Politechnika Warszawska

WYDZIAŁ ELEKTRONIKI
I TECHNIK INFORMACYJNYCH

Instytut Informatyki

Praca dyplomowa magisterska

na kierunku informatyka
w specjalności Sztuczna inteligencja

Generowanie podpisów do obrazków

Kacper Kostecki
Numer albumu 300236

promotor
dr inż. Piotr Andruszkiewicz

WARSZAWA 2024

Generowanie podpisów do obrazków

Streszczenie. We wszelkich dziedzinach życia człowieka obrazy stanowią istotny element komunikacji i przekazu informacji, jednak człowiek napotyka trudności w efektywnym przetwarzaniu i wykorzystywaniu tej ogromnej ilości wizualnych danych. Analiza i interpretacja obrazów są zadaniem wymagającym znacznego nakładu czasu i wysiłku intelektualnego. Rozwiązaniem tego problemu są algorytmy uczenia maszynowego umożliwiające automatyczne generowanie podpisów do obrazów, co znacząco ułatwia cały proces wydobywania z nich informacji. Jednakże ich wykorzystanie generuje kolejną przeszkodę: zasoby komputerowe. W celu rozwiązania tak skomplikowanego problemu, jakim jest generowanie podpisów do obrazów, wykorzystane algorytmy są niezwykle skomplikowane, a co za tym idzie, potrzebują coraz większą ilość zasobów komputerowych. