą obejmuje jej temat. W tym celu, korzystając z literatury, wyjaśniłem działanie sztucznych neuronów, sieci neuronowych, a także wprowadziłem kilka pojęć, które uznałem za konieczne. Wyjaśniłem także działanie konwolucyjnych sieci neuronowych oraz zaprezentowałem gotowe architektury sieci neuronowych, służące do analizy obrazów.

Głównym tematem pracy była jednak analiza aktualnego stanu wiedzy i rozwoju technologii w rozważanej dziedzinie, którą jest analiza obszarów istotności dla obrazów medycznych. W tym celu dokonałem dogłębniego przeglądu literatury w celu poszerzenia wiedzy na temat metod interpretacji wyników działania głębokich sieci neuronowych. Wiedzę teoretyczną dotyczącą tego zagadnienia zawarłem w rozdziale 3.

Rozdział 4 poświęciłem opisowi parametrów wykorzystanych do statystycznego porównania metod analizy istotności. Dodatkowo, uzasadniłem w nich wybór konkretnych architektur sieci neuronowych, a także przedstawiłem dwie metodologie prowadzenia testów z zastosowaniem zróżnicowanych zbiorów uczących.

Rozdział 5 oraz 6 tej pracy poświęciłem przedstawieniu wyników prowadzonych testów oraz wyciągnięciu wniosków na temat aktualnego stanu wiedzy w zakresie metod wyznaczania map istotności w celu analizy obszarów istotności.

2. Sieci Neuronowe

Moją pracę inżynierską chciałbym zacząć od wytłumaczenia najważniejszych pojęć z nią związanych, aby następnie przejść do bardziej szczegółowych informacji oraz analizy obszarów istotności dla konkretnie sformułowanego problemu.

„Sieci neuronowe” swoją nazwę zawdzięczają temu, że jest to model uczenia maszynowego zainspirowany występującymi w naszych mózgach sieciami neuronów biologicznych. Jednak mimo, że samoloty są inspirowane ptakami, to nie muszą machać skrzydłami. Analogicznie, mimo że sieci neuronowe wywodzą się od biologicznych neuronów, to wiele naukowców uważa, że powinniśmy porzucić tę analogię, aby nie ograniczać naszej twórczości tylko do tworów możliwych do istnienia w naturze.[1]

Sieci neuronowe zbudowane są z perceptronów (sztucznych neuronów), które połączone są ze sobą, tworząc bardziej skomplikowaną strukturę. Aby dokładnie zrozumieć działanie sieci neuronowych, należy najpierw wytłumaczyć jak działa pojedynczy perceptron.

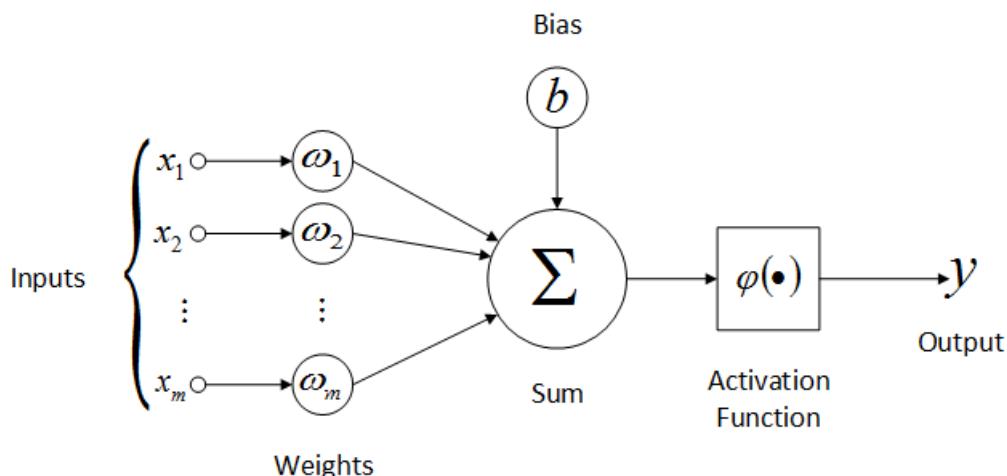
2.1. Budowa i zasada działania perceptronu

Perceptron, zaproponowany w 1957 roku przez Franka Rosenblatta, stanowi najmniej złożoną sztuczną sieć neuronową. Jego podstawę stanowi sztuczny neuron, zwany progową jednostką logiczną (z ang. *Threshold Logic Unit – TLU*).[1]

Rysunek 2.1. przedstawia budowę pojedynczego perceptronu. Składa się on z czterech elementów:

1. wartości wejściowych
2. wag i wartości obciążenia
3. funkcji sumującej
4. funkcji aktywacji - na przykład funkcja skokowa heaviside'a

Działanie perceptronu jest następujące: wartościami wejść/wyjść są liczby, a każde połączenie ma przyporządkowaną wagę. Perceptron wylicza ważoną sumę sygnałów wejściowych oraz dodaje do tej sumy wartość obciążenia:



Rys. 2.1. Budowa perceptronu[2]

$$z = b + x_1w_1 + \dots + x_nw_n = x^T w + b \quad (2.1)$$

Następnie zostaje użyta funkcja aktywacji wobec ważonej sumy, co daje ostateczny wynik – wartość wyjścia z perceptronu:

$$y = \phi(z) \quad (2.2)$$

Perceptron składa się z jednej warstwy jednostek *TLU*, gdzie każdy neuron jest połączony ze wszystkimi wejściami. Tego typu warstwa jest nazywana warstwą w pełni połączoną. Wejścia są przekazywane do sztucznych neuronów, które nazywane są także warstwą wejściową (z ang. *input layer*) i ich zadaniem jest przekazywanie dalej wszystkich dostarczonych do nich danych. Warto dodać, że perceptron może mieć więcej niż jedno wyjście. Pozwala to na klasyfikowanie próbek do kilku różnych klas, co stanowi klasyfikator wielowyjściowy.[1]

2.1.1. Trenowanie perceptronu

Algorytm uczący zaproponowany przez Franka Rosenblatta był w znacznej mierze inspirowany regułą Hebb'a. Mówi ona, że gdy biologiczny neuron często pobudza inną komórkę nerwową, to połaczenie pomiędzy tymi dwoma neuronami staje się silniejsze. Koncepcja ta została później podsumowana przez Siegrida Löwela: "*Cells that fire together, wire together*", czyli w wolnym tłumaczeniu: „Komórki pobudzające się wzajemnie wiążą się ze sobą”, Oznacza to, że waga połączenia pomiędzy dwoma neuronami zazwyczaj staje się coraz większa w przypadku, gdy są one aktywowane jednocześnie. Reguła ta została nazwana regułą Hebb'a.

Perceptrony są uczone za pomocą odmiany tej reguły, w której jest brany pod uwagę błąd popełniany przez sieć podczas prognozowania wyniku – oznacza to, że wzmacniane są połączenia pomagające zmniejszyć wartość błędu. Dokładniej mówiąc perceptron za każdym razem przetwarza jeden przykład uczący i wylicza dla niego prognozy. Dla każdego neuronu wyjściowego odpowiedzialnego za nieprawidłowy wynik zwiększone są wagi połączeń ze wszystkimi wejściami przyczyniającymi się do właściwej prognozy:

$$w_{i,j}^{k+1} = w_{i,j}^k + \eta(y_j - \bar{y}_j)x_i \quad (2.3)$$

gdzie

$w_{i,j}$ – waga połączenia pomiędzy i-tym wejściowym i j-tym wyjściowym neuronem

x_i – i-ta wartość wejściowa bieżącego przykładu uczącego

\bar{y}_j – wynik j-tego neuronu wyjściowego dla bieżącego przykładu uczącego

y_j – oczekiwany wynik j-tego neuronu wyjściowego dla bieżącego przykładu uczącego

η – współczynnik uczenia

k – krok

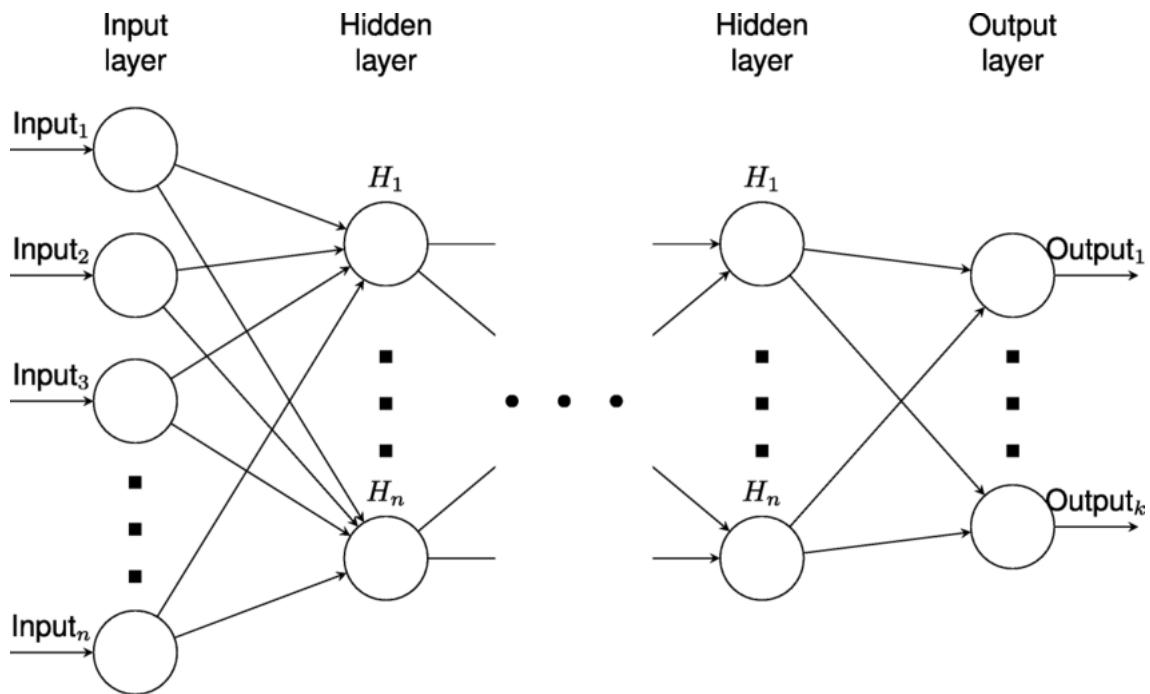
Warto jednak dodać, że Rosenblatt udowodnił, że jeśli próbki uczące są liniowo rozdzielne, to algorytm uczenia osiągnie zbieżność i sformułował tę tezę w twierdzeniu o zbieżności perceptronu.[1]

Perceptron nie jest w stanie uczyć się skomplikowanych wzorców, dlatego też w następnym podrozdziale omówię bardziej skomplikowane struktury sieci neuronowych i towarzyszący im algorytm uczenia.

2.2. Omówienie działania głębszych sieci neuronowych

Jak już wspomniałem w poprzednim rozdziale, perceptrony w swojej najprostszej postaci mają wiele poważnych wad. Część z nich została opisana przez Marvina Minsky i Seymoura Paperta w monografii „*Perceptrons*” z 1969 roku. Przykładową wadą perceptronu jest niemożność rozwiązywania trywialnych problemów, między innymi zadania klasyfikacyjnego alternatywy rozłącznej – XOR.

Okazuje się jednak, że część ograniczeń możemy wyeliminować tworząc wiele warstw perceptronów. Perceptron wielowarstwowy *MLP* (z ang. *Multi-Layer Perceptron*) składa się z jednej warstwy wejściowej, co najmniej jednej warstwy jednostek *TLU* (warstwy ukrytej, z ang. *hidden layer*) oraz ostatniej warstwy jednostek *TLU* – warstwy wyjściowej.



Rys. 2.2. Model sieci neuronowej - perceptronu wielowarstwowego[3]

Warstwy znajdujące się blisko warstwy wyjściowej nazywane są warstwami górnymi (z ang. *upper layer*), natomiast warstwy znajdujące się blisko warstwy wejściowej nazywane są warstwami dolnymi (z ang. *lower layer*). Każda warstwa zawiera neuron obciążający (mimo że nie został on ukazany na rysunku 2.2 – częsta praktyka) i jest w pełni połączona z następną warstwą.

Sztuczna sieć neuronowa nazywana jest głęboką siecią neuronową (z ang. *Deep Neural Network – DNN*) jeśli posiada wiele warstw ukrytych. Warto dodać, że pojęcie głęboka jest dość względne – może oznaczać kilka, ale nawet ponad 100 warstw ukrytych.

2.2.1. Trenowanie głębokiej sieci neuronowej

Naukowcy próbowali znaleźć sposób uczenia głębokich sieci neuronowych przez wiele lat. Udało się to dopiero w 1985 grupie naukowców w której skład wchodzili: David Rumelhart, Geoffrey Hinton oraz Ronald Williams. Opublikowali oni dokument[4], w którym przedstawili koncepcję algorytmu propagacji wstecznej (z ang. *backpropagation*). Algorytm ten jest stosowany po dziś dzień.[1]

Algorytm propagacji wstecznej wykorzystuje algorytm gradientu prostego. Przy użyciu techniki automatycznego obliczania gradientu (z wykorzystaniem różniczkowania automatycznego) algorytm propagacji wstecznej jest w stanie w dwóch przebiegach – jednym do przodu i jednym do tyłu – obliczyć gradient błędu sieci w odniesieniu do każdego parametru modelu, czyli jest w stanie określić stopień w którym należy zmodyfikować wszystkie wagie połączeń i

członów obciążających, tak aby zmniejszyć wartość błędu. Po uzyskaniu tych gradientów realizowany jest klasyczny algorytm gradientu prostego, wagi i obciążenia są aktualizowane a cały proces jest kontynuowany aż do momentu osiągnięcia zbieżności.

Warto dokładniej przyjrzeć się przebiegowi algorytmu propagacji wstecznej:

1. Pierwszym krokiem algorytmu jest tzw. propagacja w przód (z ang. *forward pass*). Polega ona na tym, że dla wszystkich neuronów w warstwie obliczamy wyniki. Następnie wyniki przekazujemy do dalszych warstw i ponawiamy proces. Krok ten różni się od uzyskiwania prognoz z sieci jedynie tym, że wszystkie wyniki pośrednie zostają zachowane, gdyż będą one potrzebne w przebiegu wstecz.
2. Mierzony jest błąd na wyjściu z sieci neuronowej. Mając wynik oczekiwany i uzyskany obliczamy dla każdego neuronu wyjściowego różnice za pomocą błędu kwadratowego z dodanym współczynnikiem o wartości 1/2, który ułatwia różniczkowanie, które zostanie wykonane w kolejnych krokach:

$$\epsilon = \sum \frac{1}{2}(y_o - y_u)^2 \quad (2.4)$$

3. Rozpoczynamy etap propagacji wstecz (z ang. *backwards pass*). Naszym celem jest uaktualnienie wartości każdej wagi i obciążenia w sieci tak aby minimalizowała ona wcześniej obliczone błędy pomiędzy wartością oczekiwana (y_o) a uzyskaną (y_u). Obliczamy wkład każdego połączenia wyjściowego w wartość błędu w sposób analityczny, za pomocą reguły łańcuchowej (z ang. *chain rule*). Przebiega to w następujący sposób: dla ostatniej warstwy ukrytej sprawdzamy jaki jest wpływ każdej z jej wag na ostateczny wynik, następnie cofamy się o jedną warstwę i powtarzamy obliczenia z wykorzystaniem reguły łańcuchowej. W ten sposób kierujemy się coraz niżej aż dotrzmemy do warstwy wejściowej – stąd nazwa propagacja wstecz.
4. Ostatnim krokiem jest zastosowanie mechanizmu gradientu prostego do uaktualnienia wszystkich wag połączeń w sieci za pomocą obliczonych w poprzednim kroku gradientów.

W przypadku sieci głębszych wykorzystywana ilość danych potrafi być ogromna. W takim przypadku dane dzieli się na mniejsze grupy (z ang. *batch*) zawierające na przykład 32 przypadki treningowe. Każdy przebieg algorytmu nazywamy epoką.

Podsumowując: algorytm propagacji wstecznej najpierw wylicza prognozę (propagacja w przód), obliczany jest błąd otrzymanych wyników, następnie rozpoczynamy etap propagacji wstecz, której celem jest wyznaczenie wkładu każdego połączenia (wagi) w zmierzony błąd. Na koniec wagi są modyfikowane (etap gradientu prostego) i cały proces jest ponawiany.

Warto dodać, że aby ten algorytm działał prawidłowo, jego twórcy musieli wprowadzić kluczową zmianę w architekturze głębokiej sieci neuronowej. Funkcja aktywacji, którą poprzednio najczęściej była funkcja skokowa heaviside'a, została zastąpiona funkcją logistyczną (sigmoidalną) o następującym wzorze:

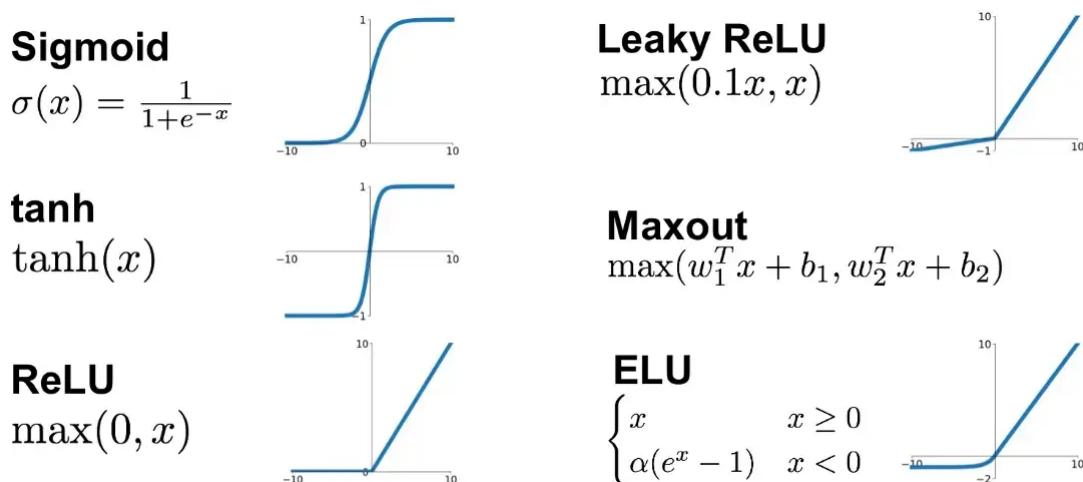
$$\varphi(z) = \frac{1}{1 + e^{-z}} \quad (2.5)$$

Było to niezbędne, ponieważ funkcja skokowa nie pozwala korzystać z gradientu (zawiera tylko płaskie segmenty, więc jej pochodna wynosi 0). W odróżnieniu od funkcji skokowej, pochodna funkcji logistycznej jest różna od zera w całej jej dziedzinie.[1]

2.2.2. Funkcje aktywacji

Zanim przejdę do kolejnego tematu, warto jeszcze wspomnieć dlaczego potrzebujemy funkcji aktywacji oraz wymienić kilka popularnych funkcji aktywacji wykorzystywanych w sieciach neuronowych. Funkcji aktywacji potrzebujemy, ponieważ nie wykorzystując ich, z połączenia kilku funkcji liniowych za pomocą przekształcenia liniowego, otrzymujemy kolejną funkcję liniową. Oznacza to, że nieważne jak głęboka byłaby nasza sieć, można by ją zredukować do sieci o jednej warstwie. Funkcji aktywacji używamy do wprowadzania pewnych nieliniowości pomiędzy warstwami, co umożliwia nam rozwiązywanie znacznie bardziej skomplikowanych problemów. W teorii wystarczająca głęboka sieć zawierająca nieliniowe funkcje aktywacji jest w stanie aproksymować dowolną funkcję ciągłą.

Przykładowe funkcje aktywacji zostały przedstawione na rysunku 2.3



Rys. 2.3. Przykładowe funkcje aktywacji[5]

2.2.3. Przetwarzanie zdjęć - wady modelu MLP

Zwykłe wielowarstwowe sieci neuronowe nie radzą sobie najlepiej w przypadku przetwarzania zdjęć. Poniższe wady zostały przytoczone z popularnego blogu dotyczącego uczenia maszynowego o nazwie „*Towards Data Science*”[6].

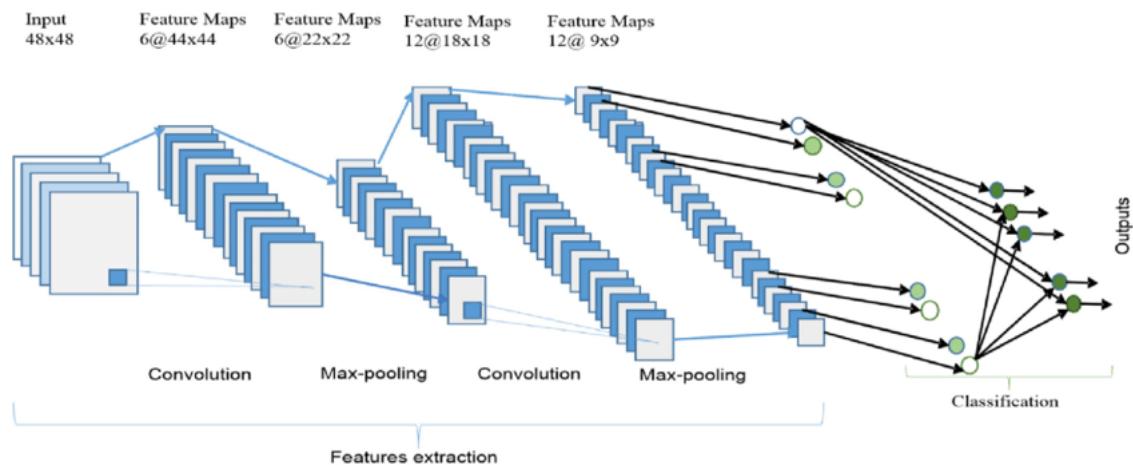
Pierwszym problemem jest to, że sieci *MLP* używają jednego perceptronu wejściowego dla każdego piksela obrazka, jeśli jest on zapisany w odcieniach szarości. Jeśli obraz, który przetwarzamy jest zapisany w formacie RGB, to każdy piksel wymaga trzech perceptronów wejściowych, co oznacza, że na każdy piksel przypadają trzy wag, które muszą zostać wyliczone i zapamiętane. Oznacza to, że dla obrazka o wymiarach 224 x 224 w skali RGB musimy aktualizować ponad 150 tysięcy wag. Znacznie wydłuża to czas trenowania sieci oraz może prowadzić do przeuczenia (z ang. *overfitting*), czyli sytuacji, gdy sieć działa bardzo dobrze dla danych treningowych, jednak dla nowych, wcześniej niewidzianych danych skuteczność jest znacznie mniejsza.

Warto także wspomnieć, że sieci *MLP* źle reagują, jeśli zdjęcia wejściowe są przesunięte bądź obrócone, a także gdy obiekt, który chcemy rozpoznać znajduje się na kolejnych obrazach w różnych miejscach. Jeśli w dużej części danych treningowych rozpoznawany obiekt znajduje się w lewym górnym rogu zdjęcia, to wielowarstwowa sieć neuronowa może uznać, że obiekt ten zawsze powinien znajdować się w tej części obrazu, źle klasyfikując obiekty, które znajdują się w innym miejscu. Wynika to z faktu, że sieć neuronowa będzie zwiększać wagę tych pikseli, na których widoczny jest rozpoznawany obiekt.

Wadą jest także utrata informacji przestrzennej podczas spłaszczenia obrazu na wejściu do modelu *MLP*. Neurony, które są blisko siebie tworzą pewne cechy obrazu, dlatego ważne jest, abyśmy byli w stanie wykorzystać informację przestrzenną, tak, abyśmy mogli nauczyć model rozpoznawania danego obiektu bez względu na to, w której części obrazu on się znajduje. Zwykłe wielowarstwowe sieci neuronowe nie mają tej umiejętności – nie potrafią korzystać z sąsiedztwa pikseli tworząc pewne cechy, które mogą zostać później wykorzystane do rozpoznawania obiektów.

2.3. Konwolucyjne sieci neuronowe

Jako, że w mojej pracy będę się zajmował obrazami, uznałem za niezbędne przedstawienie rodzajów sieci neuronowych, które są najczęściej stosowane w przypadku analizy zdjęć. Mowa oczywiście o sieciach konwolucyjnych (z ang. *CNN, Convolutional Neural Network*), które swoją nazwę zawdzięczają głównej operacji, która jest wykorzystywana do ekstrakcji cech z obrazu.



Rys. 2.4. Model konwolucyjnej sieci neuronowej[7]

Konwolucyjne sieci neuronowe pozwalają analizować obrazy korzystając z ekstrakcji cech. Ich działanie można przedstawić w następujących krokach:

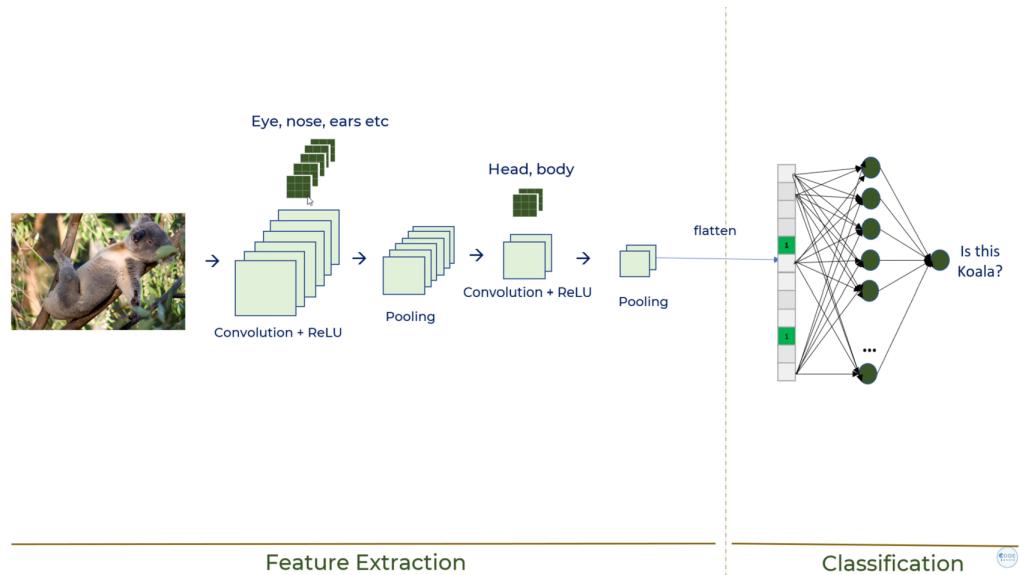
1. Na początku najczęściej stosowane jest rozszerzenie obrazu wejściowego (z ang. *padding*). Do obrazu wejściowego dodawana jest ramka o szerokości kilku pikseli (w zależności od wielkości filtra), tak aby w wyniku konwolucji wymiar obrazu powrócił do oryginalnych wartości.
2. Za pomocą filtra (często o rozmiarze 3x3 lub 5x5) sprawdzane jest czy w obrazie wejściowym znajdują się kształty charakterystyczne dla szukanego obiektu. Odbywa się to za pomocą konwolucji – operacji, która na podstawie oryginalnych pikseli w obrazie i filtra tworzy nowe piksele, biorąc pod uwagę sąsiedztwo. Aby tego dokonać wycinany jest lewy górny fragment obrazu wejściowego o wymiarze równym wymiarowi filtra, każdy piksel z wyciętego fragmentu przemnażany jest z odpowiadającą mu wartością z filtra, wynik jest sumowany. Następnie filtr jest przesuwany i analizowane jest kolejne sąsiedztwo a proces ten powtarzany jest aż wszystkie otoczenia z obrazu wejściowego zostaną poddane operacji konwolucji. W ten sposób powstają mapy cech (z ang. *feature map*), których liczbę możemy wybrać jako jeden z parametrów CNN. Filtry często działają w taki sposób, że jeśli w danym otoczeniu występuje szukana cecha to nowa wartość piksela jest bliska maksymalnej.

Uwaga! Wartości wag w filtrach obliczane są automatycznie (a więc i cechy charakterystyczne analizowanych obiektów wyznaczane są podczas procesu nauki sieci konwolucyjnej).

3. Następnie wykorzystywane są funkcje aktywacji – najczęściej ReLU w celu usunięcia z map cech mało znaczących informacji, które jedynie opóźniałyby trenowanie sieci

(ujemne wartości nowych pikseli świadczące o tym, że w danym otoczeniu nie ma poszukiwanych cech zastępowane są zerem, bądź inną małą wartością).

4. Kolejnym krokiem jest łączenie (z ang. *pooling*). Najczęściej stosowaną odmianą jest *max pooling*, który polega na ponownym okienkowym przejściu po mapie cech, z każdego sąsiedztwa wybierany jest piksel o największej wartości (dla łączenia uśredniającego (z ang. *average poolingu*) obliczana jest wartość średnia dla otoczenia) i całe otoczenie zostaje zastąpione tą jedną wartością, co redukuje wymiar map cech pozwalając zarazem zachować najważniejsze informacje.
5. Następnie operacje konwolucji i *poolingu* powtarzane są naprzemiennie a o ilości tych operacji decyduje architekt sieci. Każda kolejna warstwa odpowiada za detekcję bardziej abstrakcyjnych cech np. jeśli sieć miałaby być odpowiedzialna za detekcję ludzi to pierwsze warstwy wydobywałyby krawędzie, następne mogłyby być odpowiedzialne za detekcje nosa, oczu, nóg, kolejne warstwy za detekcje twarzy, tułowia i ostatnia, w pełni połączona odpowiadałaby na pytanie, czy na obrazie jest człowiek.
6. Ostatnia warstwa sieci konwolucyjnej jest zawsze w pełni połączona i odpowiada za ostateczną klasyfikację. Wejście do niej należy wcześniej spłaszczyć (z wejściowych macierzy należy zrobić wektor).



Rys. 2.5. Przykład działania sieci konwolucyjnej - detekcja koali[8]

Można więc powiedzieć, że sieci konwolucyjne rozwiązuje problemy związane ze zwykłymi wielowarstwowymi sieciami neuronowymi. Sieci konwolucyjne to na tyle ważny temat

Tabela 2.1. Zastosowanie i działanie kolejnych etapów sieci konwolucyjnych

	Działanie	Zastosowanie
Konwolucja	Za pomocą filtra wydobywa cechy charakterystyczne z obrazu.	<ul style="list-style-type: none"> – Powoduje, że połączenia są rzadsze niż w przypadku zwykłych sieci <i>MLP</i>, dzięki czemu sieć neuronowa jest opisana przez mniejszą liczbę parametrów, co zapobiega przetrenowaniu (z ang. <i>overfitting</i>) sieci. – W połączeniu z <i>poolingiem</i> sprawia, że umiejscowienie szukanego obiektu na obrazku nie ma wpływu na detekcję.
Funkcja aktywacji	Wprowadza bardzo ważną cechę opisaną w poprzednich rozdziałach - nieliniowość.	<ul style="list-style-type: none"> – Przyspiesza trenowanie sieci. – Przyspiesza predykcje.
<i>Pooling</i>	Zastępuje pewne sąsiedztwo pikseli jedną wartością.	<ul style="list-style-type: none"> – Zmniejsza wymiarowość. – Zwiększa odporność na zniekształcenia i zmienność w obrazach, ponieważ sąsiedztwo jest zastępowane jedną wartością, ułożenie wartości w tym sąsiedztwie nie ma znaczenia.
Klasyfikacja	Przewiduje ostateczny wynik za pomocą w pełni połączonej warstwy neuronów.	<ul style="list-style-type: none"> – Zapewnia rozpoznanie obiektów różniących się od siebie, ale reprezentujących ten sam typ obiektu. Powoduje, że sieć jest odporna na różnorodność przypadków testowych.

w kontekście analizy zdjęć, że warto jeszcze raz przyjrzeć się poszczególnym etapom ich działania i podsumować korzyści płynące z podjętych kroków.

Na sam koniec warto dodać, że konwolucyjne sieci neuronowe potrzebują takiego zbioru treningowego, który zawiera w sobie różne przypadki (obrócone, różnej skali) ponieważ bez tego nie jest w stanie nauczyć się rozpoznawania obiektów o różnych rozmiarach. Jeśli nasz zbiór danych nie posiada takich przypadków to musimy je sztucznie wygenerować odpowiednio obracając, powiększając, zmniejszając i pogrubiając posiadane zdjęcia. Proces generacji takich przypadków nazywamy powiększaniem danych (z ang. *data augmentation*).

2.4. Transfer Learning

Uczenie przez douczanie (z ang. *transfer learning*) zostało pierwszy raz zaprezentowane w 1976 roku przez Stevo Bozinovski'ego. Przedstawił on matematyczną i geometryczną teorię związaną z tym zagadnieniem, a dokładny jej opis wraz z komentarzami możemy znaleźć w tym artykule [9], który ukazał się w 2020 roku. Poprzedni artykuł, z 1976 roku był bowiem napisany w języku chorwackim.

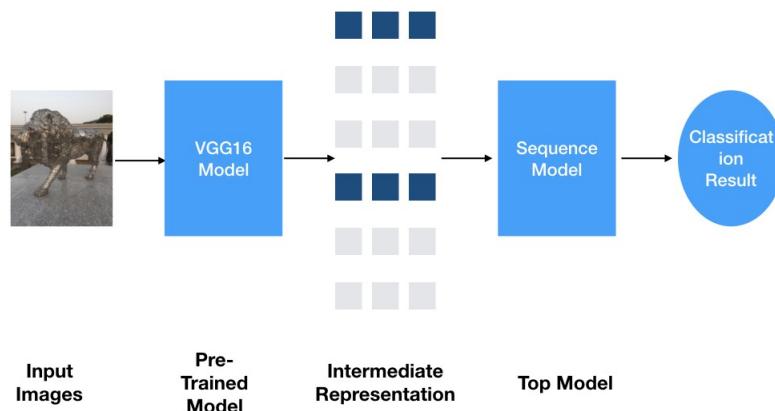
Na przestrzeni tego podrozdziału będę czerpał wiedzę z książki[10], która dokładnie omawia teorię związaną z transfer learningiem, ale także jego praktyczne wykorzystanie z użyciem języka programowania Python, w celu bardzo ogólnego omówienia tego zagadnienia.

Główną ideą stojącą za uczeniem przez douczanie jest wykorzystanie wiedzy już zdobytej do nauki nowej rzeczy. Na przykład, gdy człowiek umie już grać na klasycznym pianinie, dużo łatwiej będzie mu się nauczyć gry na pianinie jazzowym, niż gdyby zaczynał naukę od początku. Analogicznie, możemy zastosować gotowe architektury sieci (takie jak ResNet) wcześniej wytrenowane na dużym zbiorze danych, ponieważ, jak już wspomniałem wcześniej, dolne warstwy sieci konwolucyjnych uczą się prostych cech obrazów. Usuwając kilka ostatnich warstw architektury sieci wcześniej wytrenowanej oraz dodając kilka warstw konwolucyjnych a także finalną warstwę klasyfikującą (które możemy wytrenować w celu detekcji bardziej abstrakcyjnych cech charakterystycznych dla naszego nowego zagadnienia) jesteśmy w stanie w znacznie krótszym czasie otrzymać bardzo dobrze działającą sieć neuronową.

Schemat działania transfer learningu został zobrazowany na rysunku 2.6. Na początku wybieramy znany model, który został wyuczony na dużym zbiorze danych. W przypadku zdjęć często wykorzystywanym zbiorem danych jest ImageNet zawierający około 14 milionów zdjęć przynależących do ponad 20 tysięcy kategorii. Następnie decydujemy, które z wyuczonych wag chcemy zablokować (uniemożliwić zmianę ich wartości, najczęściej blokuje się wszystkie wagi z modelu bazowego, w celu przyspieszenia nauki). Kolejnym krokiem jest dodanie modelu, który umieścimy na górze wcześniej wytrenowanej sieci (na rysunku 2.6 oznaczony jako "Top

Model"). Wagi tego modelu nie są zablokowane, dostosujemy je do naszego problemu, dzięki czemu otrzymujemy model o wielu warstwach bardzo małym nakładem czasu.

Transfer Learning



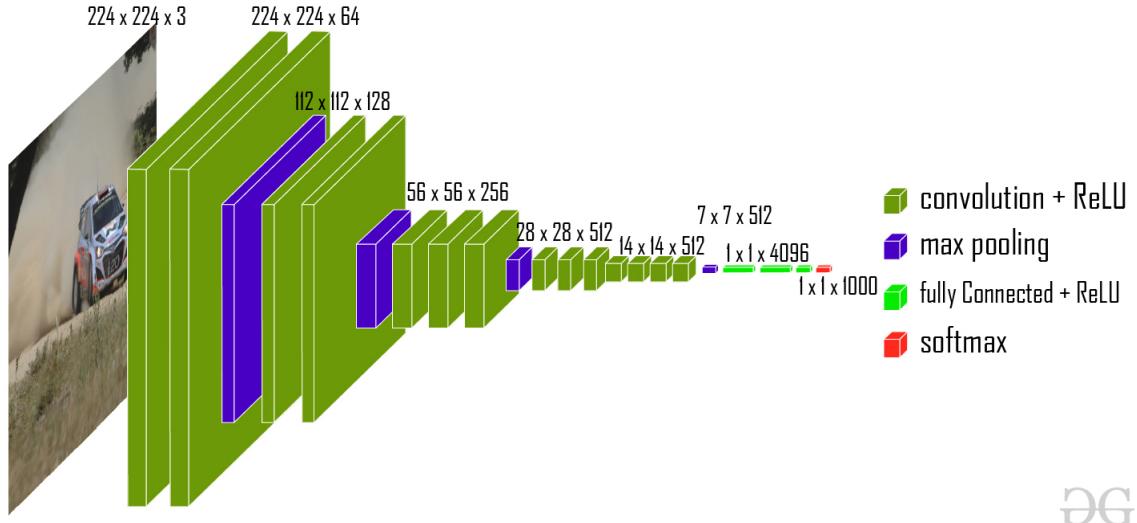
Rys. 2.6. Schemat działania transfer learningu[11]

2.5. Wybrane architektury sieci neuronowych

2.5.1. VGG

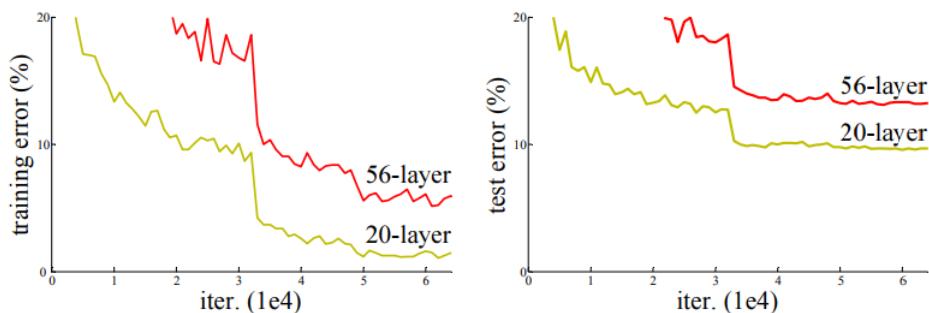
Do 2015 roku najlepiej działającą głęboką siecią konwolucyjną była sieć VGG (z ang. *Visual Geometry Group*), która została przedstawiona w artykule pod tytułem *Very Deep Convolutional Networks for Large-Scale Image Recognition*[12]. Liczyła ona w zależności od wersji od 11 do 19 warstw i działała w taki sposób, że w kolejnych jej warstwach zmniejszano rozdzielcość obrazu wejściowego zarazem zwiększając liczbę używanych filtrów (rys. 2.7). Warto dodać, że w przypadku warstw konwolucyjnych jedyną używaną przez twórców funkcją aktywacji była ReLU, a po wszystkich warstwach konwolucyjnych występowały trzy warstwy w pełni połączone. Dwie pierwsze z nich składały się z 4096 neuronów, a ostatnia, ze względu na rozwiązywany problem zawierała 1000 neuronów oraz funkcję aktywacji *softmax*.

Przełomowym na tamte czasy rozwiązaniem było używanie jedynie filtrów o wymiarach 3x3. Architekci wykorzystali fakt, że trzy takie filtry następujące po sobie mają efektywność filtru o wymiarze 7x7, wykorzystując po drodze aż trzy funkcje nieliniowe zamiast jednej, dodatkowo zmniejszając liczbę parametrów - jeden filtr 7x7 wymaga 81% więcej parametrów niż trzy filtry 3x3.



Rys. 2.7. Architektura sieci VGG[13]

Twórcy tej sieci podejmowali próby nauki głębszych sieci, jednak zwiększanie liczby warstw ukrytych nie poprawiało skuteczności sieci, a nawet ją pogarszało, co pokazuje Rysunek 2.8 zaprezentowany w artykule prezentującym sieć ResNet[14]. Możemy zauważyć, że sieci konwolucyjne, które składają się z 56 warstw mają gorszą skuteczność w przypadku danych testowych, jak i treningowych. Okazuje się więc, że źródłem problemu nie jest przeuczenie (z ang. *overfitting*) a zanikający gradient (z ang. *vanishing gradient*) oraz degradacja (z ang. *degradation*).



Rys. 2.8. Porównanie krzywych uczenia zwykłych sieci konwolucyjnych z 20 i 56 warstwami

Problem zanikającego gradientu wpływa głównie na wagę pierwszych warstw sieci neuronowych. Polega on na tym, że jeśli wartość gradientu jest bliska零 to zgodnie z algorytmem propagacji wstecznej, który omówiłem we wcześniejszych rozdziałach tej pracy, wartości wag pierwszych warstw sieci będzie praktycznie stała (zmiany będą bardzo małe ponieważ są one wprost proporcjonalne do wartości gradientu, która jest bliska零). Pogarsza to zbieżność algorytmu propagacji wstecznej, powodując, że zmiany wag sieci neuronowej są tak małe, że sieć

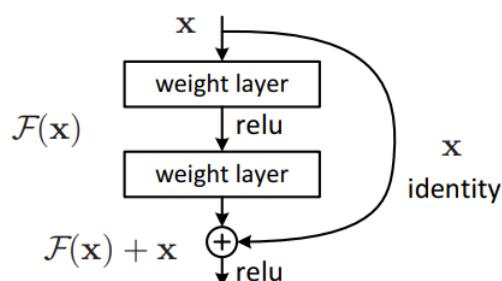
prawie w ogóle nie poprawia swojej skuteczności. Problem zanikającego gradientu pojawia się głównie wtedy, gdy sieć neuronowa ma wiele warstw, ponieważ w takim przypadku, aby obliczyć gradient dla wag pierwszych warstw sieci, musimy pomnożyć wiele wartości ze sobą, a część z nich jest bardzo mała (znacznie mniejsza niż 1), więc w wyniku takiego mnożenia otrzymujemy gradient o bardzo małej wartości. To, że wiele wartości jest bliskich zera wynika z używanych funkcji aktywacji a także z inicjalizacji wag wartościami bliskimi zera. Rozwiązaniami problemu zanikającego gradientu są miedzy innymi inicjalizacja znormalizowana oraz pośrednie warstwy normalizujące.

Warto zauważyć, że sieć która ma więcej warstw powinna działać przynajmniej tak samo dobrze jak sieć płytsza od niej. Wynika to z faktu, że jeśli płytsza sieć jest w stanie nauczyć się jakiejś funkcji, to sieć głębsza powinna być w stanie wykorzystać pierwsze warstwy do nauki tej samej funkcji, a kolejne warstwy jako funkcje tożsamościowe (z ang. *identity functions*) nie powinny pogarszać wyniku.

Problemem jednak jest to, że funkcja tożsamościowa, która w przypadku konwolucyjnych sieci neuronowych jest filtrem, który składa się z jedynki na środku i zer dokoła niej, jest tak samo trudna do nauczenia jak wszystkie inne funkcje. Problem ten został rozwiązany dzięki następcy sieci VGG - sieci ResNet.

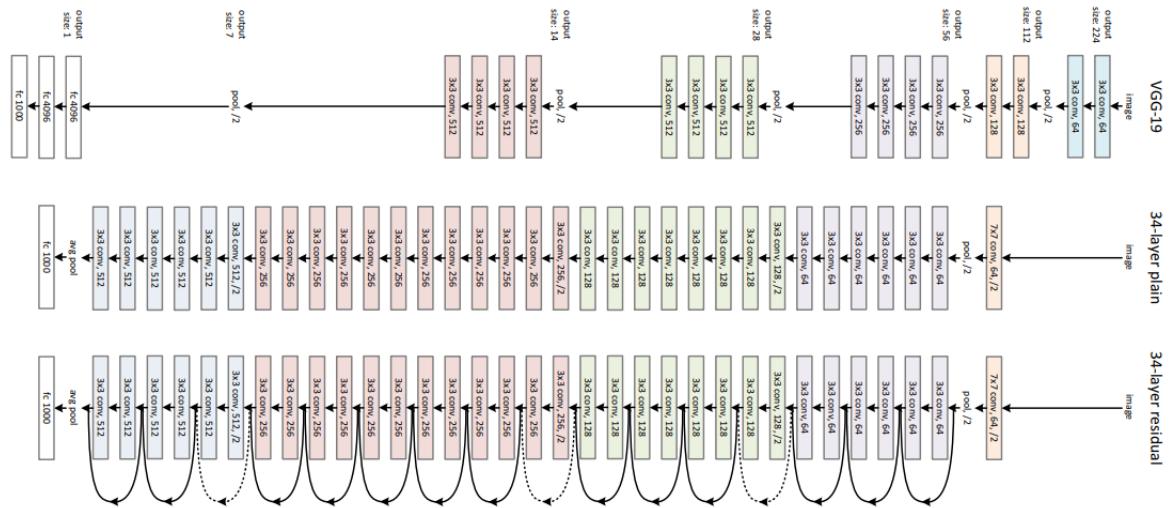
2.5.2. ResNet

Sieć ResNet (z ang. *Residual Networks*) została zaprezentowana w 2015 w dokumencie[14] autorstwa Kaiming He, do którego odwouję się na przestrzeni tego podrozdziału. Autorzy tego rozwiązania zauważili, że bardziej optymalnym rozwiązaniem od nauki kolejnych połączeń funkcji tożsamościowych, jest pominięcie kilku połączeń i nauczenie filtrów pomiędzy funkcji bliskich zera (ponieważ wagi inicjalizowane są wartościami bliskimi zera) a następnie zsumowanie wyników obu operacji, tak jak to pokazano na rysunku 2.9 Dzięki tej operacji otrzymujemy ekwiwalent funkcji tożsamościowej w prostszy sposób (jeśli tylko zajdzie taka potrzeba).



Rys. 2.9. Podejście zastosowane w sieciach ResNet

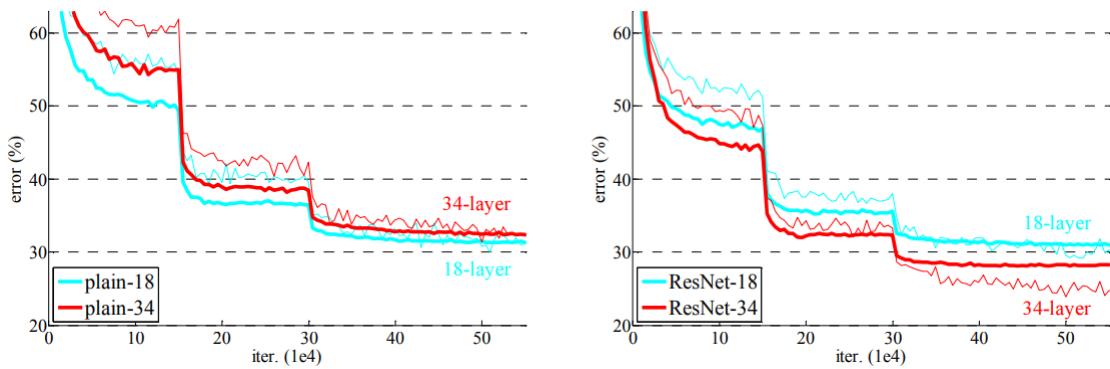
Sieć ResNet jest zbudowana w taki sposób, że połączenia rezydualne występują co kilka warstw. Na rysunku 2.10 przedstawiono porównanie budowy sieci VGG-19, zwykłej sieci konwolucyjnej posiadającej 34 warstwy oraz sieci rezydualnej posiadającej 34 warstwy.



Rys. 2.10. Porównanie architektur różnych sieci konwolucyjnych

Warto dodać, że pomimo, że sieci ResNet mają nawet 152 warstwy, dzięki zastosowaniu połączeń rezydualnych są one mniej złożone od sieci VGG posiadających 19 warstw - sieć ResNet-152 wykonuje około 11.3 miliardów operacji dodawania/mnożenia, gdy jej odpowiednik VGG-19 potrzebuje około 19.6 miliarda operacji.

Należy się także przyjrzeć porównaniu krzywych uczenia w przypadku prostych sieci konwolucyjnych a także sieci rezydualnych. Możemy zauważyć, że w przypadku sieci rezydualnych zwiększanie liczby warstw poprawia ich skuteczność.

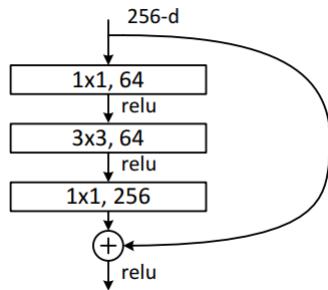


Rys. 2.11. Porównanie krzywych uczenia sieci prostych i rezydualnych

Potwierdza to także tabela prezentująca procentowe błędy detekcji za pomocą pojedynczego modelu dla zbioru danych walidacyjnych ImageNet 2015.

method	top-1 err.	top-5 err.
VGG [40] (ILSVRC'14)	-	8.43 [†]
GoogLeNet [43] (ILSVRC'14)	-	7.89
VGG [40] (v5)	24.4	7.1
PReLU-net [12]	21.59	5.71
BN-inception [16]	21.99	5.81
ResNet-34 B	21.84	5.71
ResNet-34 C	21.53	5.60
ResNet-50	20.74	5.25
ResNet-101	19.87	4.60
ResNet-152	19.38	4.49

Rys. 2.12. Porównanie wyników dla zbioru danych walidacyjnych ImageNet 2015 oraz różnych architektur sieci

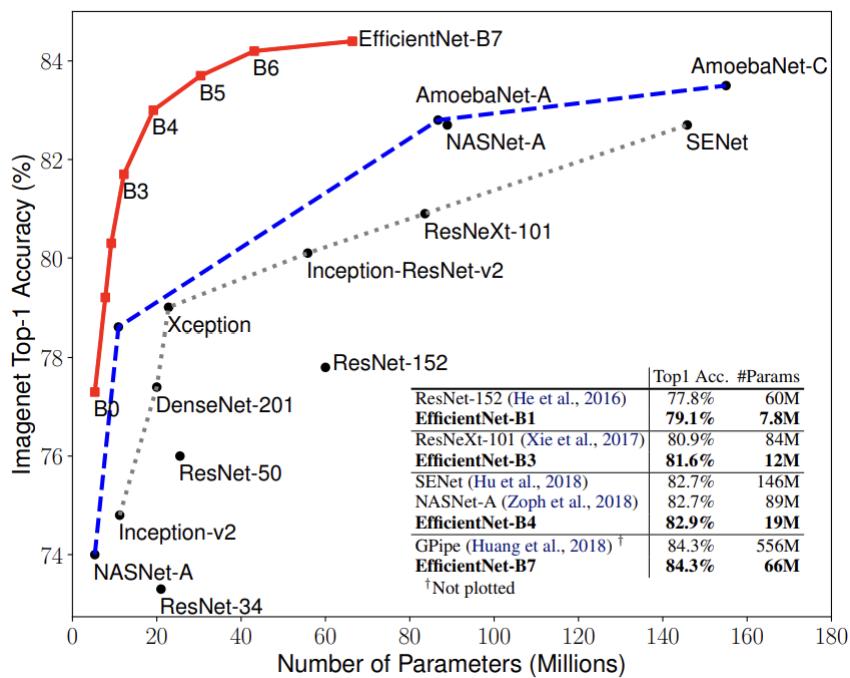


Rys. 2.13. Struktura przyspieszająca obliczenia zastosowana w sieciach ResNet

Na koniec warto dodać, że sieci ResNet z każdą dodawaną warstwą mają coraz większą skuteczność (dopóki nie zaczniemy doświadczać przetrenowania) a także im więcej warstw sieć ma, tym bardziej rezydualne połączenia wpływają pozytywnie na jej działanie. Dodatkowo, dla sieci ResNet, które posiadają więcej niż 34 warstwy często stosuje się specjalne struktury przyspieszające wykonywanie obliczeń przedstawione na rysunku 2.13 Składają się one ze trzech warstw: warstwy zwężającej (filtr 1x1), warstwy szukającej cech (filtr 3x3) oraz warstwy zwiększającej wymiary do pierwotnych (filtr 1x1). Dzięki temu warstwa szukająca cech za pomocą filtra 3x3 wykonuje operacje na mniejszym obrazie, co przyspiesza obliczenia.

2.5.3. EfficientNet

Sieć konwolucyjna EfficientNet została zaprezentowana w 2019 roku w artykule pt. "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks"[15]. Jak nazwa artykułu wskazuje, autorzy skupili się na analizie efektywności skalowania poszczególnych wymiarów sieci konwolucyjnych oraz jej wpływu na skuteczność prognozowania.



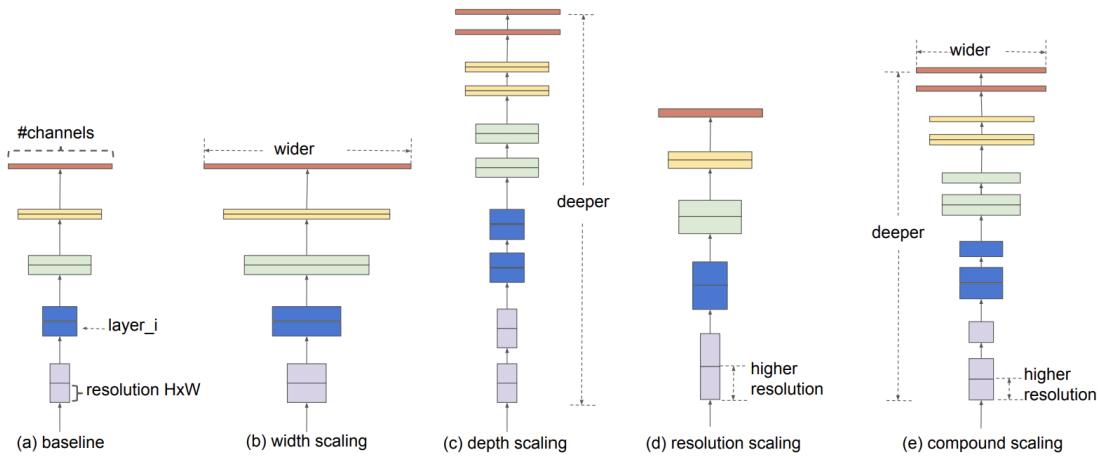
Rys. 2.14. Precyza sieci EfficientNet w porównaniu z innymi znymi architekturami

Jak można zauważyć na rysunku 2.14 sieć EfficientNet deklasuje inne znane architektury sieci pod kątem precyzji dla zbioru zdjęć ImageNet zarazem posiadając względnie małą liczbę parametrów. Dołączona do wykresu tabela porównuje kolejne wersje sieci EfficientNet z dającymi podobną precyzję innymi architekturami. Okazuje się, że sieć EfficientNet w każdym z przypadków jest opisana przez znacznie mniejszą liczbę parametrów. Na przykład sieć EfficientNet-B7 ma taką samą precyzję jak sieć GPipe, jest jednak opisana przez 8.4 razy mniej parametrów dzięki czemu jest 6.1 razy szybsza.

Jednak na jakiej zasadzie działa sieć EfficientNet? Autorzy wspomnianego wcześniej artykułu przeanalizowali możliwości skalowania modeli. Uznali, że model możemy skalować w trzech wymiarach:

1. szerokość sieci (liczba filtrów sieci konwolucyjnych)
2. głębokość sieci (liczba warstw ukrytych)
3. rozdzielcość obrazu wejściowego

Obrazuje to rysunek 2.15. Autorzy zauważyl, że najlepsze efekty daje skalowanie sieci we wszystkich kierunkach jednocześnie oraz zaproponowali nową metodę skalowania złożonego (z ang. *compound scaling method*), która korzysta z parametru ϕ w następujący sposób:



Rys. 2.15. Skalowanie modeli sieci: (a) obrazuje bazowy model, (b) oznacza skalowanie wszerz, (c) obrazuje skalowanie wgłęb, (d) zwiększenie rozdzielczości obrazu wejściowego, (e) prezentuje skalowanie w każdym wymiarze jednocześnie

$$\begin{cases} \text{depth : } d = \alpha^\phi \\ \text{width : } w = \beta^\phi \\ \text{resolution : } r = \gamma^\phi \end{cases} \quad (2.6)$$

przy założeniach, że $\alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$ oraz $\alpha \geq 1, \beta \geq 1, \gamma \geq 1$ gdzie α, β oraz γ mogą być wyznaczone za pomocą manualnego przeszukiwania małej siatki wartości. Intuicyjnie ϕ to parametr, który oznacza jak wiele zasobów obliczeniowych mamy do przydzielenia, a parametry α, β oraz γ wskazują jak chcemy te zasoby rozlokować względem każdego z wymiarów.

W celu znacznego polepszenia działania sieci korzystając z powyżej przedstawionej metody skalowania wymagane jest także, aby architektura bazowa (2.15) była stosowna. Przy tworzeniu sieci EfficientNet autorzy zdefiniowali architekturę bazową składającą się głównie z mobilnych odwróconych wąskich gardeł (z ang. *mobile inverted bottleneck*).

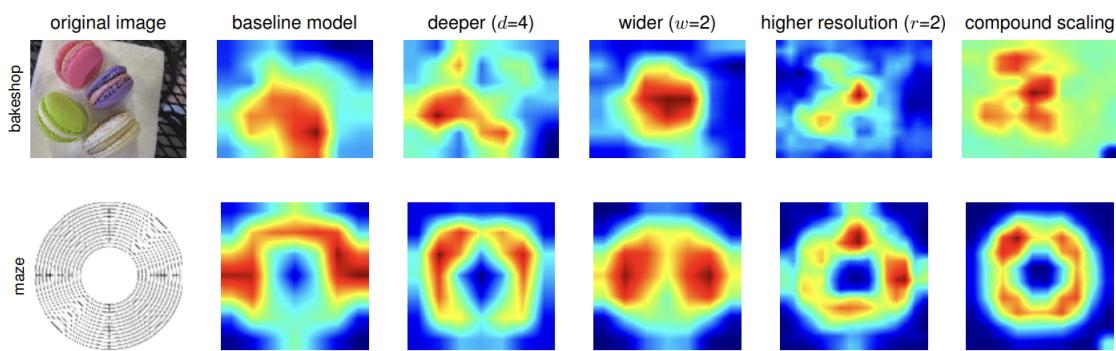
Następnie zoptymalizowali jej architekturę:

1. Krok 1: Ustalono parametr $\phi = 1$, założono, że jest dostępnych dwa razy więcej zasobów obliczeniowych i zastosowano poszukiwanie na małej siatce dla parametrów α, β oraz γ . Tym sposobem znaleziono najbardziej optymalne wartości tych parametrów dla danego modelu bazowego, kolejno: 1.2, 1.1, 1.15 dla ograniczenia $\alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$.
2. Krok 2: Uznano parametry α, β oraz γ za stałe i skalowano sieć bazową za pomocą różnych wartości parametru ϕ korzystając z układu równań 2.6.

Stage i	Operator $\hat{\mathcal{F}}_i$	Resolution $\hat{H}_i \times \hat{W}_i$	#Channels \hat{C}_i	#Layers \hat{L}_i
1	Conv3x3	224×224	32	1
2	MBConv1, k3x3	112×112	16	1
3	MBConv6, k3x3	112×112	24	2
4	MBConv6, k5x5	56×56	40	2
5	MBConv6, k3x3	28×28	80	3
6	MBConv6, k5x5	14×14	112	3
7	MBConv6, k5x5	14×14	192	4
8	MBConv6, k3x3	7×7	320	1
9	Conv1x1 & Pooling & FC	7×7	1280	1

Rys. 2.16. Bazowy model stosowany w sieciach EfficientNet

Autorzy w opublikowanym artykule korzystają także z map istotności do oceny działania ich modelu, co podkreśla wagę tej metody dla oceny działania głębokich sieci neuronowych. Okazuje się, że skalowanie sieci w każdym wymiarze z osobna poprawia jej działanie, jednak sieć uzyskana za pomocą skalowania złożonego dla wszystkich wymiarów jednocześnie skupia się na najważniejszych fragmentach zdjęć wejściowych.



Rys. 2.17. Mapy istotności w ocenie działania modelu bazowego oraz modeli uzyskanych dla różnych rodzajów skalowań.

3. Analiza obszarów istotności

3.1. Uzasadnienie potrzeby stosowania metod analizy istotności

Konwolucyjne sieci neuronowe działają bardzo dobrze w zastosowaniu jakim jest klasyfikacja obrazów, jednak jak już wspomniałem na wstępie tej pracy, specjalisci często nie są w stanie zaufać ich wynikom, ponieważ sieci nie odpowiadają na podstawowe pytania: Dlaczego podjęto taką decyzję, a nie inną? Która część obrazu wejściowego o niej zadecydowała?

Mapy istotności (z ang. *saliency maps*) pozwalają na wizualizację pikseli, które miały największy wpływ na podjętą przez konwolucyjną sieć neuronową decyzję. Odbywa się to poprzez stworzenie mapy ciepła (z ang. *heat map*), bądź innego przedstawienia graficznego, o wymiarze równym wymiarowi analizowanego obrazu, w którym piksele, które miały większy wpływ na podejmowanie decyzji mają większą wartość. Dzięki mapom istotności możemy mieć lepszy wgląd w podejmowane przez sieć decyzje, co może prowadzić do tworzenia lepszych architektur sieci, a także możemy wychwycić błędnie podjęte przez sieć decyzje (przed odkryciem map istotności poprawianie architektur sieci ograniczało się do metody prób i błędów co było mocno niezadowalające z naukowego punktu widzenia). Warto wspomnieć, że dzięki mapom istotności możemy dokładnie przyjrzeć się temu na czym skupia się każda warstwa sieci konwolucyjnej, co pozwala na lepsze zrozumienie procesu podejmowania decyzji.

Na rysunku 3.1 zaprezentowane zostały przykładowe mapy istotności zaczerpnięte z artykułu omawiającego możliwości wizualizacji i interpretowalności wyników działania konwolucyjnych sieci neuronowych[16].

Jak możemy zauważyć, detekcja dotyczyła widocznych na zdjęciach psa oraz owocu. Piksele, które miały największy wpływ na podejmowanie przez sieć neuronową decyzji na mapach istotności pokrywają się z lokalizacją obiektów na zdjęciach oryginalnych, co może nas utwierdzić w przekonaniu, że decyzja podjęta przez sieć nie jest losowa i wynika z poprawnej ekstrakcji cech przez sieci konwolucyjne.



Rys. 3.1. Przykładowe mapy istotności[16]

W kolejnych podrozdziałach omówię kilka dostępnych metod tworzenia map istotności, które następnie zostaną użyte w mojej pracy w celu porównania ich działania dla obrazów medycznych. Jest to szczególnie ciekawy temat, ponieważ nie jest on dokładnie zbadany oraz same sieci neuronowe są bardzo dokładnie omówione w wielu artykułach i filmach dostępnych w Internecie, a wiedza o mapach istotności jest dość ograniczona, szczególnie w języku polskim.

3.2. Dostępne metody analizy istotności

Dostępnych jest kilka metod tworzenia map istotności. Pierwsza z nich, najbardziej klasyczna powstała w 2013 roku, w momencie gdy sieci konwolucyjne działały już bardzo dobrze a problemem była interpretowalność. Metoda ta opierała się o sieci dekonwolucyjne. W następnych latach pojawiały się kolejne metody, które coraz częściej bazowały na gradientach oraz starały się zwracać lepsze wizualnie wyniki. W dalszych podrozdziałach dokonam przeglądu literatury aby ocenić aktualny stan wiedzy na temat map istotności i wybiorę z nich te, które przetestuję i poddam analizie w następnym rozdziale tej pracy.

3.2.1. Sieci dekonwolucyjne – Zeiler i Fergus

Metoda korzystająca z sieci dekonwolucyjnych została zaprezentowana w 2013 przez naukowców z Nowego Jorku w artykule naukowym[17], z którego wiedzę będę czerpał na przestrzeni tego podrozdziału. Zaproponowane przez nich rozwiązanie było nowatorską metodą wizualizacji działania konwolucyjnych sieci neuronowych, pozwalało ono na wizualizację działania pośrednich warstw ukrytych, co pozwalało na dokładną analizę procesu predykcji.

Technologia ta opierała się o sieci dekonwolucyjne i pozwalała na przekształcenie map cech uzyskanych przez sieć konwolucyjną z powrotem do przestrzeni pikseli. Przed odkryciem map istotności możliwa była wizualizacja działania pierwszej warstwy CNN, gdzie projekcja na przestrzeń pikseli jest możliwa i względnie prosta. W wyższych warstwach stosowano zejście

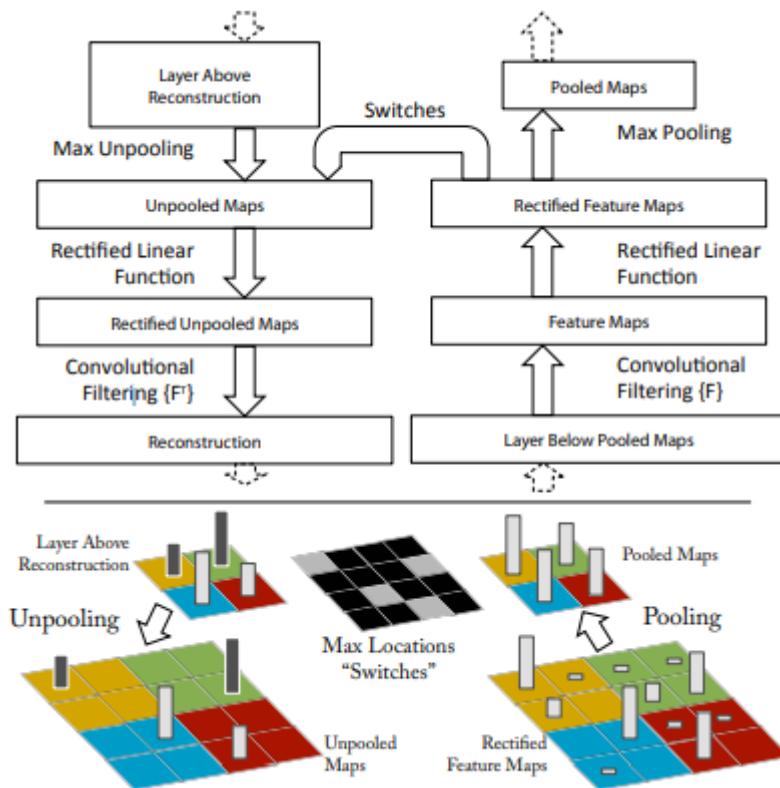
gradientowe w przestrzeni obrazu w celu znalezienia maksymalnej aktywacji jednostki (neuronu). Wymagało ono jednak uważnej inicjalizacji cech oraz nie dawało gwarancji co do niezmienności przekształcenia jednostki. Z tego powodu metoda ta została nieco rozwinięta i znaleziono sposób wyznaczania niezmienności (za pomocą Hessianu - macierzy drugich pochodnych wokół optymalnych odpowiedzi neuronów), jednak okazało się, że dla wyższych warstw był on zbyt skomplikowany i nie był możliwy do aproksymacji za pomocą zwykłej funkcji kwadratowej, jaką zaproponowano. Podejście zastosowane w mapach istotności jest inne, nieparametryczne, pokazujące które wzorce ze zbioru treningowego aktywują mapy cech. Mapy istotności wyróżniają się także tym, że są projekcjami od najwyższej warstwy do najniższej, dzięki czemu ujawniają struktury na każdym poziomie sieci konwolucyjnej.

Sieci dekonwolucyjne mogą być rozumiane jako sieci konwolucyjne, które korzystają z tych samych operacji (filtrowanie, pooling), ale w przeciwnym kierunku, a więc na przykład zamiast mapowania pikseli na mapy cech, działają na odwrót. Sieć dekonwolucyjna jest połączona z każdą warstwą sieci konwolucyjnej (jak pokazano na rysunku 3.2) zapewniając tym sposobem drogę z przestrzeni map cech do przestrzeni pikseli na każdym etapie działania sieci konwolucyjnej. W celu sprawdzenia aktywacji konkretnego neuronu, aktywacje innych neuronów są zerowane i podają mapę cech na wejście do dołączonej do danej warstwy sieci dekonwolucyjnej.

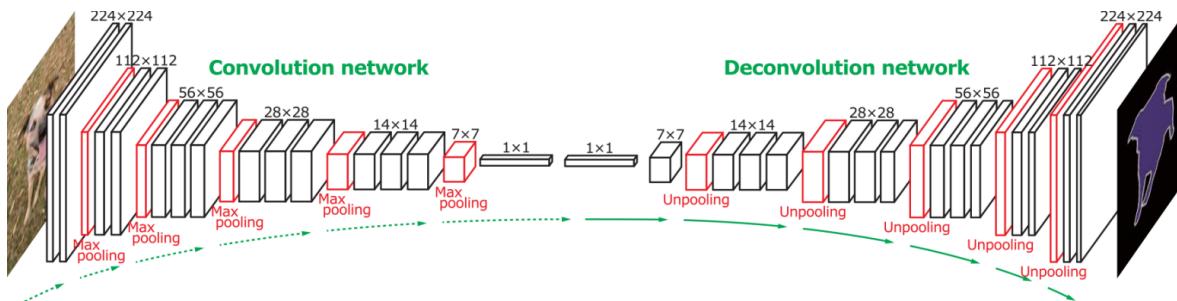
Sieć dekonwolucyjna składa się z następujących operacji (przypominających propagację wsteczną wykorzystywaną do nauki sieci neuronowych) powtarzanych aż do osiągnięcia przestrzeni pikseli - aż dotrzemy do pierwszej warstwy sieci:

1. Rozdzielenie (z ang. *unpooling*): Ponieważ podczas operacji *poolingu* tracimy informacje, nie jesteśmy w stanie przeprowadzić operacji odwrotnej do niej, możemy jedynie dokonać przybliżenia za pomocą zapamiętywania lokalizacji maksimów podczas operacji *poolingu* w każdym obszarze mapy cech. Lokalizacje maksimów zapisane są w tabeli przełączników (z ang. *switches*) - rysunek 3.2.
2. Rektyfikacja: używana jest ta sama funkcja *ReLU* co w przypadku konwolucji. Używamy jej w celu pozbycia się ujemnych wartości.
3. Filtrowanie: Na wyprostowanych mapach cech uzyskanych w poprzednim punkcie wywoływana jest operacja konwolucji, jednak zamiast oryginalnych filtrów, używamy ich transpozycji.

Rysunek 3.3. obrazuje ideę, która stoi za sieciami dekonwolucyjnymi. Pokazuje on, że jest to dokładnie odwrotna operacja do konwolucji i same sieci dekonwolucyjne mogą służyć nie tylko do tworzenia map istotności, ale także do rekonstrukcji uogólnionych obiektów.



Rys. 3.2. Schemat działania sieci dekonwolucyjnej



Rys. 3.3. Przykład działania operacji dekonwolucji dla najwyższej warstwy sieci neuronowej[18]

Jak możemy zauważyć na rysunku 3.4. pierwsze warstwy sieci neuronowych uczą się wykrywania takich cech jak proste kształty (okrągi, krawędzie), kolejne warstwy uczą się rozpoznawania bardziej skomplikowanych kształtów oraz kolorów a wyższe warstwy reprezentują całkiem abstrakcyjne elementy jak twarze psów, koła samochodów czy napisy. Jest to zgodne z ogólną teorią dotyczącą sieci konwolucyjnych, przedstawioną w rozdziale 2.3 tej pracy.



Rys. 3.4. Przykładowe mapy istotności uzyskane za pomocą sieci dekonwolucyjnych

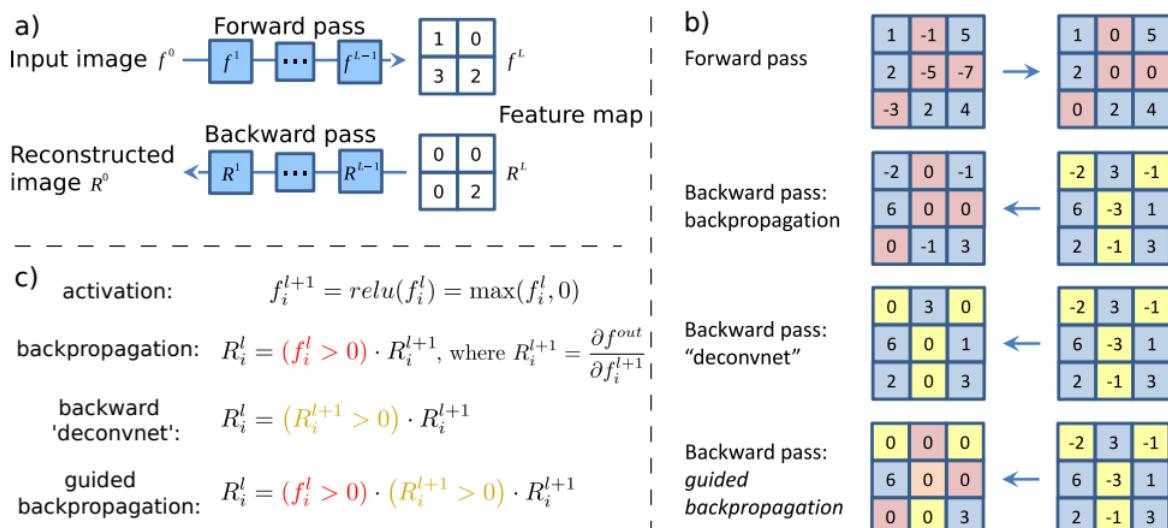
3.2.2. Algorytm nadzorowanej propagacji wstecznej – Jost Springenberg

Algorytm nadzorowanej propagacji wstecznej (z ang. *Guided Backpropagation Algorithm*) został zaprezentowany w artykule pt. "Striving for Simplicity: The All Convolutional Net"[19] z

2015 roku. Autorzy tej pracy skupili się na analizie działania najprostszej możliwej architektury CNN składającej się jedynie z warstw konwolucyjnych - warstwy dokonujące redukcji wymiaru za pomocą *poolingu* zostały zastąpione konwolucją o przesunięciu wynoszącym 2, ponieważ autorzy stwierdzili, że jeśli operacji tej używa się głównie do zmniejszenia wymiarów obrazu, to warto spróbować zmniejszenia wymiarów za pomocą zwykłej konwolucji. Okazało się, że sieć CNN o takiej architekturze jest w stanie osiągnąć dokładność podobną do najlepszych dostępnych ówcześnie sieci.

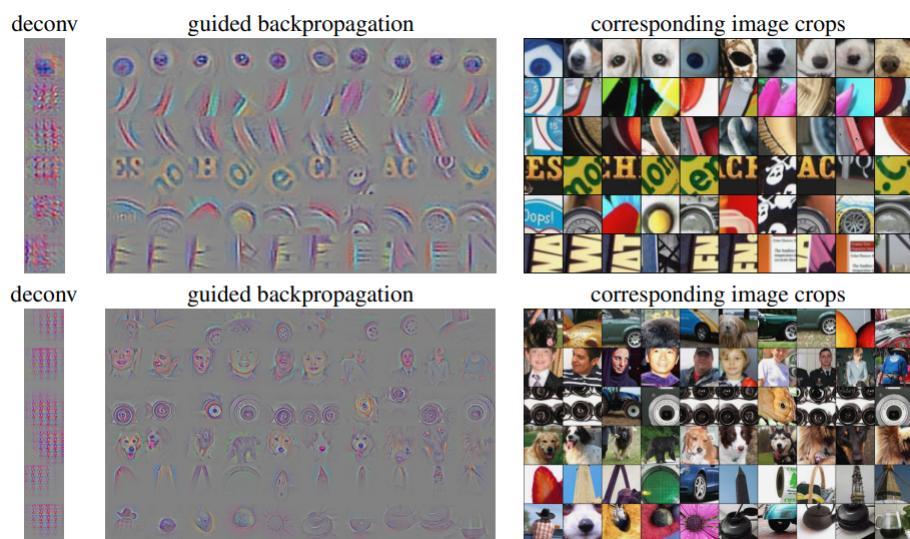
Kolejną zmianą jakiej dokonali autorzy było używanie jedynie filtrów o małych wymiarach (nie większych niż 5x5). Pozwoliło to na zmniejszenie liczby parametrów co jest formą bardzo prostej regularyzacji. Dodatkowo aby pozbyć się ostatniej warstwy, która nie jest warstwą konwolucyjną, warstwę w pełni połączoną zastąpiono warstwą konwolucyjną z filtrami o wymiarach 1x1. Prowadzi to do przewidywania klas obiektów w różnych pozycjach, które można następnie uśrednić na całym obrazie. Działanie to także prowadzi do mniejszej liczby parametrów i regularyzacji. Podsumowując, sieć konwolucyjna poddana analizie składała się z warstw konwolucyjnych, funkcji aktywacji *ReLU*, warstwy uśredniającej oraz warstwy z aktywacją *softmax* służącej do predykcji klas.

Autorzy wcześniej wspomnianej pracy do pierwszej oceny działania ich modelu używali metody wizualizacji wyników korzystającej z sieci dekonwolucyjnych, przytoczonej w poprzednim rozdziale. Zauważali oni jednak, że metoda ta bez użycia warstw *max-pooling* nie zawsze działa poprawnie, więc na przestrzeni artykułu zaprezentowali oni nową metodę, która opiera się o propagacje wstecz.



Rys. 3.5. Porównanie metod: korzystającej z sieci dekonwolucyjnych i propagacji wstecz

Metoda ta jest modyfikacją metody korzystającej z sieci dekonwolucyjnych, powoduje że wynikowe mapy istotności są znacznie ostrzejsze, szczególnie jeśli chodzi o wyższe warstwy sieci i działa w następujący sposób: najpierw dokonywane jest przejście w przód sieci, tym sposobem obliczany jest gradient aktywacji względem obrazu wejściowego. Sama dekonwolucja jest równoważna przejściu wstecz sieci, z tym wyjątkiem, że podczas propagacji poprzez funkcję nieliniową, gradient jest obliczany jedynie ze względu na gradienty o największych wartościach, sygnał dla mniejszych gradientów jest pomijany. W przypadku funkcji *ReLU* oznacza to zerowanie wszystkich gradientów z wyższej warstwy sieci o ujemnej wartości. Z drugiej strony, w przypadku propagacji wstecz, zerowane są te wartości, dla których wartości przy przejściu w przód dla obliczanej warstwy były ujemne. Metoda nadzorowanego gradientu jest połączeniem obu z tych metod, tzn. maskowane są wartości, które spełniają jeden z wymienionych wyżej warunków (rys. 3.5). Algorytm ten ma w nazwie słowo "nadzorowany" ze względu na dodatkowy sygnał z wyższych warstw w porównaniu do zwykłej propagacji wstecz. Zapobiega to wstecznemu przepływowi ujemnych gradientów, odpowiadających neuronom, które zmniejszają aktywację jednostki wyższej warstwy, którą chcemy wizualizować.



Rys. 3.6. Porównanie wyników działania sieci dekonwolucyjnych i propagacji wstecz

Ciekawą obserwacją jest także fakt, że w przypadku nie używania przełączników zapamiętujących lokalizację maksimów podczas operacji *poolingu* Nadzorowana Propagacja Wstecz działa znacznie lepiej niż metoda korzystająca z sieci dekonwolucyjnych. Oznacza to, że sygnał dodany do propagacji wstecz zastępuje przełączniki. Analizując rys. 3.6 możemy stwierdzić, że wizualizacje uzyskane za pomocą Nadzorowanej Propagacji Wstecz zawierają mniej artefaktów i są ostrzejsze.

3.2.3. Algorytm mapowania aktywacji klas **CAM** – Bolei Zhou

Sieci konwolucyjne mogą działać jako detektory obiektów. Jeśli ostatnią, w pełni połączoną warstwę neuronów zastąpimy globalnym łączeniem uśredniającym (z ang. *global average pooling - GAP*) to możemy osiągnąć bardzo dobrą jakość lokalizacji obiektów. Autorzy pracy[20] skorzystali z tej właściwości przy konstruowaniu kolejnej metody tworzenia obszarów istotności - algorytmu mapowania aktywacji klas (z ang. *Class Activation Mapping - CAM*). Metodę tę wyróżnia fakt, że lokalizacja obiektów może zostać otrzymana podczas pojedynczej propagacji w przód sieci a wyniki są bardzo ogólne. Kolejną różnicą pomiędzy metodą *CAM* a metodą korzystającą z sieci dekonwolucyjnych jest to, że sieci dekonwolucyjne analizują tylko warstwy konwolucyjne pomijając ostatnią warstwę w pełni połączoną przez co nie pozwalają na analizę zbierającą wszystkie zebrane przez warstwy dla danej klasy informację.

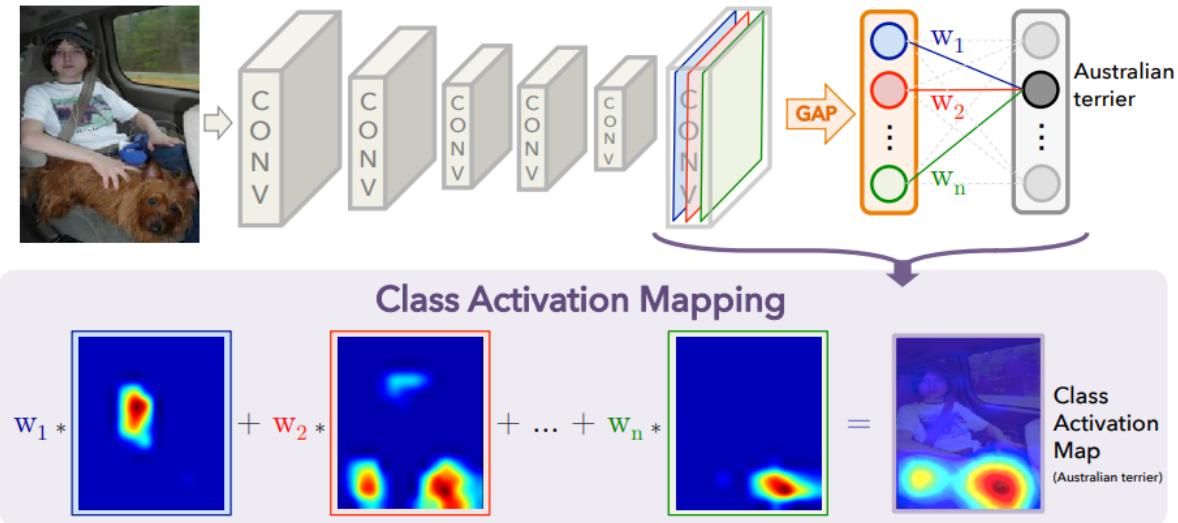


Rys. 3.7. Wynik działania algorytmu *CAM*. Mapy ciepła wskazują regiony, które miały największy wpływ na klasyfikację zdjęcia do danej klasy.

Mapa aktywacji klas dla danej kategorii wskazuje regiony zdjęcia, które miały największy wpływ podczas predykcji. Aby w pełni rozumieć działanie tej metody tworzenia obszarów istotności, należy opisać procedurę tworzenia map aktywacji klas z użyciem *GAP*, przedstawioną na rysunku 3.8.

W celu użycia metody *CAM*, sieć neuronowa musi składać się w głównej mierze z warstw konwolucyjnych i zaraz przed warstwą wyjściową dokonywać globalnego łączenia uśredniającego (*GAP*) na mapach cech, a także używać wyniku tej operacji jako wejścia do warstwy w pełni połączonej, jak przedstawiono na rysunku 3.8. Za pomocą tej struktury połączeń możemy identyfikować wpływ danych regionów zdjęć na decyzję za pomocą projekcji wstępnej warstwy wyjściowej na konwolucyjne mapy cech - tę technikę nazywamy mapowaniem aktywacji klas - *CAM*.

GAP jako wynik zwraca przestrzenną średnią mapy cech każdego neuronu z ostatniej warstwy sieci. Ważona suma tych wartości jest używana do obliczenia wyniku działania sieci. Aby obliczyć *CAM* dokonujemy podobnych obliczeń. Formalnie, w przypadku klasyfikacji zdjęć,



Rys. 3.8. Zasada działania metody CAM.

proces ten może być opisany w następujący sposób: niech $f_k(x, y)$ to wartość aktywacji k-tego neuronu warstwy wyjściowej sieci dla wartości na pozycji (x,y) w mapie cech. Wtedy dla neuronu k wynik globalnego łączenia uśredniającego to

$$F^k = \sum_{x,y} f_k(x, y) \quad (3.1)$$

Dla danej klasy c wejście do klasyfikatora $softmax$:

$$S_c = \sum_k w_k^c F_k \quad (3.2)$$

Gdzie w_k^c to waga wartości F_k dla klasy c . W takim przypadku prawdopodobieństwo przynależności wejścia do klasy c wynosi:

$$P_c = \frac{\exp(S_c)}{\sum_c \exp(S_c)} \quad (3.3)$$

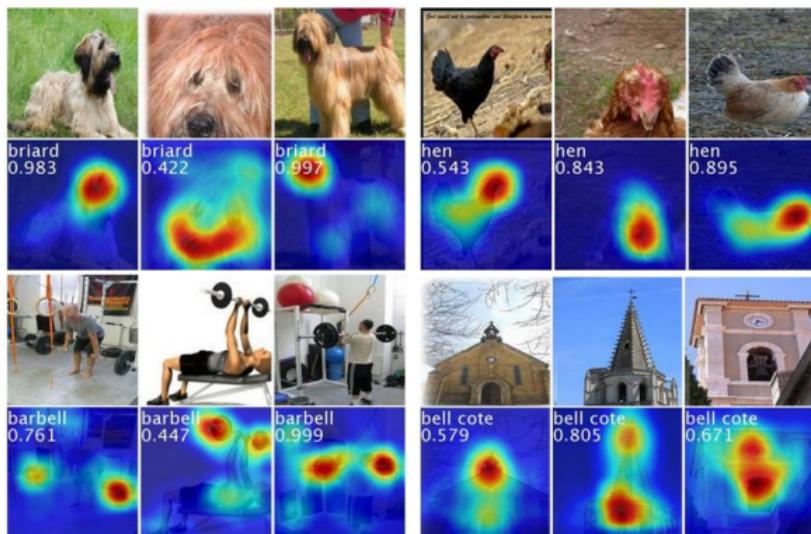
Podstawiając równanie 3.1 do równania 3.2 i zmieniając kolejność wyrażeń ze względu na liniowość operacji otrzymujemy

$$S_c = \sum_k w_k^c \sum_{x,y} f_k(x, y) = \sum_{x,y} \sum_k w_k^c f_k(x, y) \quad (3.4)$$

Korzystając z równania 3.4 definiujemy M_c mapę aktywacji klas dla klasy c , gdzie każdy element przestrzeni dany jest następującym wzorem:

$$M_c(x, y) = \sum_k w_k^c f_k(x, y) \quad (3.5)$$

Oznacza to, że $S_c = \sum_{x,y} M_c(x, y)$, co potwierdza, że mapa aktywacji klas bezpośrednio wskazuje wagę danej aktywacji w przestrzennej siatce pikseli (x,y) prowadząc do klasyfikacji obrazu do danej klasy. Podobnie jak w przypadku innych metod jednostki powinny być aktywowane przez pewne wizualne kształty, więc f_k jest mapą występowania kształtów w obrazie. Mapa aktywacji klas w takim przypadku jest ważoną liniową sumą występowania kolejnych kształtów w obrazie w różnych miejscach w przestrzeni obrazu. Oznacza to, że rozszerzając mapę aktywacji klas do rozmiarów wejściowego obrazu jesteśmy w stanie zidentyfikować, która część obrazu miała największy wpływ przy podejmowaniu decyzji, dlatego właśnie możemy powyższej metody używać jako mapy istotności w ocenie działania głębokich sieci neuronowych.



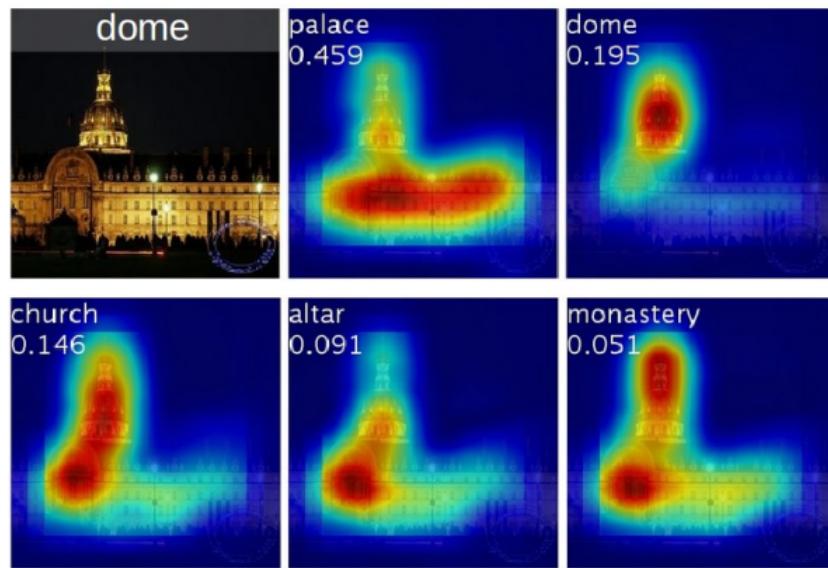
Rys. 3.9. Przykładowe wyniki działania metody CAM.

Na koniec warto dodać, że dla tego samego zdjęcia, dla różnych klas metoda generuje różne mapy istotności, ponieważ inne czynniki wpływają na te decyzje. Obrazuje to rysunek 3.10.

3.2.4. **Grad-CAM – Ramprasaath R. Selvaraju**

Jedna z najnowszych metod tworzenia map istotności została zaprezentowana w artykule pod tytułem *Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization* jest *Grad-CAM*[21] autorstwa Ramprasaath R. Selvaraju et al.. W przeciwieństwie do poprzednich metod, *Grad-CAM* możemy stosować do bardzo szerokiej gamy sieci konwolucyjnych:

1. CNN w pełni połączonych (np. sieci VGG)
2. CNN wykorzystywanych do ustrukturyzowanych wyników (np. napisy)
3. CNN używanych w zadaniach z multimodalnymi wejściami (np. wizualne odpowiadanie na pytania)



Rys. 3.10. Mapy istotności dla tego samego zdjęcia wejściowego w przypadku różnych wyników klasyfikacji

4. CNN używanych w uczeniu przez wzmacnianie (z ang. *reinforcement learning*)

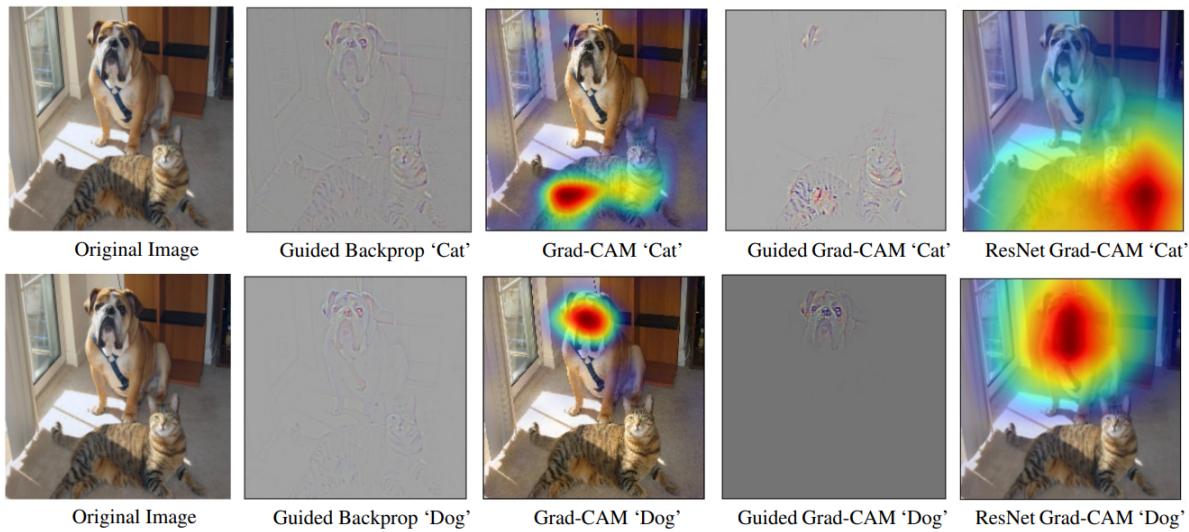
Warto zaznaczyć, że metoda *Grad-CAM* nie wymaga żadnych zmian architektonicznych ani ponownego trenowania sieci. Jest to cecha, która wyróżnia tę metodę na tle innych i jest bardzo ważna, ponieważ dzięki niej możemy uniknąć kompromisu pomiędzy interpretowalnością a dokładnością sieci (z ang. *interpretability vs. accuracy trade-off*). Za pomocą map istotności powstających w wyniku tej metody możemy zauważać, że błędne wyniki sieci, które z poziomu są nierozsądne, mają bardzo sensowne wyjaśnienia. Dodatkowo autorzy zapewniają tekstowe wyjaśnienie decyzji modelu, wskazujące które neurony skupiały się na jakich fragmentach obrazu.

Autorzy podkreślają, że jeśli zastosujemy nadzorowaną wersję metody *Grad-CAM* (z ang. *Guided Grad-CAM*), to w przeciwieństwie do innych metod, ta metoda wyznaczania map istotności będzie w stanie pogodzić dwie cechy pełnowartościowej wizualizacji:

1. rozróżnialność klas - lokalizacja kategorii na zdjęciu - cecha metody *CAM*, *Grad-CAM*
2. wysoka rozdzielczość - możliwość uchwycenia drobnych szczegółów - cecha metody nadzorowanej propagacji wstecznej oraz sieci dekonwolucyjnych

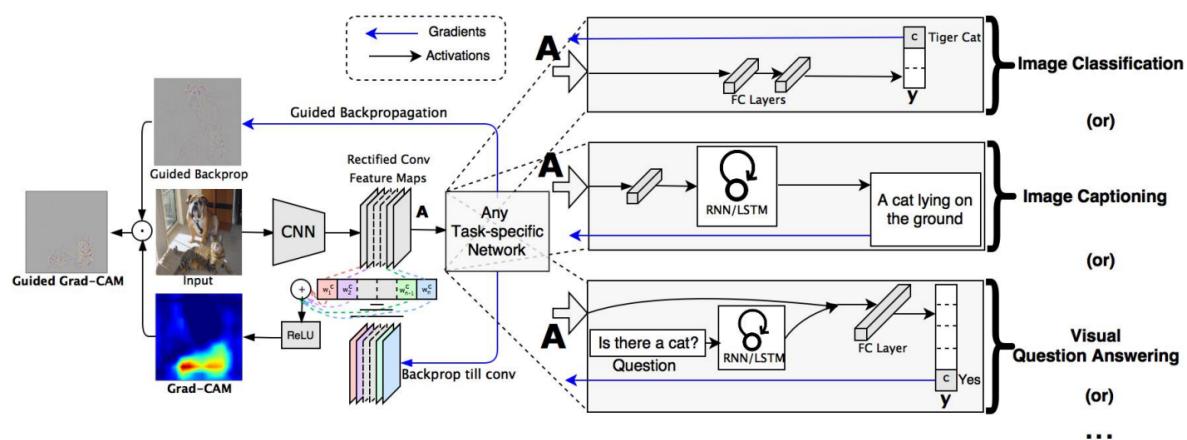
Rysunek 3.11. pokazuje, że różne metody wyznaczania map istotności przedstawiają wyniki w inny sposób. Dla obrazu wejściowego przedstawionego na tym rysunku, najlepsze wyniki zwraca *Guided Grad-CAM* - jest w stanie zaprezentować wyniki w wysokiej rozdzielczości zarazem oddzielając od siebie dwie klasy - psa i kota.

Metodą najbardziej podobną do metody opisanej w tym rozdziale jest *CAM*. Minusem metody *CAM* jest to, że jest ona możliwa do zastosowania jedynie dla konkretnie zbudowanych



Rys. 3.11. Porównanie różnych map istotności pod kątem rozróżnialności cech i rozdzielczości

sieci konwolucyjnych, które wykonują globalne łączenie uśredniające bezpośrednio przed ostatnią warstwą sieci. Takie sieci mogą mieć gorszą skuteczność dla niektórych zadań (np. klasyfikacja zdjęć), a dla niektórych zadań nawet nie można ich zastosować (np. podpisywanie zdjęć). Metoda *Grad-CAM* wykorzystuje gradient do łączenia map cech, dzięki czemu, jak już wspomniano, nie wymaga zmian w architekturze sieci, a więc może być stosowana dla wszystkich sieci CNN. Inną metodą, która wykonuje podobne zadanie są opisane przeze mnie sieci dekonwolucyjne[17]. Przewagą metody *Grad-CAM* w tym przypadku jest to, że otrzymuje ona lokalizację ważnych obszarów zdjęcia podczas jednej propagacji w przód i częściowej propagacji wstecz dla każdego zdjęcia, dzięki czemu jest o rząd wielkości bardziej wydajna.



Rys. 3.12. Schemat działania metody *Grad-CAM*

Rysunek 3.12 przedstawia schematycznie działanie metody *Grad-CAM*. Mając dane wejściowe zdjęcie oraz kategorię, która nas interesuje jako wyjście, wykonujemy propagację

wprzód przez część konwolucyjną sieci i następnie przez część sieci specyfczną dla wykonywanego zadania, aby otrzymać wynik dla kategorii. Gradienty wszystkich innych klas są ustawione na 0, a klasy która nas interesuje na 1. Następnie sygnał gradientu jest propagowany wstecz do konwolucyjnych map cech, które nas interesują i które chcemy następnie połączyć, aby obliczyć wynikową mapę istotności. Na końcu, jeśli stosujemy *Guided Grad-CAM*, punktowo przemnażamy każdą wartość mapy istotności z wynikiem nadzorowanej propagacji wstecz, aby otrzymać wizualizację, która ma wysoką rozdzielcość, ale zarazem jest specyficzna dla analizowanej klasy.

Z matematycznego punktu widzenia, w celu otrzymania mapy istotności $L_{Grad-CAM}^C$ o szerokości u i wysokości v dla klasy c , najpierw obliczamy gradient wyniku ostatniej warstwy sieci przed operacją $softmax$ y^c względem map cech A^k warstwy konwolucyjnej, czyli

$$\frac{\partial y^c}{\partial A_{ij}^k} \quad (3.6)$$

Tym sposobem otrzymujemy wielowymiarowe gradienty o rozmiarze map cech. Gradienty te płynąc z powrotem są globalnie uśredniane względem obu wymiarów (szerokości oraz wysokości, indeksowane jako i oraz j) aby obliczyć wagę neuronów α_k^c :

$$\alpha_k^c = \frac{1}{Z} \sum_i \sum_j \frac{\partial y^c}{\partial A_{ij}^k} \quad (3.7)$$

podczas propagowania wstecz α_k^c , gradientów względem aktywacji, obliczenia sprowadzają się do mnożenia macierzowego macierzy wag oraz gradientów względem aktywacji, aż do ostatniej warstwy konwolucyjnej, do której gradienty są propagowane. Więc α_k^c reprezentuje częściową linearyzacje głębokiej sieci neuronowej, poniżej punktu A, która przechwytuje ważność mapy cech k dla klasy docelowej c .

Gdy mamy już macierz ważności danych map cech (przechowującą elementy α_k^c) to dokonujemy mnożenia wag α_k^c wraz z odpowiadającymi im mapami cech A^k , sumujemy otrzymane składowe finalnej mapy istotności i na końcu wynikową macierz podajemy jako argument do funkcji *ReLU*:

$$L_{Grad-CAM}^C = ReLU\left(\sum_k \alpha_k^c A^k\right) \quad (3.8)$$

wynik tej operacji to zgrubna mapa istotności o tym samym rozmiarze co konwolucyjne mapy cech. Funkcja *ReLU* jest stosowana ponieważ interesują nas jedynie cechy, które mają pozytywny wpływ na klasę, która nas interesuje (są ważne przy podejmowaniu przez sieć decyzji). Ujemne wartości najprawdopodobniej odpowiadają za inną klasę, więc nie są dla nas ważne. Gdybyśmy nie zastosowali funkcji *ReLU*, mapy istotności czasami zaznaczałyby więcej niż klasę, która nas interesuje i gorzej działałyby przy lokalizacji obiektów.

3.2.4.1. Porównanie metody *Grad-CAM* oraz *CAM*

Ogólnie rzecz biorąc, *Grad-CAM* jest generalizacją metody *CAM* dla większej liczby architektur sieci konwolucyjnych. Metoda *CAM* tworzy mapę istotności dla sieci konwolucyjnej przeznaczonej do klasyfikacji zdjęć, jeśli przed operacją *softmax* dodamy warstwę uśredniającą globalnie (z ang. *global average pooling*).

Poniżej przedstawiam matematyczny dowód tej tezy. Założymy, że przedostatnia warstwa sieci tworzy K mapy cech, $A^k \in \mathbb{R}^{u \times v}$ z elementami indeksowymi za pomocą i oraz j . Te mapy cech są przestrzennie przekształcane za pomocą operacji uśredniania globalnego a następnie liniowo transformowane w celu obliczenia wyniku Y^c dla każdej klasy c :

$$Y^c = \sum_k w_k^c \frac{1}{Z} \sum_i \sum_j A_{ij}^k \quad (3.9)$$

w_k^c – - wagi cech klas

$\frac{1}{Z} \sum_i \sum_j$ - globalne uśrednianie

A_{ij}^k – - mapa cech

Niech F^k będzie wynikiem globalnego uśredniania:

$$F^k = \frac{1}{Z} \sum_i \sum_j A_{ij}^k \quad (3.10)$$

W metodzie *CAM* obliczamy końcowy wynik jako

$$Y^c = \sum_k w_k^c \cdot F^k \quad (3.11)$$

gdzie w_k^c jest wagą k -tej mapy cech dla klasy c . Obliczając gradient wyniku dla klasy c (Y^c) względem średniej globalnej mapy cech F^k za pomocą reguły łańcuchowej otrzymujemy:

$$\frac{\partial Y^c}{\partial F^k} = \frac{\frac{\partial Y^c}{\partial A_{ij}^k}}{\frac{\partial F^k}{\partial A_{ij}^k}} \quad (3.12)$$

Korzystając z równania 3.10 obliczamy pochodną F^k względem A_{ij}^k otrzymujemy $\frac{\partial F^k}{\partial A_{ij}^k} = \frac{1}{Z}$. Podstawiając tę wartość do równania 3.12. otrzymujemy:

$$\frac{\partial Y^c}{\partial F^k} = \frac{\partial Y^c}{\partial A_{ij}^k} \cdot Z \quad (3.13)$$

Z równania 3.11. wiemy, że $\frac{\partial Y^c}{\partial F^k} = w_k^c$. Wstawiając tę wartość do równania 3.13 otrzymujemy:

$$w_k^c = Z \cdot \frac{\partial Y^c}{\partial A_{ij}^k} \quad (3.14)$$

Sumując obie strony tego równania po wszystkich pikselach:

$$\sum_i \sum_j w_k^c = \sum_i \sum_j Z \cdot \frac{\partial Y^c}{\partial A_{ij}^k} \quad (3.15)$$

ponieważ Z oraz w_k^c nie zależą od pozycji pikseli (i, j) możemy to równanie przepisać w następujący sposób:

$$Zw_k^c = Z \sum_i \sum_j \frac{\partial Y^c}{\partial A_{ij}^k} \quad (3.16)$$

Z jest liczbą pikseli w mapach cech, możemy więc skrócić tą wartość:

$$w_k^c = \sum_i \sum_j \frac{\partial Y^c}{\partial A_{ij}^k} \quad (3.17)$$

Równanie 3.17 dowodzi tezie, że metoda *Grad-CAM* jest generalizacją metody *CAM*. Możemy zauważać, że waga α_k^c wyraża się takim samym wzorem jak waga w_k^c używana w metodzie *CAM*. Właśnie generalizacja ta pozwala nam stosować metodę *Grad-CAM* do szerszej gamy architektur sieci konwolucyjnych.

3.2.4.2. Nadzorowany *Grad-CAM*

Mimo, że wyniki uzyskane za pomocą metody *Grad-CAM* są rozróżnialne względem klas i przechowują informację o najważniejszych pod kątem podejmowania decyzji obszarów w obrazie, to szczegółowość wyniku nie jest zbyt duża (nie możemy wizualizować wyników w przestrzeni pikseli).

Łącząc wyniki uzyskane przez metodę *Grad-CAM* z wynikami uzyskanymi za pomocą metody Nadzorowanej Propagacji Wstecz poprzez mnożenie element po elemencie wyników obu metod ($L_{Grad-CAM}^c$ jest najpierw rozszerzane tak, aby miało wymiary obrazu wejściowego) możemy uzyskać wyniki, które są charakterystyczne dla danej klasy, ale także mają wysoką rozdzielcość, czyli możemy zauważać bardzo szczegółowe detale, które wpłynęły na podjęcie decyzji przez sieć. Dobrze obrazuje to rysunek 3.11. Stosując Nadzorowaną wersję metody *Grad-CAM*, w przypadku detekcji kota tygrysiego możemy zauważać charakterystyczne cechy tej rasy, takie jak prążki na całym dziele czy spiczaste uszy. Jednocześnie metoda nie zwraca uwagi na psa, który także znajduje się na zdjęciu, czego nie możemy powiedzieć o metodzie Nadzorowanej Propagacji Wstecz.

Aby uzyskać podobne wyniki można by także zastąpić wyniki Nadzorowanej Propagacji Wstecz wynikami uzyskanymi za pomocą sieci dekonwolucyjnych, jednak doświadczalnie sprawdzono, że metoda ta generuje więcej artefaktów.

4. Sposób prowadzenia testów

4.1. Środowisko programistyczne i wykorzystane biblioteki

Jako że większość projektów z dziedziny uczenia maszynowego oraz analizy danych realizowana jest za pomocą notatników *Jupyter*, ja także zdecydowałem się z nich skorzystać. Jest to spowodowane tym, że językiem programowania, z którego się w nich korzysta jest *Python*, który posiada wiele gotowych bibliotek do trenowania sieci neuronowych a także wizualizacji wyników. Dodatkowo, notatnik *Jupyter* pozwala kod języka *Python* przeplatać z komentarzami oraz wizualizacjami, dzięki czemu powstaje dobrze udokumentowana praca. Najważniejszym powodem, dla którego zdecydowałem się na wybór notatników *Jupyter* jest jednak możliwość korzystania ze środowiska *Google Colab*, które umożliwia trenowanie sieci z wykorzystaniem darmowych układów *GPU*, co w moim przypadku było bardzo użyteczne ze względu na to, że wykorzystywałem jako zbiory uczące zdjęcia. Korzystanie z układów GPU zmniejszyło czas trenowania sieci nawet o 95%, dzięki czemu mogłem przetestować znacznie większą ilość architektur.

Jak już wcześniej wspomniałem, część techniczną mojej pracy inżynierskiej zrealizowałem za pomocą języka *Python*. Korzystałem z takich bibliotek jak:

1. *NumPy* - obliczenia na macierzach;
2. *MatPlotLib*, *Cv2*, *Seaborn* - wizualizacja wyników;
3. *TensorFlow*, *Keras* - nauka modeli uczenia maszynowego, augmentacja danych;
4. *Sklearn* - podział na zbiór uczący i testowy.

4.2. Zbiór danych

Głównym celem mojej pracy było porównanie działania map istotności dla różnych architektur sieci neuronowych dla obrazów medycznych. Zdecydowałem się wybrać jeden z popularniejszych rodzajów zbiorów danych medycznych, który dostępny jest na stronie *Kaggle*[22]

i zawiera 4600 zdjęć rentgenowskich mózgów. Zdjęcia osób chorych stanowią około 55% całości. Około 98% zdjęć przechowywana jest w formacie *JPEG*, prawie 2% w formacie *TIFF*, pojedyncze zdjęcia przechowywane są w innych formatach, na przykład *PNG*.



Rys. 4.1. Przykładowe elementy zbioru danych

Wymiary zdjęć są różne, przy czym wyróżnić można dwa często występujące: (512, 512, 3) - 19% oraz (225, 225, 3) - 8% oraz inne - 73%. Warto dodać, że 97% obrazów przechowywana jest w modelu *RGB*, reszta zapisana została między innymi w modelu *L*.

Obrazy podzielone są na dwa foldery: *Brain Cancer* - zdjęcia osób chorych oraz *Healthy* - zdjęcia mózgów osób zdrowych. Do folderów dołączony został plik przechowujący ogólne informacje na temat zbioru danych, przedstawione powyżej.

4.3. Przygotowanie danych

Aby móc skorzystać z zbioru danych otrzymując wysoką precyżję wytrenowanych sieci należało poczynić kilka kroków. Pierwszym z nich było przeskalowanie zdjęć, aby wszystkie miały taki sam rozmiar. Zdecydowałem się wybrać wymiar (150, 150, 3), ponieważ jest to zdjęcie na tyle duże, aby zachować wszystkie szczegóły, a zarazem na tyle małe, aby wykonywać obliczenia w zadowalającym tempie.

Następnie należało dokonać podziału zbioru danych na zbiór treningowy i testowy. W tym celu trzeba było nadać zdjęciom etykiety. Zdjęciom mózgów osób chorych przyporządkowałem etykietę [0, 1], osobom zdrowym [1, 0]. Zdecydowałem się na jeden z bardziej typowych podziałów - do zbioru treningowego trafiło 80% wszystkich zdjęć, do zbioru testowego 20%.

Należało także zastosować augmentację danych (z ang. *data augmentation*) polegającą na sztucznym zwiększeniu liczby zdjęć wejściowych poprzez odpowiednie rotacje, zbliżenia oraz zmiany ostrości zdjęć. Dokonałem tego za pomocą gotowej klasy dostępnej w module *Keras*[23] służącej do tworzenia w czasie rzeczywistym dodatkowych elementów zbioru danych zgodnie z zadanymi parametrami - *ImageDataGenerator*. Parametry podane do tej klasy to:

1. Kąt rotacji (z ang. *rotation angle*) = 40%,
2. Zakres przesunięcia w dla osi x oraz y (z ang. *width / height shift range*) = 0.2,

3. Zakres ścinania (z ang. *shear range*) = 0.2,
4. Zakres przybliżenia (z ang. *zoom range*) = 0.2,
5. Odwracanie w poziomie (z ang. *horizontal flip*) = Prawda,
6. Sposób wypełniania (z ang. *fill mode*) = najbliższy

Ostatnim krokiem był podział przykładów uczących i testowych na paczki (z ang. *batches*) składające się z 32 zdjęć. W taki sposób uzyskałem gotowy zbiór danych i mogłem przystąpić do nauki sieci neuronowych.

4.4. Wybrane architektury i metody wyznaczania map istotności

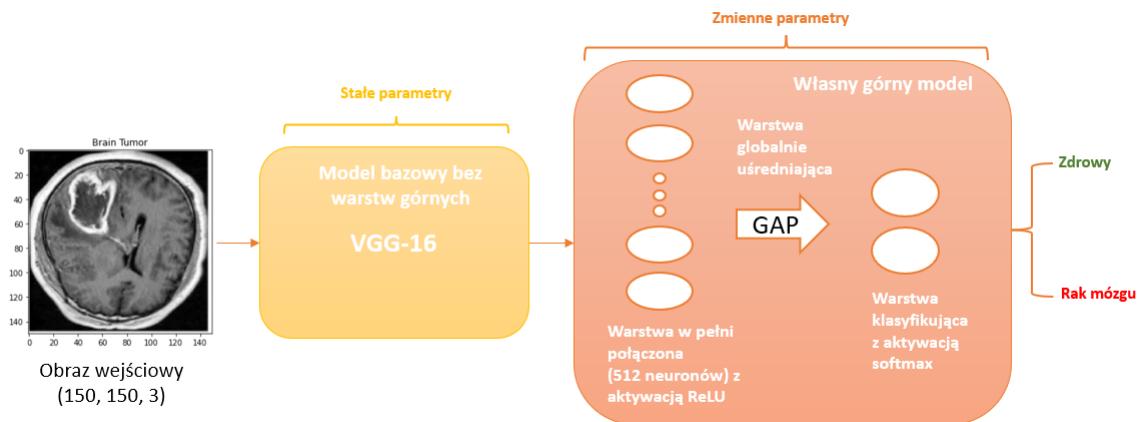
Zdecydowałem się na wybór trzech gotowych architektur sieci i zastosowanie dla nich *transfer learningu* w celu uzyskania modeli o wysokiej precyzyji dla zdefiniowanego zagadnienia. W tym celu zainportowałem gotowe implementacje sieci neuronowych z pakietu *Keras* dla wszystkich wybierając wagi wytrenowane na zbiorze danych *ImageNet*, ustawiając rozmiar wejścia sieci na (150, 150, 3) oraz pomijając górne warstwy modeli w celu zastosowania uczenia przez transfer wiedzy. Dodatkowo, porównałem wyniki dla tak otrzymanych architektur z wynikami dla dość prostej sieci konwolucyjnej, składającej się z kilku warstw.

Przed opisem konkretnych modeli sieci, warto dodać, że parametry wszystkich warstw modeli bazowych traktowałem jako stałe, ponieważ testy dla mojego zastosowania pokazały, że modele takie są w stanie osiągnąć znacznie lepszą dokładność. Eksperymenty udowodniły, że w przypadku modeli bazowych o zmiennych parametrach dla kilku ostatnich warstw sieci konwolucyjnych skuteczność po 40 epokach trenowania jest nawet o 30% gorsza. Może to wynikać z tego, że sieci te zostały wytrenowane na zbiorach danych, które zawierają miliony zdjęć dzięki czemu są w stanie odnajdować znacznie bardziej skomplikowane, abstrakcyjne wzorce. Wszystkie wytrenowane przeze mnie modele zostały zapisane w katalogu "Models".

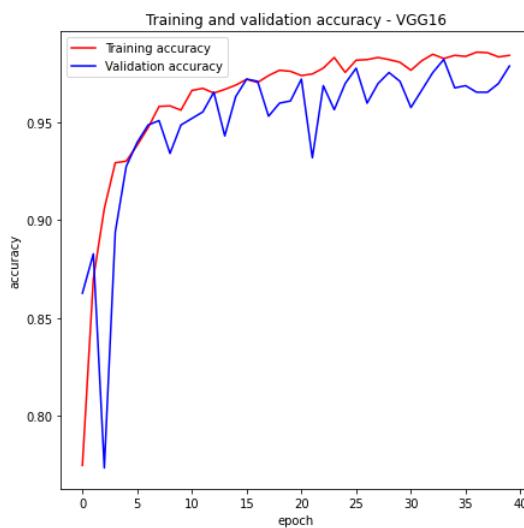
4.4.1. Model bazujący na sieci VGG-16

Pierwszą wybraną przeze mnie gotową architekturą, była sieć *VGG*, czyli najlepsza architektura dostępna przed pojawiением się sieci *ResNet*. Zdecydowałem się z niej skorzystać, ponieważ nie korzysta ona z połączeń rezydualnych, ma dość prostą budowę, oraz jest bardzo dobrze przebadana i opisana w wielu artykułach naukowych. Wcześniejszta dogłębna analiza artykułów naukowych pozwoliła mi na wybór najlepszej możliwej architektury sieci *VGG*, czyli

VGG-16. Jest to ilość warstw, która nie prowadzi do efektu zanikającego gradientu, zarazem zapewniając największą skuteczność.



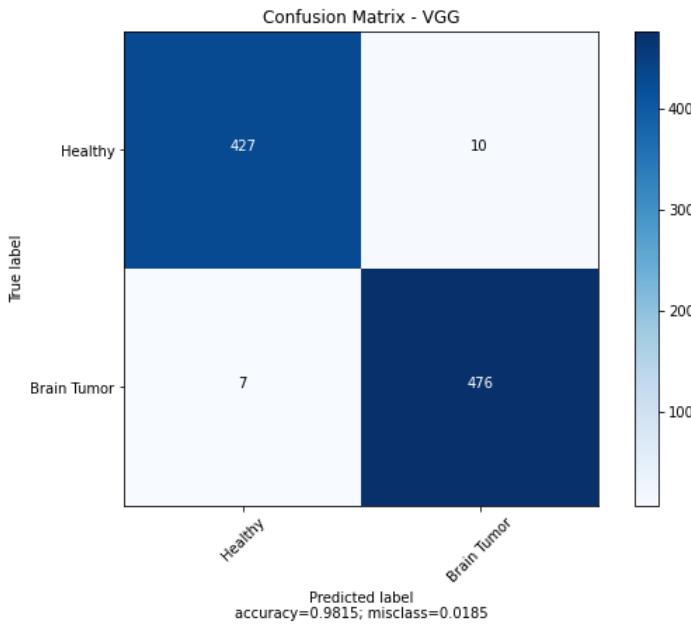
Rys. 4.2. Schemat modelu bazującego na sieci VGG-16



Rys. 4.3. Krzywa uczenia dla modelu bazującego na sieci VGG-16

Następnie najwyższe warstwy powyższej sieci zostały przystosowane do predykcji dla opisanego wcześniej zbioru danych. Dzięki temu, że wagi modelu bazowego zostały wcześniej nauczone na zbiorze danych ImageNet, operacja uczenia była znacznie mniej złożona matematycznie: tylko 263 682 z 14 978 370 parametrów było zmiennych.

Rys. 4.3 przedstawia krzywą uczenia dla modelu bazującego na sieci VGG. Można z niej wywnioskować, że wybrana liczba epok była wystarczająco duża, ponieważ wartość precyzji modelu przestaje rosnąć. Możemy także zauważyć, że nie występuje zjawisko *overfittingu*, ponieważ różnica pomiędzy dokładnością dla zbioru treningowego i testowego nie jest znaczna.



Rys. 4.4. Macierz pomyłek dla modelu bazującego na sieci VGG-16

Z macierzy pomyłek przedstawionej na rys. 4.4 możemy odczytać, że w większości przypadków ze zbioru testowego model działa bardzo dobrze. Jeśli chodzi o błędne działanie, to częściej zdarza się sytuacja, w której model uzna za chorego osobę zdrową, niż nie rozpozna raka mózgu, a więc czułość modelu jest większa niż swoistość, co jest oczywiście cechą pożądaną. Dokładność otrzymanego modelu została przedstawiona w tabeli 4.1.

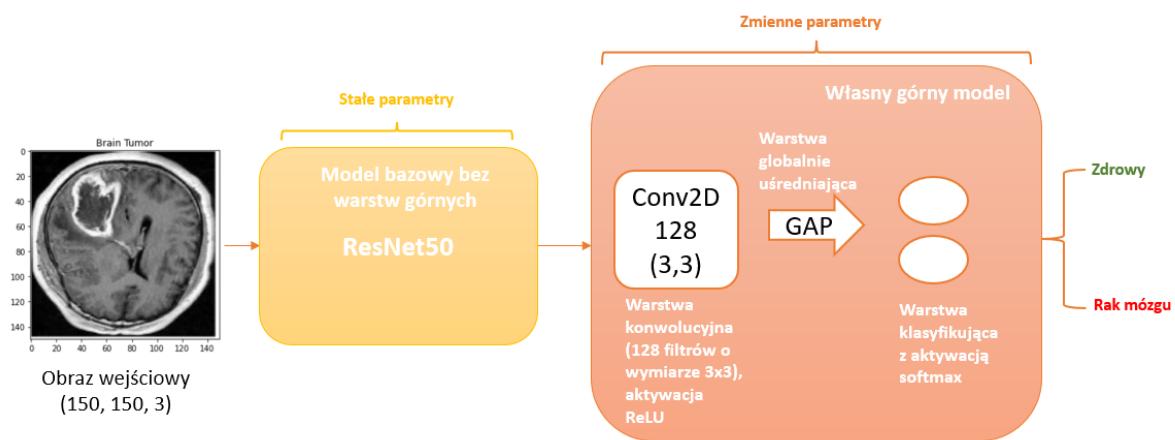
Tabela 4.1. Precyza modelu bazującego na sieci VGG-16

Zbiór treningowy	Zbiór testowy
98, 42%	97, 88%

4.4.2. Model bazujący na sieci ResNet50

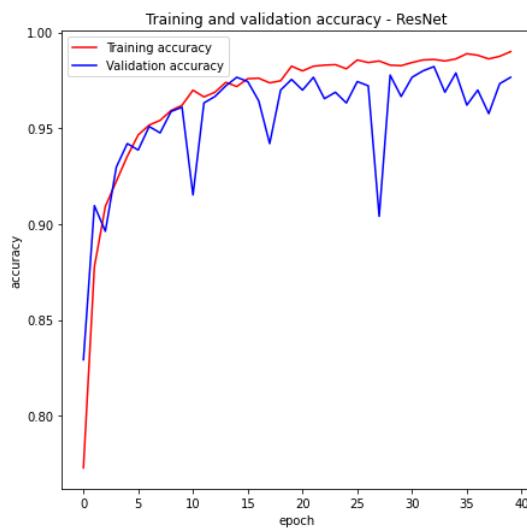
Kolejnym dokonanym przeze mnie wyborem była sieć *ResNet50*. Zdecydowałem się na wykorzystanie tej sieci, ponieważ chciałem sprawdzić czy metody wyznaczania map istotności działają poprawnie w przypadku zastosowania połączeń rezydualnych w architekturze sieci. Kolejnym powodem, dla którego wykorzystałem tę sieć jest jej znaczenie dla rozwoju głębokich sieci neuronowych w przypadku analizy obrazów.

Wybrałem sieć, która składa się z 50 warstw o zmiennych parametrach, ponieważ jest ona dobrym kompromisem pomiędzy złożoną, składającą się z wielu parametrów siecią 152 warstwową a sieciami składającymi się z 18 czy 34 warstw, dla których nie obserwujemy wystarczająco znacznego wzrostu skuteczności w stosunku do sieci VGG-16.



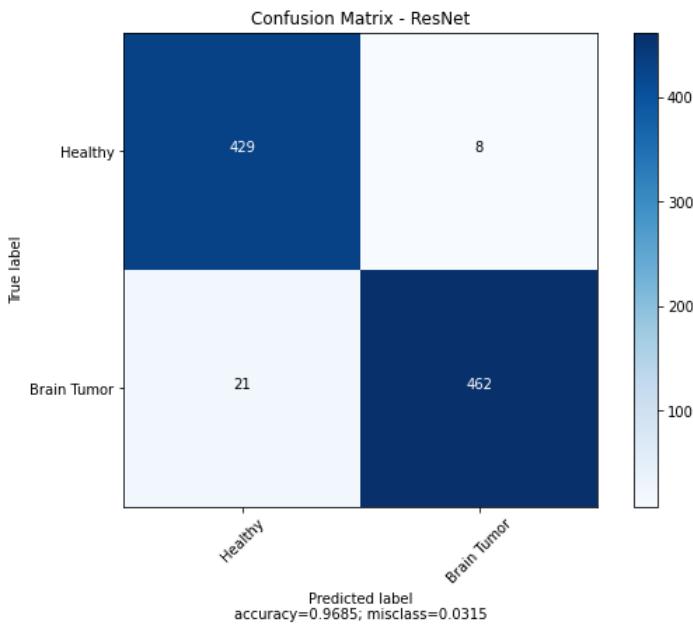
Rys. 4.5. Schemat modelu bazującego na sieci *ResNet50*

W przypadku tego modelu, zdecydowałem się na dołożenie własnej, dodatkowej warstwy konwolucyjnej składającej się ze 128 filtrów o wymiarach 3x3, ponieważ chciałem sprawdzić jak wytrenowanie własnych parametrów warstwy konwolucyjnej wpłynie na sposób podejmowania przez sieć decyzji. Następnie zastosowałem *GAP* - globalne łączenie uśredniające - po którym dodałem warstwę klasyfikującą z aktywacją *softmax*. Wynikało to z faktu, że jedną z metod badania istotności, którą chciałem zastosować był algorytm *CAM*, który wymaga, aby dwie ostatnie warstwy sieci były właśnie takie.



Rys. 4.6. Krzywa uczenia dla modelu bazującego na sieci *ResNet50*

Z macierzy pomyłek przedstawionej na rys. 4.7 możemy wywnioskować, że w przeciwieństwie do sieci *VGG-16*, w przypadku sieci bazującej na modelu *ResNet50*, czułość jest na niższym poziomie niż swoistość. Jest to mniej oczekiwany wynik, ponieważ w przypadku chorób



Rys. 4.7. Macierz pomyłek dla modelu bazującego na sieci *ResNet50*

bardzo duże znaczenie ma czułość. Generalnie chcemy unikać sytuacji, w której predykcja modelu dla osoby chorej jest błędna.

Tabela 4.2. Precyza modelu bazującego na sieci *ResNet50*

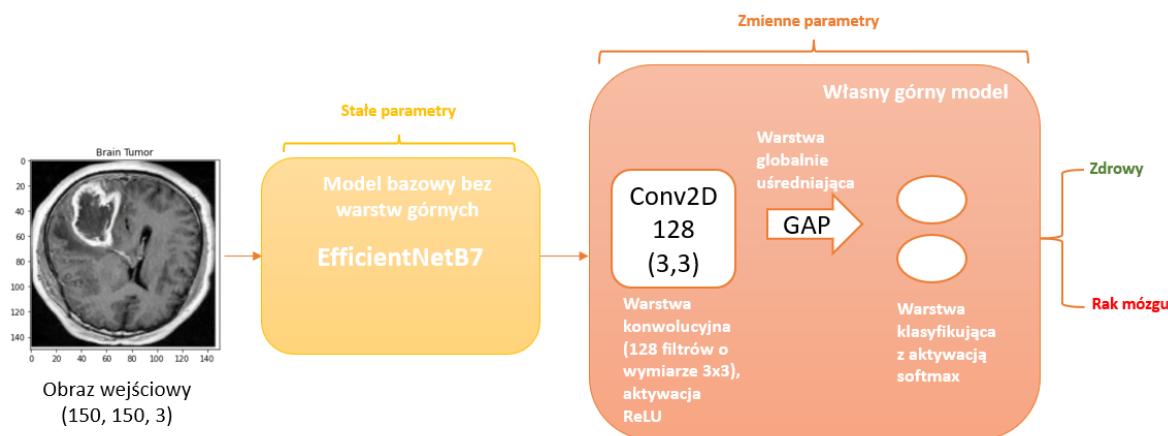
Zbiór treningowy	Zbiór testowy
98, 99%	97, 66%

4.4.3. Model bazujący na sieci *EfficientNet*

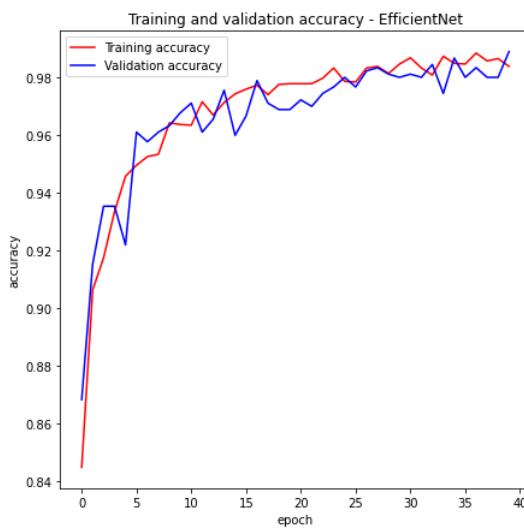
Ostatnim gotowym rozwiążaniem, które wybrałem była sieć *EfficientNet*. Przy wyborze kierowałem się głównie tym, że jest to jak najbardziej aktualna sieć osiągająca bardzo dobre wyniki w przypadku klasyfikacji zdjęć.

Modele sieci *EfficientNet* w zależności od ilości parametrów zostały oznaczone od *B0* do *B7*. W przypadku tego modelu zdecydowałem się na wybór największej możliwej architektury, ponieważ ma ona podobną ilość parametrów do sieci *ResNet152*, co pozwala mi na przetestowanie działania metod wyznaczania obszarów istotności dla modelu o większej złożoności.

Z rys. 4.9 możemy wywnioskować, że zastosowana liczba epok jest wystarczająca do osiągnięcia wysokiej precyzyji, ponieważ szybkość wzrostu dokładności znacznie zmalała pod koniec procesu uczenia. Widać zarazem, że nie obserwujemy zjawiska przetrenowania.



Rys. 4.8. Schemat modelu bazującego na sieci *EfficientNetB7*

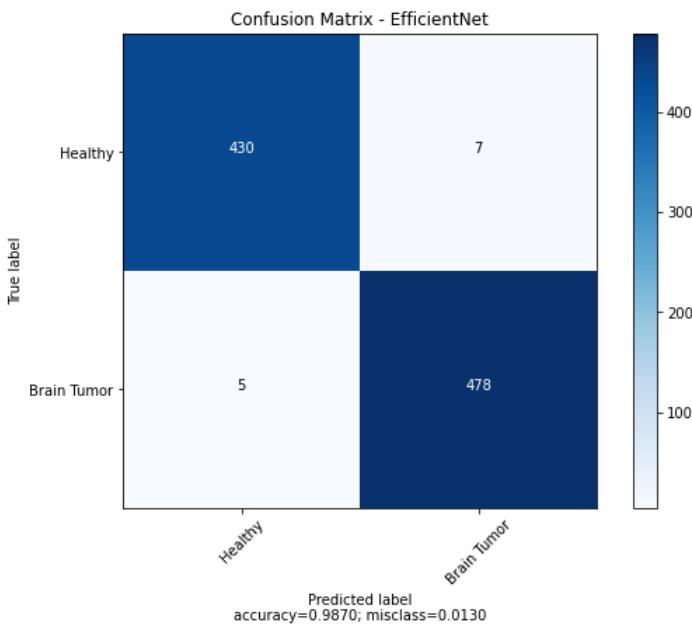


Rys. 4.9. Krzywa uczenia dla modelu bazującego na sieci *EfficientNetB7*

Kolejnym ważnym wnioskiem, który jest prawdziwy dla wszystkich modeli, dla których wykorzystujemy uczenie przez transfer wiedzy jest fakt, że już przy pierwszej epoce skuteczność jest względnie wysoka (powyżej 80%) jednak w każdym przypadku douczanie do konkretnego problemu, jakim jest detekcja guza mózgu, dodatkowo poprawia działanie sieci.

Bazując na rys. 4.3 możemy stwierdzić, że w przypadku modelu bazującego na sieci *EfficientNetB7* otrzymaliśmy najlepszą swoistość i czułość - dla obrazów testowych sieć popełniła jedynie 12 błędów.

Dla tego modelu otrzymaliśmy dokładność dla zbioru testowego lepszą niż w przypadku sieci bazującej na *VGG-16* oraz *ResNet50*. Jest to oczekiwany rezultat, ponieważ sieć *EfficientNetB7* składa się ze znacznie większej liczby parametrów, niż sieć *ResNet50*.



Rys. 4.10. Macierz pomyłek dla modelu bazującego na sieci *EfficientNetB7*

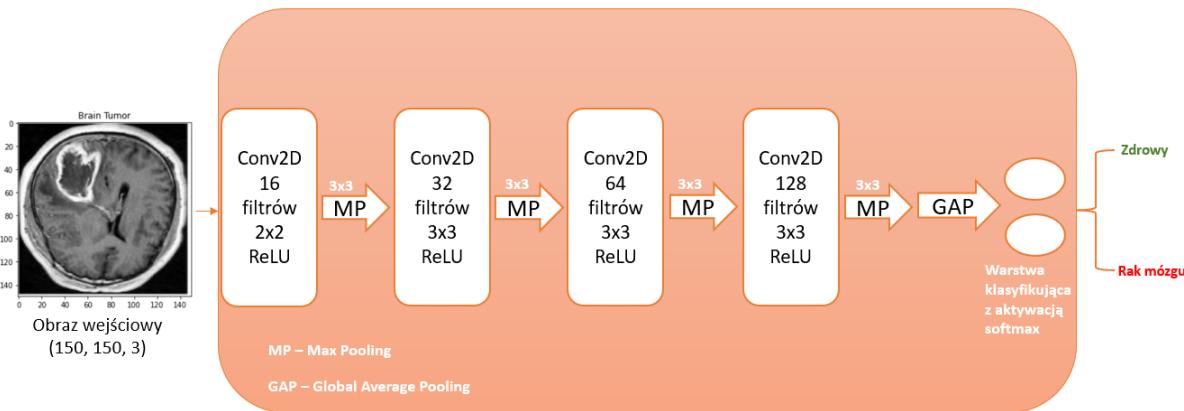
Tabela 4.3. Skuteczność modelu bazującego na sieci *EfficientNetB7*

Zbiór treningowy	Zbiór testowy
98, 37%	98, 88%

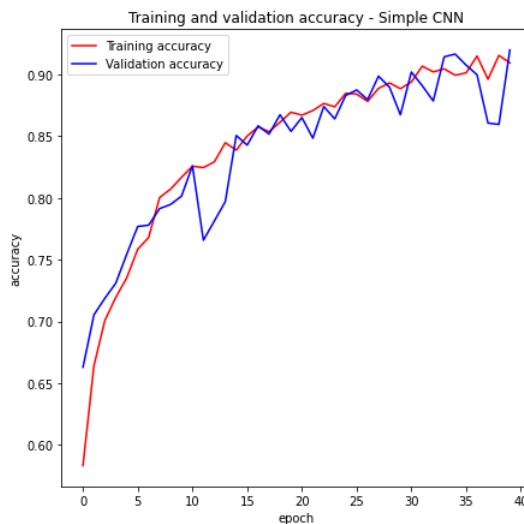
4.4.4. Prosta sieć konwolucyjna

Ostatni model, który zdecydowałem się wytrenować nie opiera się na uczeniu przez transfer. Jest to prosta sieć konwolucyjna, która składa się z czterech warstw konwolucyjnych przeplatanych operacjami *max pooling*. Pierwsza z warstw konwolucyjnych ma filtry o rozmiarze 2x2, wszystkie następne warstwy mają filtry o wymiarze 3x3. Dodatkowo wraz ze wzrostem głębości sieci rośnie liczba filtrów, co ma umożliwić wychwycenie większej liczby cech abstrakcyjnych i jest zgodne z ogólną teorią dotyczącą tworzenia sieci konwolucyjnych. Po wszystkich wymienionych warstwach występuje operacja globalnego łączenia uśredniającego i warstwa klasyfikująca z aktywacją *softmax*, prezentuje to rys. 4.11.

Zdecydowałem się przeprowadzić testy dla takiego prostego modelu, ponieważ chciałem sprawdzić czy przyczyny podejmowania przez niego decyzji są podobnie jakościowe do modeli bardziej złożonych. Obserwując krzywą uczenia zamieszczoną na rys. 4.12, możemy stwierdzić, że gdybym do uczenia zastosował więcej epok, to wynik prawdopodobnie nieznacznie by się poprawił, jednak jednym z moich założeń było zastosowanie takiej samej liczby epok uczenia dla wszystkich przygotowanych architektur.



Rys. 4.11. Schemat prostej sieci CNN



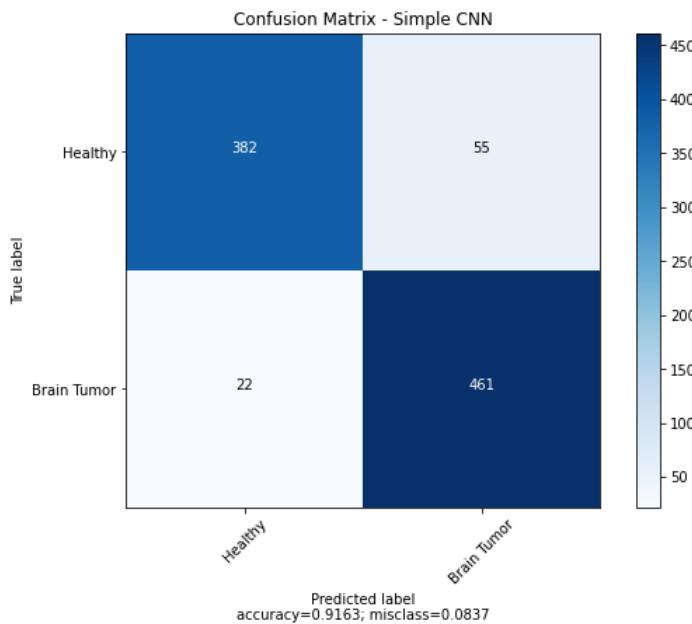
Rys. 4.12. Krzywa uczenia dla prostej sieci CNN

Macierz pomyłek przedstawiona na rys. 4.13 wskazuje, że sieć ta robi znacznie więcej pomyłek w porównaniu do poprzednich modeli (łącznie 78 dla zbioru testowego), oraz że ilość błędów w przypadku osób chorych jest ponad dwa razy mniejsza.

Pomimo tego, że zastosowany model jest znacznie prostszy niż jego poprzednicy, osiąga on dość dobrą skuteczność dla obu zbiorów: treningowego i testowego. Skuteczność ta jest przedstawiona w tab. 4.4.

Tabela 4.4. Skuteczność dla prostej sieci CNN

Zbior treningowy	Zbior testowy
90, 92%	91, 96%



Rys. 4.13. Macierz pomyłek dla prostej sieci CNN

4.4.5. Porównanie wybranych architektur

Zdecydowałem się także przygotować porównanie wytrenowanych przeze mnie sieci CNN. Najważniejsze cechy każdej sieci zawarłem w tab. 4.5. Uwypukla ona różnice pomiędzy poszczególnymi architekturami, dając do zrozumienia jak bardzo dane modele różnią się od siebie.

Tabela 4.5. Porównanie wytrenowanych architektur sieci konwolucyjnych

	Layers number	Total parameters	Trainable parameters	Train accuracy	Test accuracy	Recall	Precision
VGG16	16 + 3	14 978 370	263 682	98,42%	97,88%	98,55%	97,71%
ResNet50	50 + 3	25 947 394	2 359 682	98,99%	97,66%	95,65%	98,17%
EfficientNetB7	813 + 3	67 047 193	2 949 506	98,37%	98,88%	98,96%	98,40%
CNN	10	97 458	97 458	90,92%	91,96%	95,45%	87,41%

Pierwszą obserwacją jest fakt, że liczba warstw znacznie różni się dla danych architektur - od 10 warstw w przypadku prostej sieci CNN do 816 warstw w przypadku sieci bazującej na modelu EfficientNetB7. Warto zauważyć jednak, że pomimo, że sieć bazującą na EfficientNetB7 ma ponad 15 razy więcej warstw od sieci bazującej na ResNet50 to dzięki skalowaniu zastosowanemu w pierwszej z wymienionych architektur, ma ona około tylko 2,5 razy więcej parametrów. Jeśli rozważymy liczbę parametrów, które zmieniają się pod wpływem treningu, to różnica jest także kolosalna - najwięcej ma ich sieć EfficientNetB7 - 2 949 506, a najmniej prosta sieć CNN - 97 458 (różnica ponad 30-krotna).

Najgorszą skuteczność ma prosta sieć CNN, co przekłada się także na dwa bardzo ważne, szczególnie w przypadku zastosowań medycznych, parametry - czułość i swoistość. Informują

nas one kolejno o tym jaki procent osób chorych i zdrowych zostanie poprawnie zdiagnozowanych. Możemy zauważyć, że wśród wytrenowanych modeli istnieją takie, które mają większą swoistość, ale także takie, które mają lepszą czułość. W przypadku zastosowań medycznych lepszym wyborem byłby oczywiście model, który ma większą czułość, ponieważ lepszą sytuacją jest określenie osoby zdrowej jako chora, niż zbagatelizowanie czyjejś choroby.

4.5. Metody analizy obszarów istotności

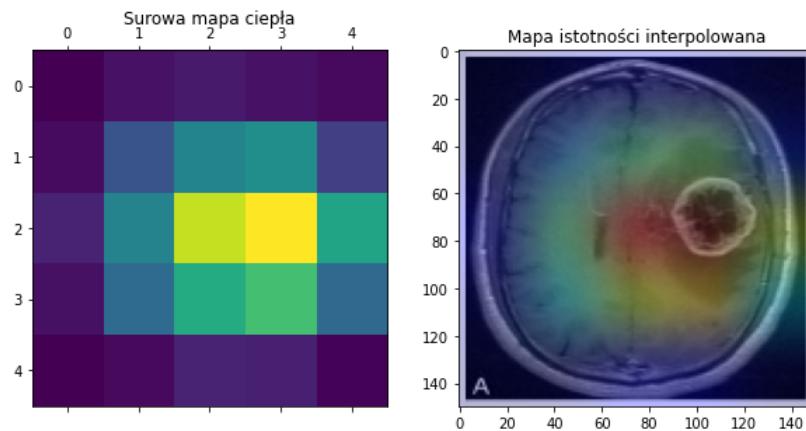
W celu analizy działania metod wyznaczania obszarów istotności opisanych w rozdziale 3 należało je zaimplementować. W tym celu wzorując się następującymi implementacjami[24][25][26] stworzyłem bibliotekę zawierającą implementację czterech metod wyznaczania obszarów istotności:

1. Nadzorowana Propagacja Wstecz (z ang. *Guided Backpropagation*)
2. Mapowanie Aktywacji Klas (z ang. *Class Activation Mapping, CAM*)
3. *Grad-CAM*
4. Nadzorowany Grad-CAM (z ang. *Guided Grad-CAM*)

Zdecydowałem się na przeprowadzenie testów dla właśnie tych czterech metod, ponieważ uznałem, że metoda korzystająca z sieci dekonwolucyjnych zwraca bardzo podobne rezultaty do metody Nadzorowanej Propagacji Wstecz, jednak generuje więcej artefaktów a wyjściowe obszary istotności są mniej ostre. Przedstawiłem jednak teorię jej dotyczącą, ponieważ to właśnie na jej podstawie powstawały kolejne metody, dała ona nowe spojrzenie na temat interpretowalności działania konwolucyjnych sieci neuronowych.

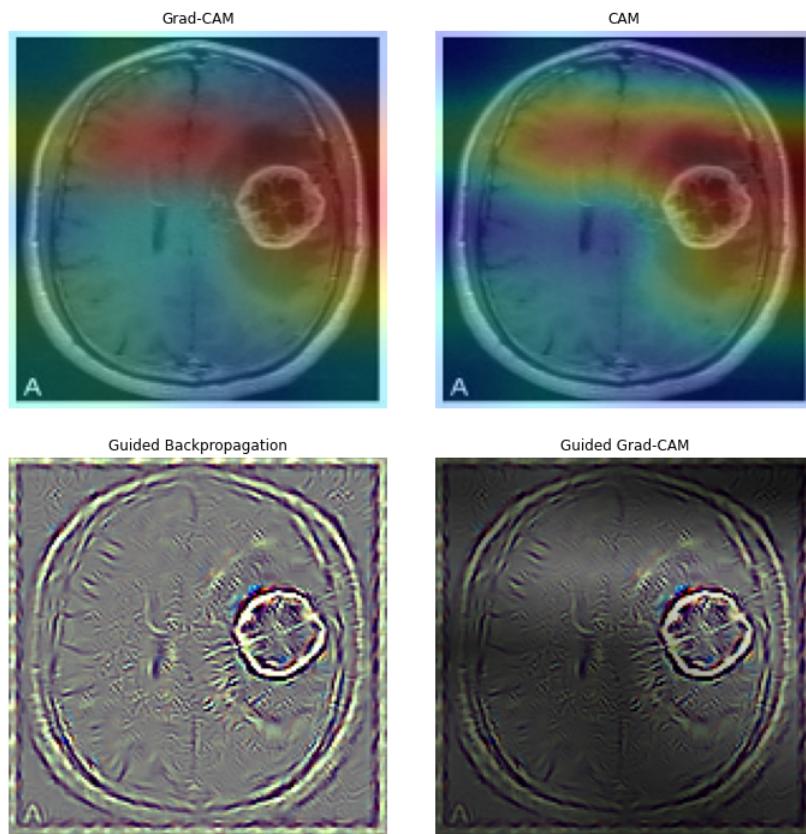
W przypadku metod *CAM* oraz *Grad-CAM* rozdzielcość otrzymanej mapy istotności zależy od rozmiaru filtrów w warstwie, dla której ją wyznaczamy. Oznacza to, że jeśli wyznaczamy mapy istotności dla najwyższej położonych warstw, to im więcej operacji konwolucji i *poolingu* wykonujemy po drodze, tym mniejszą rozdzielcość wynikowej mapy otrzymamy. Jeśli chcemy następnie otrzymać wizualizację o wymiarach równych wymiarowi obrazu wejściowego, należy taką mapę ciepła interpolować, co przedstawiłem na rys. 4.14.

Rys. 4.15 przedstawia wynikowe mapy istotności uzyskane za pomocą wszystkich analizowanych metod ich wyznaczania. Możemy zauważyć, że wyniki działania metody *Grad-CAM* oraz *CAM* są bardzo podobne, a różnice pomiędzy nimi wynikają ze sposobu wyznaczania wag, przez które mnożone są wszystkie filtry z analizowanej warstwy sieci konwolucyjnej. Metoda Nadzorowanej Propagacji Wstecz zwraca wynik, który sugeruje, że sieć skupiała się głównie



Rys. 4.14. Przykład interpolacji wynikowej mapy ciepła dla sieci *EfficientNet*

na guzie, ale także na obwodzie mózgu. Najlepszy wynik zwraca połączenie działania metody *Grad-CAM* oraz *Guided Backpropagation* poprzez mnożenie piksel po pikselu. Dzieje się tak dzięki rozróżnialności klas, którą oferuje metoda *Grad-CAM* oraz wysokiej rozdzielczości wyników metody *Guided Backpropagation*. Uwypukla to skoncentrowanie modelu na guzie mózgu, a nie na jego obwodzie.



Rys. 4.15. Przykładowe mapy istotności uzyskane za pomocą wszystkich analizowanych metod - sieć *ResNet50*

4.6. Metodologia testów

W celu opisania jak największej liczby różnic pomiędzy metodami wyznaczania obszarów istotności zastosowałem różne podejścia testowe. Pierwszym z nich były testy wizualne. Dla skrajnych przypadków obrazów wejściowych wyznaczałem obszary istotności za pomocą różnych metryk i porównywałem otrzymane wyniki. Kolejnym krokiem było zastosowanie różnych metryk w celu porównania wyników. Bardziej szczegółowy opis testów manualnych a także użytych metryk w testach statystycznych znajduje się w rozdziałach 4.6.1 oraz 4.6.2. Wyniki przeprowadzonych testów umieściłem w rozdziale 5.

4.6.1. Testy manualne

Pierwszym krokiem w celu porównania obszarów istotności były testy manualne. Polegały one na wyznaczeniu obszarów istotności dla różnych typów obrazów wejściowych:

1. przekrojowych mózgu
2. obejmujących szyję oraz czaszkę
3. obejmujących oczodoły
4. takich, gdzie guz jest koloru białego, czarnego
5. osób zdrowych...

Testy manualne były nieuniknione ze względu na to, że w przypadku metod statystycznych wiele informacji zostaje utraconych poprzez uśrednianie i nie zwracanie uwagi na typ zdjęcia wejściowego. Nie jest także możliwe porównanie wyników metod *CAM* oraz *Grad-CAM* z wynikami Nadzorowanej Propagacji Wstecz, ponieważ mają one różny charakter, co uniemożliwia zestawienie ze sobą środków ciężkości oraz parametru *IoU*. Wyniki Nadzorowanej Propagacji Wstecz w przypadku zdjęć mózgu mają zazwyczaj środek ciężkości położony blisko centrum obrazu ze względu na zwracanie uwagi przez modele na obwód mózgu a ich obszary nie są spójne, co zmniejsza skuteczność metryki *IoU*.

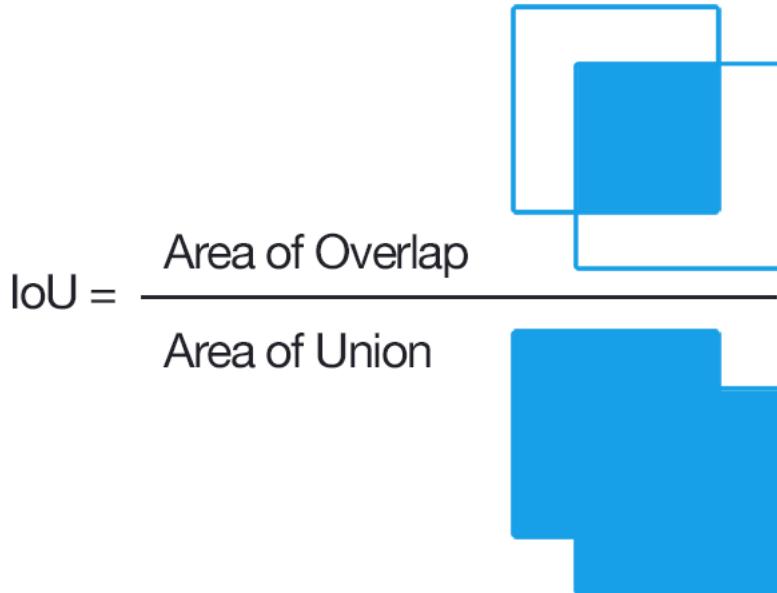
Porównywałem wyniki działania różnych metod, ale także różnych architektur sieci neuronowych. Wyniki tych testów, czyli cechy obrazów wejściowych, które wpływają na bardzo złą lub poprawną detekcję, zostały opisane w rozdziale 5.1.

4.6.2. Porównanie statystyczne

Statystyczne porównanie map istotności uzyskanych za pomocą różnych metod polegało na zastosowaniu różnych metryk i zwizualizowaniu otrzymanych wyników. Zastosowane metryki

opisałem w następnych podrozdziałach. Wyniki wraz z wizualizacjami zostały przedstawione w rozdziale 5.2.

4.6.2.1. Intersection Over Union



Rys. 4.16. Intersection over Union - zastosowanie w przypadku porównania lokalizacji obiektów za obrazach[27]

Intersection over Union jest to metryka inaczej nazywana Indeksem Jaccarda, która służy do oceny pokrycia dwóch zbiorów. Wyznacza ona podobieństwo pomiędzy dwoma zbiorami jako iloraz części wspólnej oraz sumy tych zbiorów:

$$IoU = \frac{A \cap B}{A \cup B} \quad (4.1)$$

Metryka ta została użyta do porównania pokrewieństwa obszarów istotności wyznaczonych za pomocą różnych metod. Aby ograniczyć się do obszaru, na którym głównie skupiała się sieć neuronowa, najpierw należało dokonać binaryzacji. Zastosowałem metodę *Otsu*, która pozwala na automatyczne wyznaczanie progu binaryzacji. Stały próg binaryzacji w moim przypadku nie działałby poprawnie, ponieważ zależy on od zastosowanej architektury sieci (niektóre z nich zwracają mocniejszą aktywację neuronów) a także od wejściowego obrazu (guz mózgu czasami jest bardziej widoczny). Zbinaryzowane mapy istotności porównałem za pomocą metryki *IoU*, a następnie obliczyłem także średnie pokrycie dla wszystkich obrazów testowych.

4.6.2.2. Środek ciężkości

Kolejną statystyczną metodą porównania działania map istotności było porównanie ich geometrycznych środków ciężkości. W tym celu dla każdej mapy istotności wyznaczyłem środek ciężkości za pomocą wzorów 4.2.

$$\begin{cases} X_{cm} = \frac{\sum_{i=1}^W \sum_{j=1}^H x_{ij} \cdot p_{ij}}{W \cdot H} \\ Y_{cm} = \frac{\sum_{i=1}^W \sum_{j=1}^H y_{ij} \cdot p_{ij}}{W \cdot H} \end{cases} \quad (4.2)$$

X_{cm}, Y_{cm} – współrzędna x oraz y środka ciężkości

x, y – współrzędne pikseli w obrazie

H, W – wysokość i szerokość obrazu

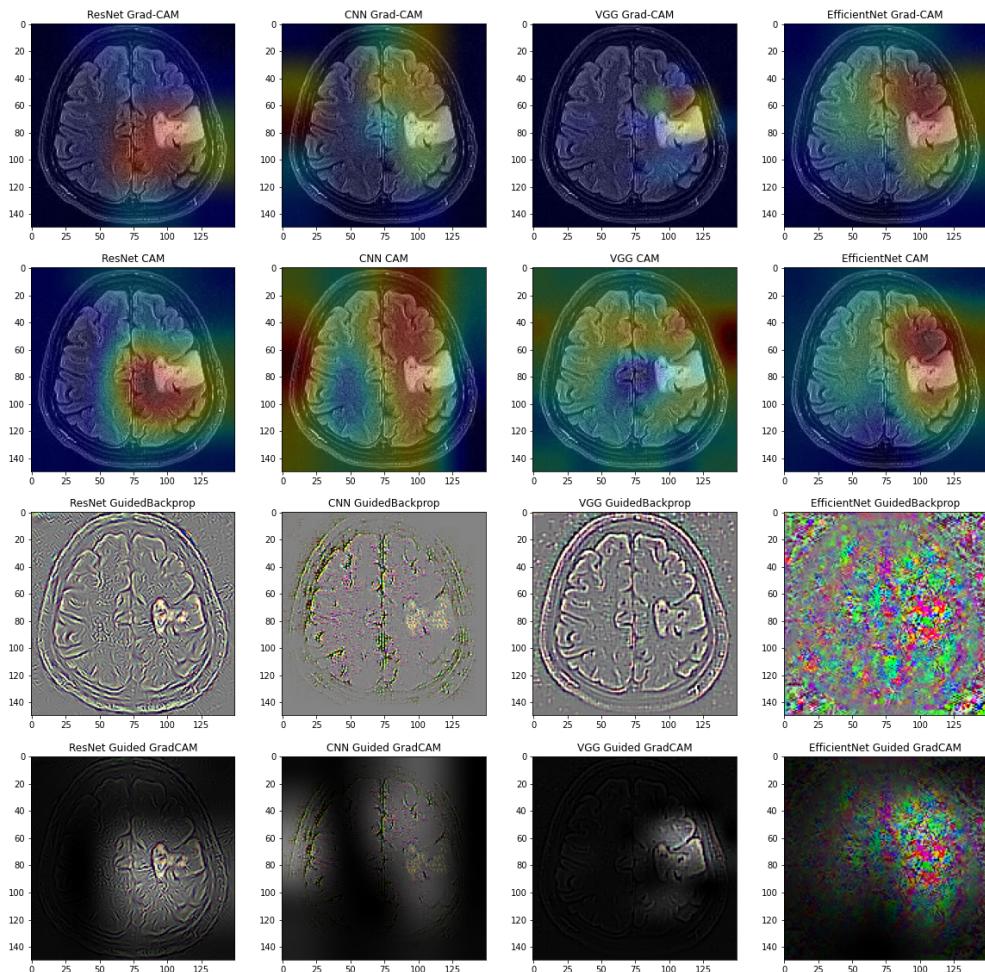
p_{ij} – wartość piksela na pozycji i, j

Zdecydowałem się obliczyć środki ciężkości najpierw dla wszystkich obrazów wejściowych ze zbioru testowego, a następnie jedynie dla zdjęć, które zostały oznaczone jako *Brain Tumor*, aby sprawdzić, czy metody są bardziej zbieżne w przypadku zdjęć zawierających guza. Porównałem środki ciężkości wyznaczone metodą *Grad-CAM* oraz *CAM* dla wszystkich użytych architektur sieci, obliczając średnią odległość kartezjańską. Następnie zwizualizowałem je za pomocą macierzy podobnej do macierzy pomyłek. Warto zaznaczyć, że zdecydowałem się przeskalać wyniki różnicy położen środków ciężkości do zakresu $<0, 1>$ (dzielenie przez maksymalną możliwą różnicę położen środków ciężkości - długość przekątnej) aby nie były one zależne od wielkości mapy ciepła, dla której były wyznaczane.

5. Przedstawienie wyników

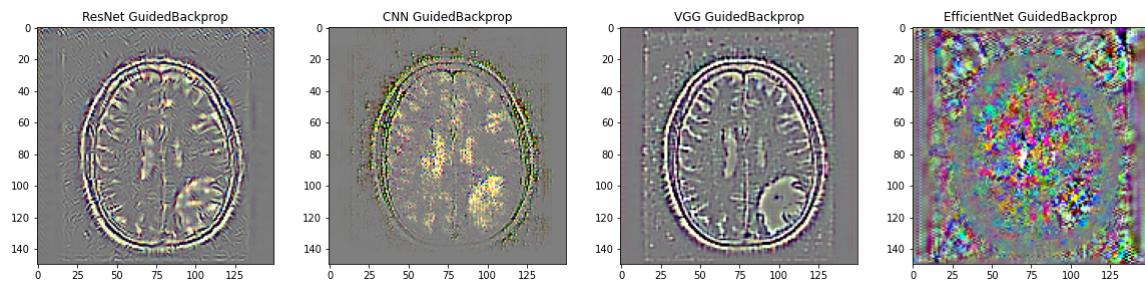
5.1. Wyniki testów manualnych

Podczas testów manualnych analizowałem wyniki działania metod wyznaczania obszarów istotności za pomocą wizualizacji dla całego zbioru testowego. Przykładowy analizowany obraz został przedstawiony na rys. 5.1.



Rys. 5.1. Przykładowa reprezentacja wyników testów manualnych dla wybranego zdjęcia wejściowego ze zbioru testowego

W przykładowej prezentacji wyników przedstawionej na rys. 5.1 wierszami zostały przedstawione wyniki kolejnych metod: *Grad-CAM*, *CAM*, Nadzorowanej Propagacji Wstecz, *Guided Grad-CAM*. Kolejne kolumny zostały poświęcone wybranym architekturom sieci: *ResNet*, prosta sieć *CNN*, *VGG* oraz *EfficientNet*. Podczas prowadzonych testów manualnych zostało wyprodukowanych 920 takich wizualizacji. Następnie, przeglądając je, zanotowałem wnioski, które zaprezentuje na przestrzeni tego rozdziału.



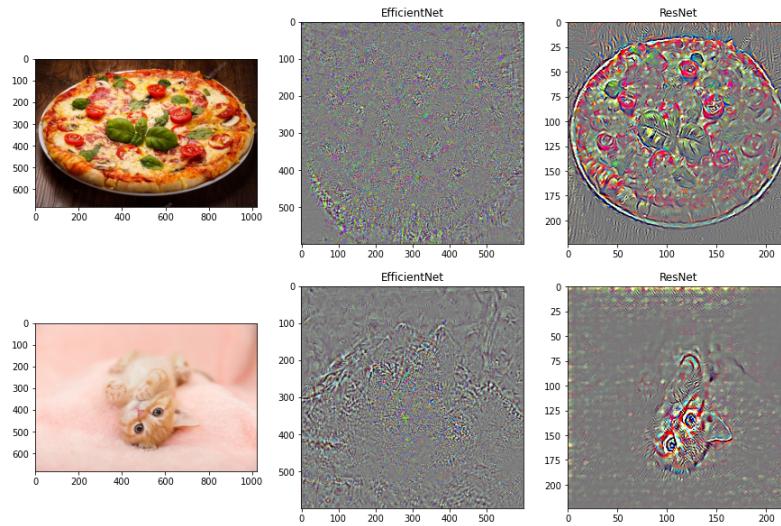
Rys. 5.2. Działanie metody Nadzorowanej Propagacji Wstecz dla przykładowego obrazu wejściowego

Analizując rys. 5.2 możemy zauważyc, że metoda działa poprawnie dla sieci *ResNet*, *VGG* oraz dla prostej sieci *CNN*, przy czym w przypadku prostej sieci *CNN* wynik jest mniej złożony niż dla dwóch pierwszych wymienionych sieci. Wynika to z mniejszej ilości warstw konwolucyjnych skutkujących mniej abstrakcyjnymi, złożonymi cechami. Wynik dla prostej sieci *CNN* jest podobny do wyników Nadzorowanej Propagacji Wstecz dla trzeciej oraz czwartej warstwy konwolucyjnej sieci *ResNet*.

Rzucającą się w oczy obserwacją jest wadliwe działanie metody Nadzorowanej Propagacji Wstecz, a co za tym idzie, metody *Guided Grad-CAM* dla sieci *EfficientNet*. Otrzymany wynik jest bardzo chaotyczny dla każdego zdjęcia wejściowego ze zbioru testowego. Dla tej sieci zdecydowałem się zwizualizować wyniki działania Nadzorowanej Propagacji Wstecz dla wszystkich dostępnych warstw konwolucyjnych. Okazało się, że wynik jest niesatysfakcjonujący dla każdej z nich. W przypadku innych architektur, każda wyższa warstwa konwolucyjna zwraca bardziej ostry oraz złożony, przypominający obraz wejściowy wynik. Pierwsze warstwy prezentują krawędzie, następne bardziej złożone ovalne kształty, które dla ostatnich warstw składają się w pełnowartościową wizualizację. W przypadku sieci *EfficientNet* pierwsze warstwy sieci zwracają wyniki, z których można cokolwiek wywnioskować, jednak wszystkie kolejne są nieinterpretowalne.

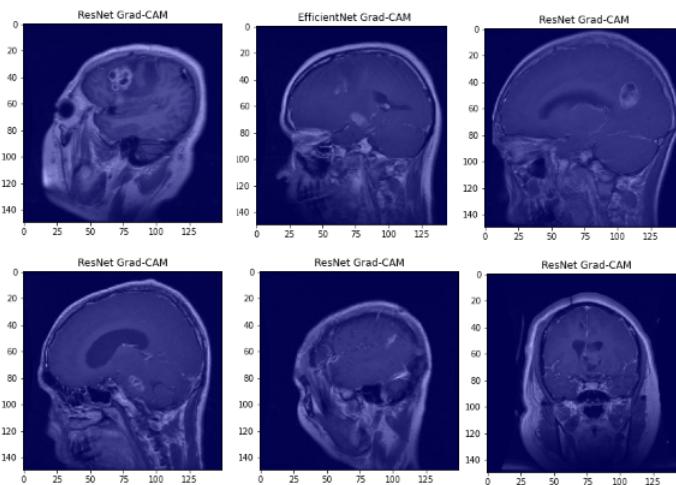
Zdecydowałem się dodatkowo porównać wyniki działania metody Nadzorowanej Propagacji Wstecz dla sieci *ResNet* oraz *EfficientNet* (bez zastosowania uczenia przez transfer wiedzy, o wszystkich warstwach przystosowanych do klasyfikowania zbioru *ImageNet*) dla dwóch obrazów przynależących do dostępnych w tym zbiorze kategorii. Okazało się, że wizualizacja

dla sieci *EfficientNet* jest znacznie gorsza, co potwierdza niepoprawne działanie metody dla tej sieci.



Rys. 5.3. Porównanie działania metody Nadzorowanej Propagacji Wstecz dla sieci *ResNet* oraz *EfficientNet* w pełni przystosowanych do zbioru *ImageNet*

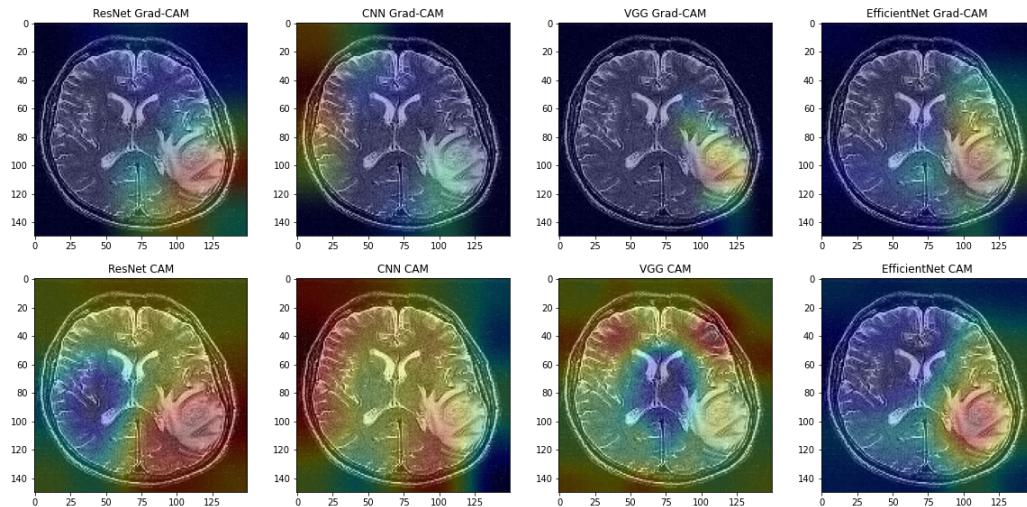
Kolejnym wnioskiem, który udało mi się wyciągnąć jest niepoprawne działanie metody *Grad-CAM* dla części obrazów wejściowych. Zaobserwowałem, że jeśli zdjęcie wejściowe czaszki zostało wykonane z tyłu bądź z boku i obejmuje więcej fragmentów niż sam mózg, to metoda *Grad-CAM* bardzo często nie zwraca poprawnego wyniku.



Rys. 5.4. Niepoprawne działanie metody *Grad-CAM* - przykładowe obrazy wejściowe

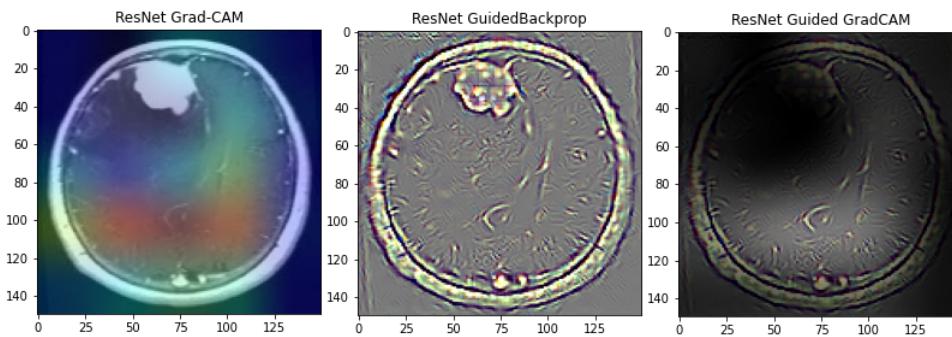
Niepoprawne działanie metody *Grad-CAM* skutkuje najczęściej zwróceniem macierzy składającej się z wartości *nan* (z ang. *not a number*, wartość nieliczbowa). Warto jednak zauważyć,

że w przypadkach gdy *Grad-CAM* działa poprawnie, prawie zawsze zwraca on mniejszy, bardziej konkretny obszar istotności niż metoda *CAM*, dlatego też lepiej on współpracuje z metodą Nadzorowanej Propagacji Wstecz, wyszczególniając z jej wyniku bardziej konkretny fragment.



Rys. 5.5. Porównanie wyników metody *Grad-CAM* oraz *CAM* - *Grad-CAM* zwraca mniejsze, bardziej konkretne obszary

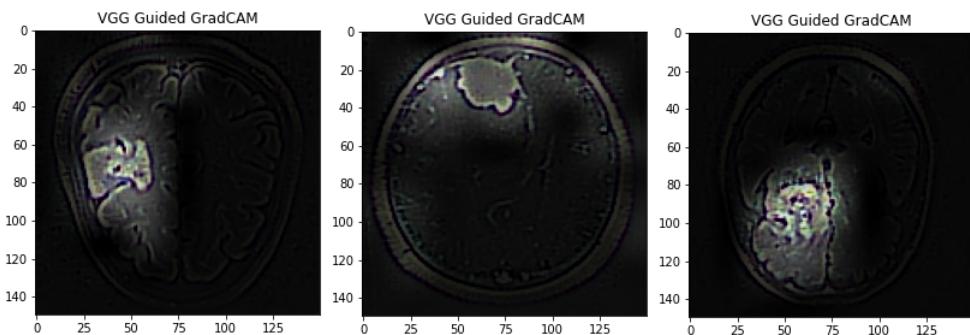
Warto jednak zauważyć, że jeśli *Grad-CAM* co do zasady zwraca mniejszy, bardziej konkretny obszar istotności, to może dojść do sytuacji, że jeśli metoda *Grad-CAM* nie zwróci odpowiedniego obszaru istotności, to wizualizacja uzyskana za pomocą metody *Guided Grad-CAM* całkowicie straci sens, ponieważ nie będzie na niej widać wpływającego na podejmowaną decyzję fragmentu mózgu, co obrazuje rys. 5.6.



Rys. 5.6. Błędne działanie metody *Grad-CAM* powoduje, że metoda *Guided Grad-CAM* zwraca nieużyteczny wynik, mimo że wynik działania Nadzorowanej Propagacji Wstecz jest poprawny

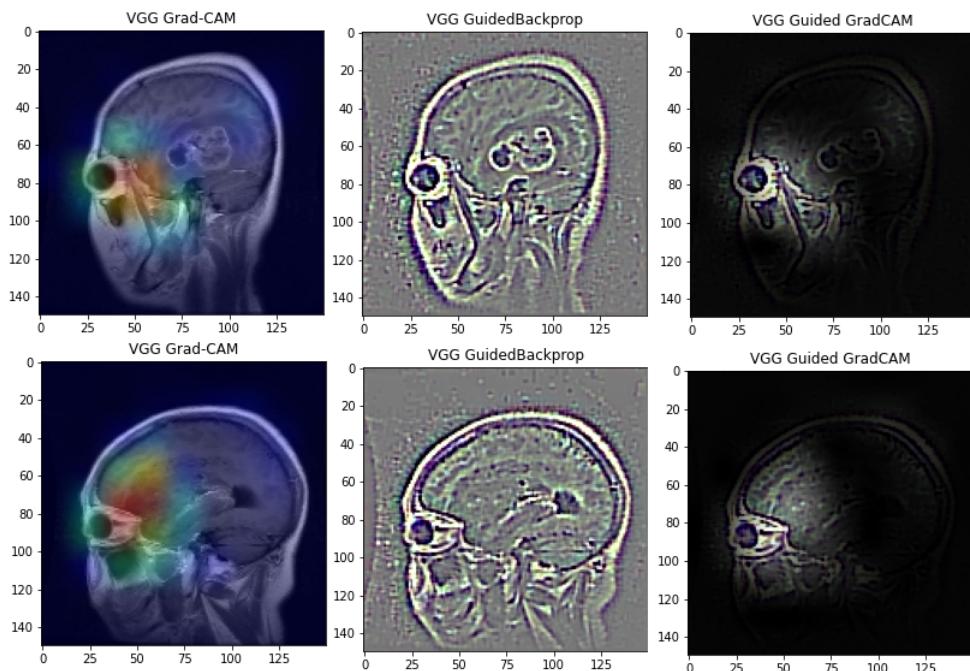
Jeśli jednak metoda *Grad-CAM* zadziała poprawnie i zwróci względnie mały obszar istotności, co dzieje się szczególnie w przypadku modelu bazującego na sieci *VGG*, to wizualizacja uzyskana za pomocą metody *Guided Grad-CAM* będzie najbardziej wartościowa, ponieważ

uzyskamy praktycznie samego guza mózgu, którego szukamy, a reszta mózgu zostanie zaciemniona, co ukazuje rys. 5.7.



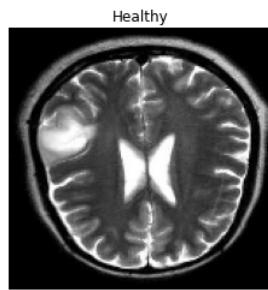
Rys. 5.7. Najlepsze wyniki metody *Guided Grad-CAM* uzyskane dla modelu bazującego na sieci *VGG*

Zostając przy temacie związanym z siecią *VGG*, zauważylem, że jeśli na zdjęciu wejściowym znajdują się oczodoły, to są one często mylone z guzem mózgu (ze względu na ovalny kształt) co może skutkować niepoprawną predykcją w przypadku osób zdrowych, a także błędny obszarem istotności w przypadku osób chorych. Obserwacja ta dowodzi temu, że metody wyznaczania obszarów istotności mogą być stosowane do walidacji poprawności zwracanych przez sieci wyników - rys. 5.8.

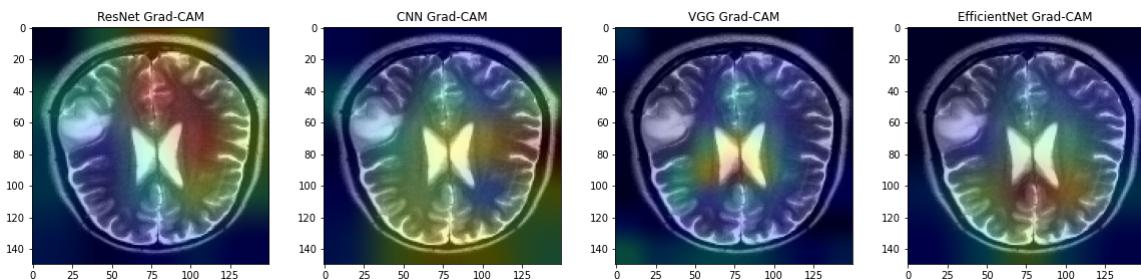


Rys. 5.8. Wpływ obecności na zdjęciu wejściowym oczodołów na wyznaczony obszar istotności oraz predykcję.

Kolejną obserwacją dotyczy zdjęcia wejściowego przedstawionego na rys. 5.9. Wszystkie zastosowane architektury sieci zwróciły rezultat, który wskazuje, że jest to mózg osoby zdrowej, co jest zgodne z etykietą danych. Po lewej stronie mózgu widoczna jest jednak zmiana, która przypomina nowotworową. Gdy przyjrzymy się wynikom metody *Grad-CAM* widocznym na rys. 5.10 możemy zauważyc, że żadna z sieci nie zwróciła obszaru istotności, który by tę podejrzana zmianę obejmował. Może to oznaczać, że jest to prawidłowe zachowanie, ale także z drugiej strony, że sieci nie zadziałyły poprawnie. Dzięki obszarom istotności wyniki działania takich sieci można poddać wątpliwości i powtórnej ocenie lekarza, co może zmniejszyć ilość błędnych predykcji.

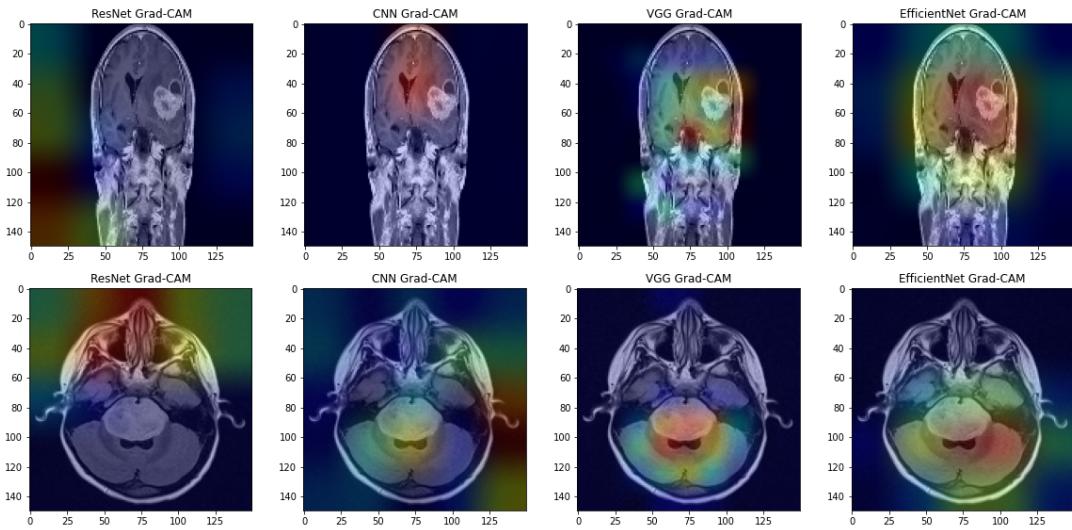


Rys. 5.9. Zdjęcie wejściowe oznaczone w zbiorze danych jako Zdrowy (z ang. *Healthy*), pomimo że widoczna jest zmiana przypominająca nowotworową.



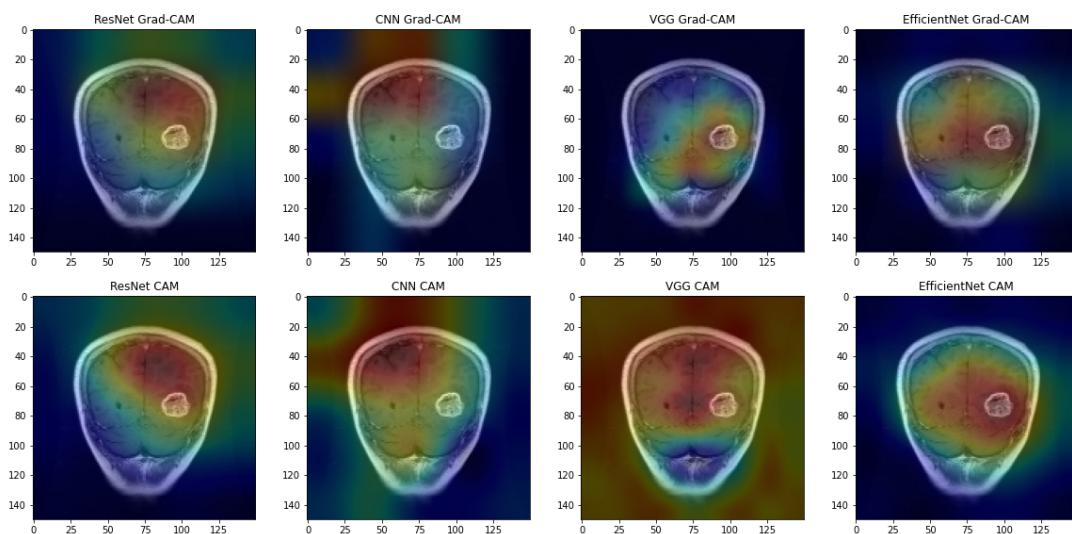
Rys. 5.10. Wynik metody *Grad-CAM* dla obrazu widocznego na rys. 5.9

Metody wyznaczania obszarów istotności mogą też pomóc w ocenie tego, które sieci zadziałyły poprawnie. Rys 5.11 przedstawia obszary istotności dla dwóch różnych obrazów wejściowych. W obu przypadkach prosta sieć *CNN* oraz model bazujący na sieci *ResNet* zwróciły błędne wyniki świadczące o tym, że na obrazie wejściowym widoczny jest mózg osoby zdrowej, a modele bazujące na sieciach *VGG* oraz *EfficientNet* zwróciły poprawny wynik. Gdybyśmy nie stosowali metod analizy obszarów istotności, to zadecydowanie, które metody zwróciły poprawny wynik byłoby trudne. Patrząc jednak na obszary istotności, oczywistym staje się, które sieci działają błędnie. Obszary istotności dla prostej sieci *CNN* oraz sieci *ResNet* informują, że sieci te skupiały się na niewłaściwych fragmentach obrazu.



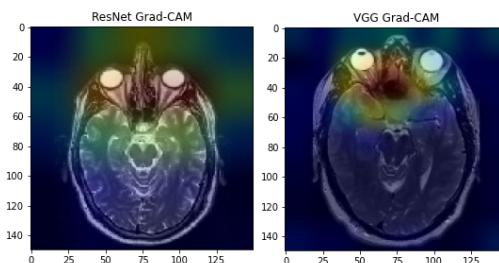
Rys. 5.11. Porównanie obszarów istotności dla sieci zwracających różne wyniki

Testy manualne pokazały jak losową i niedokładną siecią jest prosta sieć *CNN*. Rys. 5.12 przedstawia przykładową wizualizację sytuacji, która powtarza się bardzo często: trzy inne architektury podejmują decyzje bazując na obszarach, gdzie występuje nietypowa zmiana. Sieć *CNN* często zwraca dość losową część mózgu jako obszar o największej istotności. Oznacza to, że metody analizy obszarów istotności pozwalają na porównanie sieci pod kątem jakości podejmowanych decyzji i wyeliminowanie sieci, która mimo że osiąga dobry wynik na zbiorze danych (ponad 90% dokładności) to podejmuje decyzje w sposób niekonsekwencki i może się pomylić w najważniejszym momencie.



Rys. 5.12. Rysunek przedstawiający losowość obszarów istotności generowanych dla prostej sieci *CNN*

Ostatnią rzeczą, na którą zwróciłem uwagę, jest fakt, że w przypadku zdrowych mózgów, sieci często podejmują decyzję zwracając jako obszar istotności oczodoły, jeśli tylko występują one na obrazie wejściowym. Obrazuje to rys. 5.13



Rys. 5.13. Oczodoły jako obszar o największej istotności w przypadku zdrowych mózgów

5.2. Wyniki testów statystycznych

5.2.1. Izolowane testy metod dla różnych architektur sieci

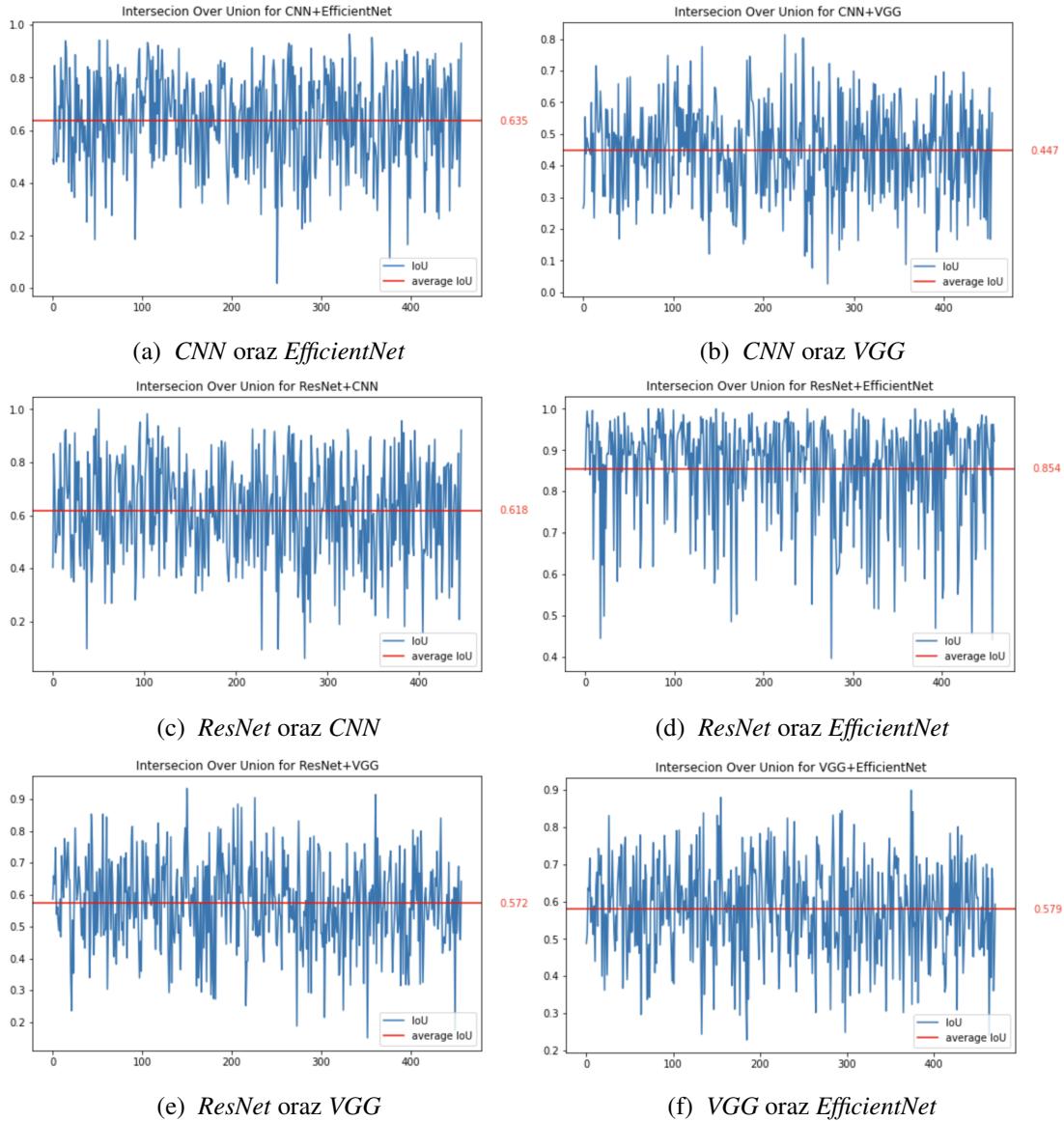
Aby dokładnie przeanalizować każdą z metod zdecydowałem się zastosować architektury sieci konwolucyjnych różniące się ilością warstw, parametrami, sposobem skalowania a także zastosowaniem w nich uczenia przez transfer wiedzy. Dlatego też pierwszym testem jaki zdecydowałem się przeprowadzić było porównanie wyników danej metody dla różnych architektur sieci. Tym sposobem chciałem sprawdzić, czy architektury różniące się w tak wielu, wymienionych wyżej aspektach podejmują decyzję bazując na podobnych cechach obrazu.

5.2.1.1. CAM

Pierwszą metodą, którą zdecydowałem się przetestować było mapowanie aktywacji klas - *CAM*. Metryką do oceny działania metod wyznaczania map istotności, którą zastosowałem jako pierwszą, była opisana w rozdziale 4.6.2.1: *Intersection over Union*.

Na warsztat wziąłem obrazy, na których występuje guz mózgu, ponieważ zaobserwowałem, że w takim przypadku sieć powinna właśnie na nim się skupiać przy podejmowaniu decyzji, co powinno zwiększyć spójność cech wypływających na podejmowane przez sieć decyzje. Potwierdzają to przeprowadzone przeze mnie badania, które wskazują, że średnia wartość parametru *IoU* dla zdjęć zawierających guza mózgu wynosi około 61,75% a w przypadku wszystkich zdjęć wartość ta wynosi około 61,19%. Oprócz tego, w przypadku zdjęć, na których występuje guz mózgu rośnie pokrycie dla najlepiej działających sieci - *ResNet* oraz *EfficientNet*. Do

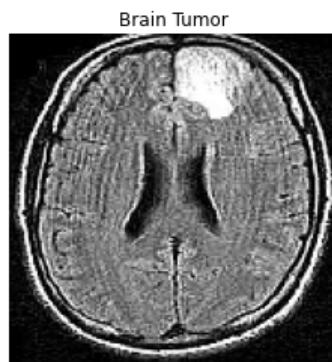
porównania wybrałem jedynie obrazy wejściowe, dla których predykcja była poprawna. Pozwoliło to usunąć punkty skrajne takie jak wartość wskaźnika IoU poniżej wartości 0,2 dla sieci *ResNet* oraz *EfficientNet*.



Rys. 5.14. Porównanie parametrów *Intersection over Union* dla wszystkich kombinacji wytrenowanych modeli sieci konwolucyjnych - metoda *CAM*

Rys. 5.14 przedstawia otrzymane przeze mnie wyniki. Każdy wykres na tym rysunku obrazuje porównanie (za pomocą IoU) wyników dla dwóch różnych architektur sieci. Jako że stosowałem 4 architektury, to liczba wykresów, równa liczbie par wybranych bez powtórzeń, wynosi 6. Na każdym z wykresów oś Y oznacza wartość parametru IoU , oś X oznacza indeks analizowanego zdjęcia. Czerwona pozioma linia oznacza średnią wartość parametru IoU dla wszystkich analizowanych obrazów.

Możemy zaobserwować, że w przypadku metody *CAM* zbiory (obszary skupienia) pokrywające się w największym stopniu występują dla sieci *ResNet* oraz *EfficientNet*. Analizując wykres możemy zauważać, że średnia wartość pokrycia obliczona za pomocą parametru *IoU* wynosi 85,4%, co całkowicie odbiega od reszty par. Obserwując wyniki dla poszczególnych obrazów możemy zauważać, że dla tych właśnie architektur wahania wartości parametru *IoU* są najmniejsze, utrzymuje on stały, dość wysoki poziom. Wyjątkiem od tej reguły są pojedyncze zdjęcia dla których pokrycie spada poniżej 50%.



Rys. 5.15. Przykładowy obraz wejściowy, dla którego *IoU* dla sieci *ResNet* oraz *EfficientNet* ma małą wartość (ok. 41%)

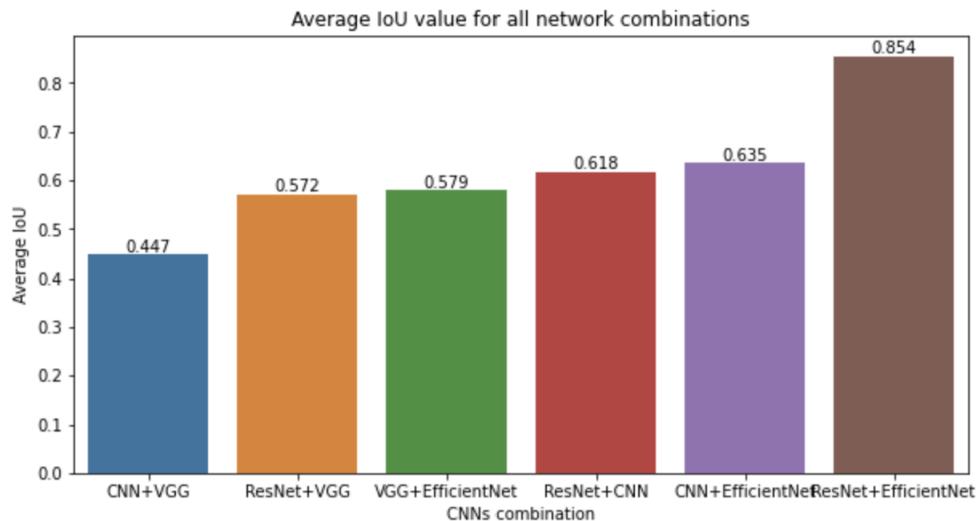
Dość podobne pokrycie - od 57,2% do 63,5% - możemy zaobserwować dla kolejnych par sieci:

1. *ResNet* oraz *VGG*,
2. *VGG* oraz *EfficientNet*,
3. *ResNet* oraz *CNN*,
4. *CNN* oraz *EfficientNet*.

Najmniejsze pokrycie występuje dla prostej sieci *CNN* oraz sieci *VGG*. W ich przypadku utrzymuje się dość stały niski poziom, którego średnia wynosi 44,7%. Wartość pokrycia nigdy nie przekracza 80% a dość często spada poniżej 20%. Jest to obserwacja, której mogliśmy się spodziewać, ponieważ są to dwie najprostsze sieci o najmniejszej liczbie warstw.

Rys. 5.16 przedstawia podsumowanie opisanych powyżej wniosków dotyczących średniej wartości parametru *IoU* dla różnych kombinacji sieci. Możemy zauważać, że rzeczywiście najgorsze pokrycie występuje, gdy porównujemy jakąkolwiek architekturę sieci z architekturą bazującą na *VGG* (3 najgorsze wyniki). Zdecydowanie najlepsze pokrycie otrzymujemy gdy zestawimy ze sobą sieć *ResNet* oraz *EfficientNet*.

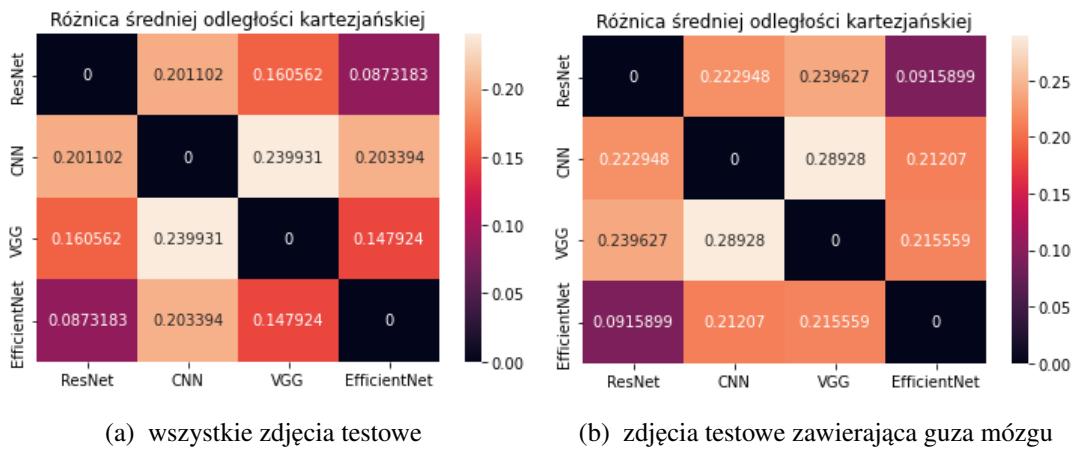
Drugi zastosowany przeze mnie parametr do oceny wyników metody *CAM* dla różnych architektur bazował na geometrycznym środku ciężkości opisany w rozdziale 4.6.2.2.



Rys. 5.16. Porównanie średnich wartości współczynnika IoU - *CAM*

Zdecydowałem się dokonać wizualizacji za pomocą macierzy. Na przecięciach wierszy oraz kolumn przypisanych odpowiednim architekturom sieci umieściłem parametr porównawczy. Jest to średnia różnica odległości środków ciężkości na przestrzeni wszystkich zdjęć wejściowych przeskalowana do zakresu $<0, 1>$ gdzie 0 oznacza zerową różnicę a 1 oznacza maksymalną możliwą różnicę (środki ciężkości znajdują się po przekątnej obrazu).

Warto zauważyć, że nieważne czy weźmiemy pod uwagę jedynie zdjęcia zawierające guza mózgu, czy wszystkie, mimo tego że liczby się różnią, to obie macierze pokolorowane są w podobny sposób. Oznacza to, że możemy wnioskować w ten sam sposób korzystając z całego zbioru zdjęć, ale także jedynie ze zdjęć zawierających raka mózgu.



(a) wszystkie zdjęcia testowe

(b) zdjęcia testowe zawierająca guza mózgu

Rys. 5.17. Mapy przedstawiające średnią różnicę odległości karteżjańskiej dla różnych architektur sieci neuronowych - *CAM*

Jak możemy odczytać z rys. 5.17, najmniejsza różnica pomiędzy środkami ciężkości występuje dla sieci *ResNet* oraz *EfficientNet* i wynosi średnio około 8,7% długości przekątnej obrazka

w przypadku gdy analizujemy wszystkie zdjęcia i 9,2% przekątnej gdy weźmiemy pod uwagę jedynie zdjęcia zawierające raka mózgu. Jest to zbieżne z wnioskami otrzymanymi za pomocą parametru IoU , metryka ta potwierdza, że obszary istotności dla tych dwóch architektur są najbardziej zbliżone.

Gdy porównujemy sieć *VGG* z siecią *CNN*, wyniki, tak jak w przypadku parametru IoU różnią się najbardziej. Warto jednak zaobserwować, że w przypadku parametru IoU wyniki sieci *VGG* były najmniej zbieżne z wynikami dla wszystkich innych sieci. Gdy porównujemy środki ciężkości sieci *VGG* ze środkami ciężkości dla sieci *ResNet* oraz *EfficientNet* to otrzymujemy bardzo małe różnice. Może to prowadzić do wniosku, że sieć *VGG* wskazuje podobny obszar istotności, jednak składający się z mniejszej liczby pikseli. Wyjaśniałoby to fakt podobnego środka ciężkości, lecz małego, że względem na małą powierzchnię, pokrycia wyznaczanego za pomocą parametru IoU .

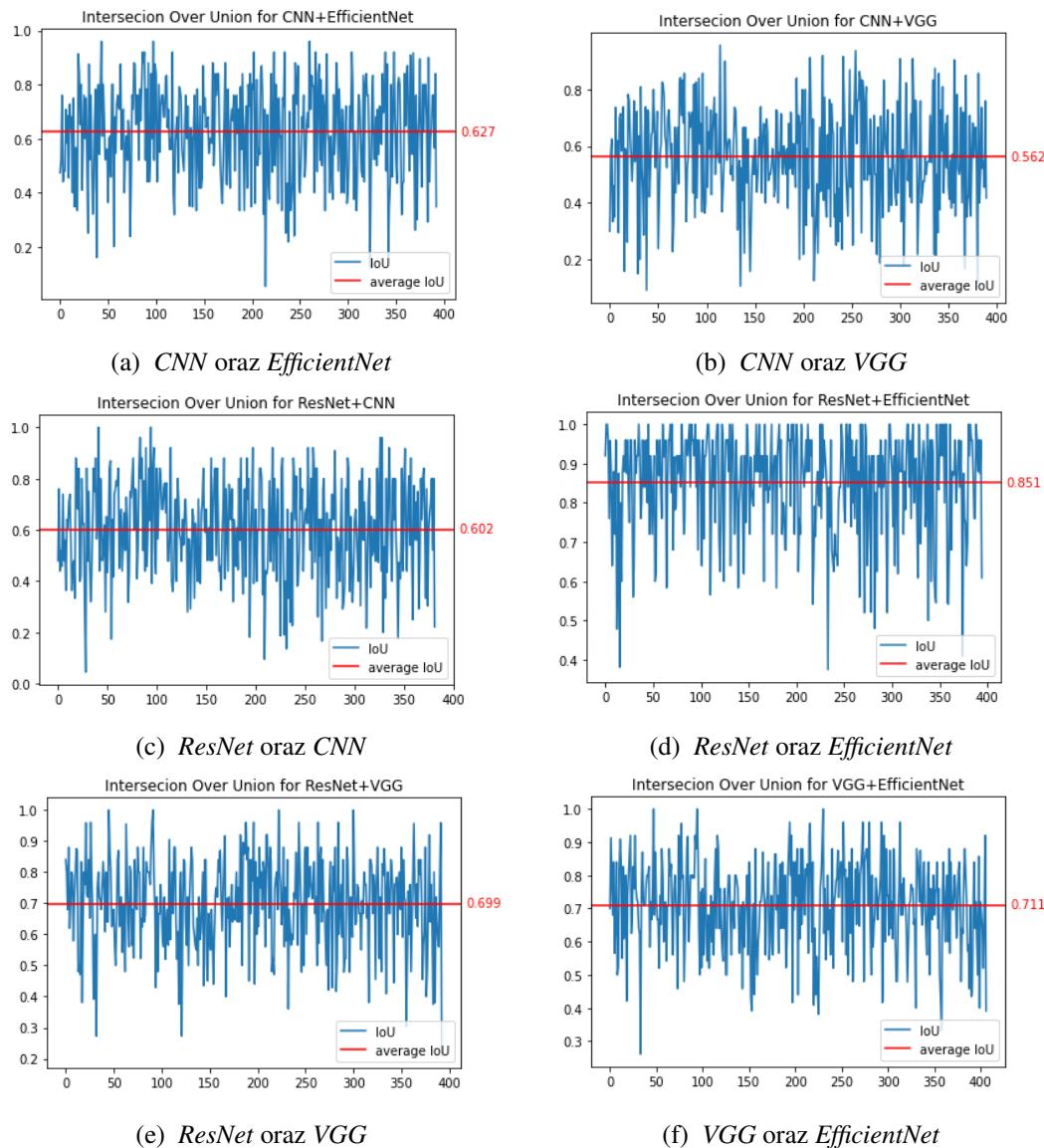
5.2.1.2. *Grad-CAM*

Następnie przeprowadziłem podobne testy i wizualizacje dla metody *Grad-CAM*. Analizując rysunek 5.18 możemy zauważyć, że tak jak w przypadku metody *CAM* największą wartość pokrycia mierzoną za pomocą parametru IoU otrzymujemy dla kombinacji sieci *ResNet* z *EfficientNet* i wynosi ona 85,1% a wartości skrajnie złych jest bardzo mało.

Przeciwieństwem powyżej opisanej relacji jest kombinacja sieci *CNN* oraz *VGG*. W tym przypadku otrzymujemy pokrycie na poziomie 56,2% (niskie, jednak w przypadku metody *CAM* wynosiło ono 44,7%). W przypadku tych sieci obserwujemy bardzo duże wahania wartości parametru IoU co może wskazywać na brak stabilności obszarów skupienia.

Reszta kombinacji sieci dzieli się na dwie grupy. Pierwsza z nich składa się z sieci *CNN* w zestawieniu z sieciami *ResNet* i *EfficientNet*. W tym przypadku IoU wynosi odpowiednio 60,2% oraz 62,7%. Drugą grupą jest sieć *VGG* z zestawieniu z dwiema najbardziej pokrywającymi się sieciami. W tym przypadku IoU wynosi 69,9% oraz 71,1%. Jest to obserwacja bardziej zgodna z intuicją, ponieważ sieć *VGG* składa się z większej liczby warstw niż prosta sieć *CNN*, więc powinna zwracać bardziej zbliżone obszary istotności do sieci *EfficientNet* oraz *ResNet*. Wnioski te zgodne są także z testami manualnymi, które wykazały, że metoda *Grad-CAM* zwraca dla wszystkich architektur sieci mniejsze obszary istotności od metody *CAM*, czyli bardziej zgodne z wynikami dla sieci *VGG*.

Warto zauważyć, że metoda *Grad-CAM* w naszym przypadku nie była w stanie obliczyć obszarów istotności dla wszystkich obrazów wejściowych. Obrazuje to tab. 5.1: metoda *Grad-CAM* najgorzej działała dla sieci *VGG*, dla której zwróciła 93,58% poprawnie obliczonych map. Niemniej jednak, w przypadku metody *Grad-CAM* otrzymujemy średnią wartość parametru IoU na poziomie 67,5%, czyli około 5,7% więcej niż w przypadku metody *CAM*. Może mieć



Rys. 5.18. Porównanie parametrów *Intersection over Union* dla wszystkich kombinacji wytrenowanych modeli sieci konwolucyjnych - metoda *Grad-CAM*

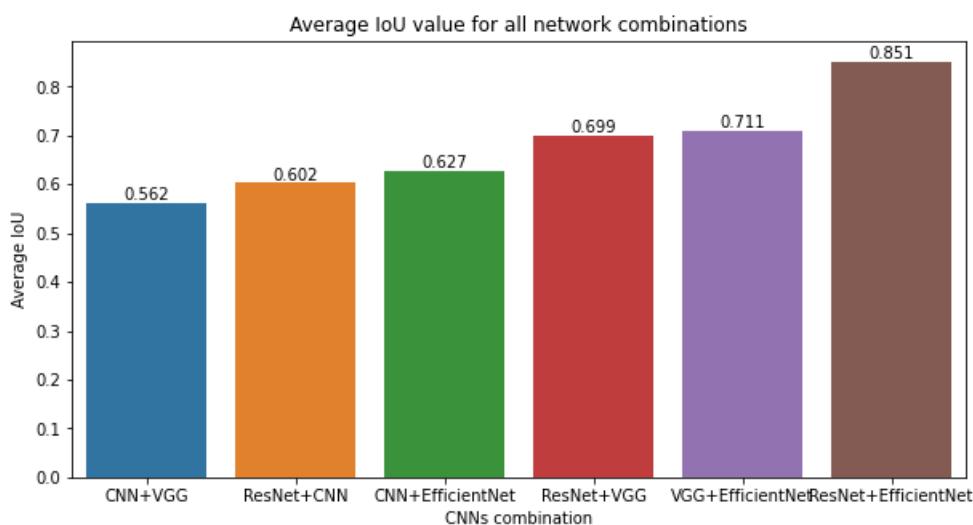
na to jednak wpływ fakt, że najbardziej trudne, skrajne przypadki obrazów wejściowych zostają odrzucone.

Rys. 5.20 przedstawia porównanie średniej odległości kartezjańskiej dla środków ciężkości map istotności wyznaczonych za pomocą metody *Grad-CAM*. Najmniejsza średnia różnica środków ciężkości występuje dla sieci *EfficientNet* oraz *VGG* a także *EfficientNet* oraz *ResNet*, a największa dla sieci *VGG* oraz *CNN* - jest to zbieżne z wynikami uzyskanymi za pomocą metryki *IoU*. Takie same wyniki uzyskałem również analizując metodę *CAM*.

Generalnie możemy zaobserwować, że w przypadku gdy nie bierzemy pod uwagę prostej sieci *CNN*, dla której otrzymujemy najbardziej odbiegające od wyników innych sieci wyniki

Tabela 5.1. Zestawienie poprawnie wyznaczonych map istotności dla metody *Grad-CAM*

Sieć konwolucyjna	Poprawnie wyznaczone mapy istotności
<i>ResNet</i>	454 (94%)
<i>CNN</i>	483 (100%)
<i>VGG</i>	452 (93, 58%)
<i>EfficientNet</i>	474 (98, 14%)

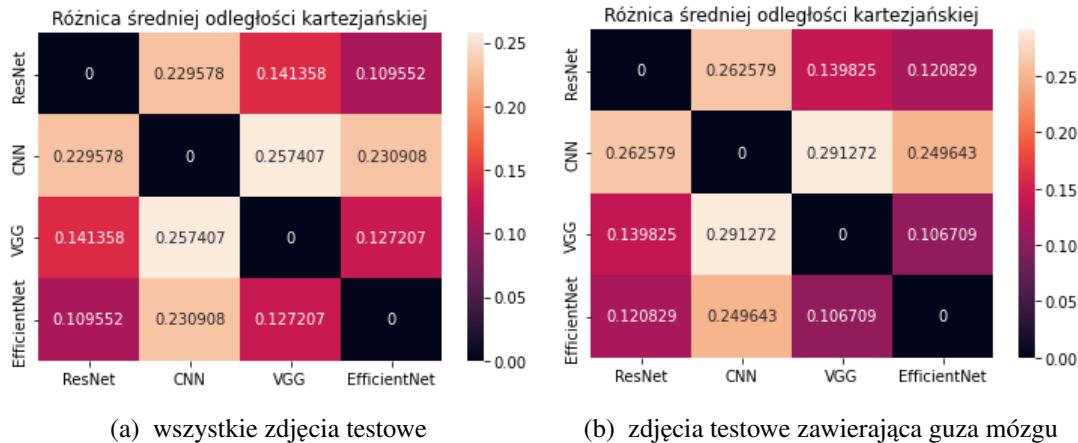


Rys. 5.19. Porównanie średnich wartości współczynnika *IoU* - *Grad-CAM*

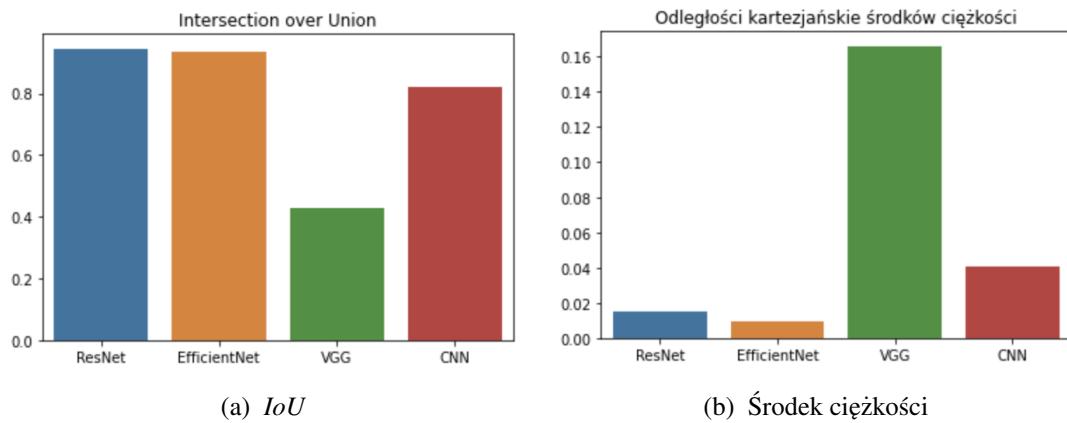
środków ciężkości (różnica na poziomie 24% długości przekątnej), to wyniki są dość zbliżone. Dla zdjęć, na których występują guzy mózgów otrzymujemy średnią odległość środków ciężkości na poziomie od 10,7% do 14%. Potwierdza to fakt, że bardziej złożone sieci konwolucyjne zwracają uwagę na cechy bardziej abstrakcyjne, dzięki czemu ich mapy istotności są podobne.

5.2.2. Testy porównawcze metody *CAM* oraz *Grad-CAM*

Rys. 5.21 przedstawia porównanie wyników działania metody *CAM* oraz *Grad-CAM* zmierzane za pomocą średnich odległości kartezjańskich pomiędzy środkami ciężkości a także za pomocą parametru *Intersection over Union*. Można zauważyć, że w przypadku sieci *ResNet* oraz *EfficientNet* pokrycie zmierzane za pomocą parametru *IoU* jest najwyższe i wynosi około 95%, a średnia odległość kartezjańska środków ciężkości wynosi odpowiednio około 1,5% oraz 1%. W przypadku prostej sieci *CNN* pokrycie jest na dość wysokim poziomie (wynosi bowiem ponad 80%) jednak różnica środków ciężkości wynosi około 4% przekątnej. Najgorzej prezentuje się sieć *VGG* dla której pokrycie jest na bardzo niskim poziomie (około 40%) a różnica



Rys. 5.20. Mapy przedstawiające średnią różnicę odległości kartezańskiej dla różnych architektur sieci neuronowych - *Grad-CAM*



Rys. 5.21. Porównanie działania metody *CAM* oraz *Grad-CAM*

środków ciężkości wynosi ponad 16%. Obserwacja ta jest jednak zbieżna z wnioskami wyciągniętymi za pomocą testów manualnych i wynika z tego, że wyniki zwarcane przez metodę *Grad-CAM* szczególnie dla sieci *VGG* mają znacznie mniejszą powierzchnię (małe pokrycie) a ich środki ciężkości są z tego powodu lekko przesunięte. Oznacza to, że generalnie metody *Grad-CAM* oraz *CAM* zwarczą bardzo podobne wyniki, jednak istnieją takie architektury sieci, dla których można zaobserwować pewne różnice.

Warto dodać, że pokrycie zmierzone pomiędzy różnymi metodami wyznaczania obszarów istotności dla tych samych architektur sieci jest wyższe niż w przypadku porównywania wyników danej metody dla różnych architektur sieci. Różnica pomiędzy środkami ciężkości jest niższa. Oznacza to, że metody *Grad-CAM* oraz *CAM* wskazują podobne obszary istotności, co może potwierdzać ich prawdziwość.

6. Podsumowanie

Celem niniejszej pracy była ocena stanu wiedzy oraz porównanie dostępnych metod w zakresie analizy obszarów istotności w ocenie działania głębokich sieci neuronowych dla obrazów medycznych. W zakres pracy wchodziła ocena działania czterech metod dla różnych architektur sieci przeprowadzona na podstawie testów manualnych oraz statystycznych.

Projekt inżynierski zrealizował powyższy cel, poddając dogłębnej analizie literaturę dotyczącą obszarów istotności. Ze względu na odpowiedni dobór różnych architektur sieci neuronowych uzyskałem wiele scenariuszy testowych. Zastosowanie gotowych architektur sieci zapewniło badanie wpływu uczenia przez transfer na działanie metod analizy istotności. Dzięki sieci *ResNet* zbadałem wpływ obecności połączeń rezydualnych na działanie metod. Zestawienie sieci *EfficientNet* z siecią *CNN* pozwoliło na analizę wpływu głębokości sieci na wynikowe mapy istotności. Informacje te zestawione ze sobą zapewniają głęboki wgląd do możliwości stosowania metod analizy obszarów istotności w obrazach medycznych.

Wybierając metodę wyznaczania obszarów istotności należy wziąć pod uwagę cechę, na której zależy nam najbardziej. Metoda *Grad-CAM* jest najbardziej znana i jest dostępnych wiele jej gotowych implementacji. Jej największymi zaletami są szybkość działania, dokładność uzyskanych wyników a także rozróżnialność klasowa. Wadą metody *Grad-CAM* jest jednak zawodność, ponieważ nie dla wszystkich obrazów wejściowych generuje ona poprawne wyniki. Jeżeli zależy nam na niezawodności, możemy wybrać metodę *CAM*, która zwraca jednak mniej szczegółowe wyniki niż metoda *Grad-CAM*. Jeśli bierzemy pod uwagę dużą rozdzielncość uzyskanych wyników, należy wybrać metodę Nadzorowanej Propagacji Wstecz, bądź jej połączenie z metodą *Grad-CAM* czyli *Guided Grad-CAM*. Wizualizacje te mają inny charakter niż dwie pierwsze wymienione metody - nie zwracają one mapy ciepła, a pełną wizualizację cech, które spowodowały największą aktywację neuronów.

Dalszym kierunkiem rozwoju powyższego projektu może być zastosowanie innego medycznego zbioru danych (rentgen klatki piersiowej, choroby oka) i porównanie dla niego wyników. Inną możliwością jest poddanie analizie metod badania interpretowalności sieci neuronowych i zestawienie ich z obszarami istotności.

Bibliografia

- [1] Aurélien Géron. „*Uczenie maszynowe z użyciem Scikit-Learn i TensorFlow*”. Warszawa: Wydawnictwo Helion, 2020.
- [2] Rodrigo De Oliveira i in. „*A System Based on Artificial Neural Networks for Automatic Classification of Hydro-generator Stator Windings Partial Discharges*”. https://www.researchgate.net/publication/320270458_A_System_Based_on_Artificial_Neural_Networks_for_Automatic_Classification_of_Hydro-generator_Stator_Windings_Partial_Discharges. 2017. DOI: 10.1590/2179-10742017v16i3854.
- [3] Weitian Tong i in. „*Deep learning PM2.5 concentrations with bidirectional LSTM RNN*”. https://www.researchgate.net/figure/A-multi-layer-neural-network-with-n-inputs-at-least-two-hidden-layers-and-one-output_fig5_330230427. 2019. DOI: 10.1007/s11869-018-0647-4.
- [4] David E. Rumelhart. „*Learning Internal Representations by Error Propagation*”. <https://apps.dtic.mil/sti/citations/ADA164453>. 1985.
- [5] Shruti Jadon. „*Introduction to Different Activation Functions for Deep Learning*”. <https://medium.com/@shrutijadon/survey-on-activation-functions-for-deep-learning-9689331ba092>. Stronę odwiedzono 20-10-2022. 2018.
- [6] Matthew Stewart. „*Simple Introduction to Convolutional Neural Networks*”. <https://towardsdatascience.com/simple-introduction-to-convolutional-neural-networks-cdf8d3077bac?gi=eb335346d214>. Stronę odwiedzono 12-10-2022. 2019.
- [7] Md. Zahangir Alom. „*A State-of-the-Art Survey on Deep Learning Theory and Architectures*.” https://www.researchgate.net/publication/331540139_A_State-of-the-Art_Survey_on_Deep_Learning_Theory_and_Architectures. 2019. DOI: 10.3390/electronics8030292.
- [8] Dhaval Patel. „*Simple explanation of convolutional neural network*”. <https://youtu.be/zfIAzpy9NM?t=1347>. Stronę odwiedzono 10-10-2022. 2020.
- [9] Stevo Bozinovski. „*Reminder of the First Paper on Transfer Learning in Neural Networks, 1976*”. 2020. DOI: <https://doi.org/10.31449/inf.v44i3.2828>.

- [10] Dipanjan Sarkar, Raghav Bali i Tamoghna Ghosh. „*Hands-On Transfer Learning with Python: Implement Advanced Deep Learning and Neural Network Models Using TensorFlow and Keras*”. Packt Publishing, 2018.
- [11] Francke Peixoto. „*InceptionV3 — Transfer Learning / Conceito básico na prática*”. <https://franckepeixoto.medium.com/inceptionv3-transfer-learning-conceito-b%C3%A1sico-na-pr%C3%A1tica-381cb5fe9ef>. Stronę odwiedzono 01-11-2022. 2020.
- [12] Karen Simonyan i Andrew Zisserman. „*Very Deep Convolutional Networks for Large-Scale Image Recognition*”. <https://arxiv.org/abs/1409.1556>. 2014. DOI: [10.48550/ARXIV.1409.1556](https://doi.org/10.48550/ARXIV.1409.1556).
- [13] GeeksForGeeks. „*VGG-16 | CNN model*”. <https://www.geeksforgeeks.org/vgg-16-cnn-model/>. Stronę odwiedzono 03-11-2022. 2022.
- [14] Kaiming He i in. „*Deep Residual Learning for Image Recognition*”. <https://arxiv.org/abs/1512.03385>. 2015. DOI: [10.48550/ARXIV.1512.03385](https://doi.org/10.48550/ARXIV.1512.03385).
- [15] Mingxing Tan i Quoc V. Le. „*EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks*”. <https://arxiv.org/abs/1905.11946>. 2019. DOI: [10.48550/ARXIV.1905.11946](https://doi.org/10.48550/ARXIV.1905.11946).
- [16] Karen Simonyan, Andrea Vedaldi i Andrew Zisserman. „*Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps*”. <https://arxiv.org/abs/1312.6034>. 2013. DOI: [10.48550/ARXIV.1312.6034](https://doi.org/10.48550/ARXIV.1312.6034).
- [17] Matthew D Zeiler i Rob Fergus. „*Visualizing and Understanding Convolutional Networks*”. <https://arxiv.org/abs/1311.2901>. 2013. DOI: [10.48550/ARXIV.1311.2901](https://doi.org/10.48550/ARXIV.1311.2901).
- [18] Adrian Horzyk. „*Sztuczna Inteligencja - Uczenie Głębokie i Głębokie Sieci Neuronowe*”. <https://home.agh.edu.pl/~horzyk/lectures/ai/SztucznaInteligencja-UczenieG%C5%82%C4%99bokichSieciNeuronowych.pdf>. Str. 12.
- [19] Jost Tobias Springenberg i in. „*Striving for Simplicity: The All Convolutional Net*”. <https://arxiv.org/abs/1412.6806>. 2014. DOI: [10.48550/ARXIV.1412.6806](https://doi.org/10.48550/ARXIV.1412.6806).
- [20] Bolei Zhou i in. „*Learning Deep Features for Discriminative Localization*”. <https://arxiv.org/abs/1512.04150>. 2015. DOI: [10.48550/ARXIV.1512.04150](https://doi.org/10.48550/ARXIV.1512.04150).
- [21] Ramprasaath R. Selvaraju i in. „*Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization*”. <https://doi.org/10.1007/s11263-019-01228-7>. 2019. DOI: [10.1007/s11263-019-01228-7](https://doi.org/10.1007/s11263-019-01228-7).
- [22] PREET VIRADIYA. „*Brian Tumor Dataset*”. <https://www.kaggle.com/datasets/preetviradiya/brian-tumor-dataset>. Stronę odwiedzono 15-10-2022.
- [23] Francois Chollet i in. „*Keras*”. 2015. URL: <https://github.com/fchollet/keras>.

- [24] François Chollet. „*Grad-CAM class activation visualization*”. https://keras.io/examples/vision/grad_cam/. Stronę odwiedzono 10-11-2022.
- [25] Patrick Brus. „*Class Activation Mapping*”. <https://towardsdatascience.com/class-activation-mapping-using-transfer-learning-of-resnet50-e8ca7cf657e>. Stronę odwiedzono 10-11-2022.
- [26] Hoa Nguyen. „*GradCAM_and_GuidedGradCAM_tf2*”. https://github.com/nguyenhoa93/GradCAM_and_GuidedGradCAM_tf2. Stronę odwiedzono 10-11-2022.
- [27] Adrian Rosebrock. „*Intersection over Union (IoU) for object detection*”. <https://pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/>. Stronę odwiedzono 14-11-2022. 2016.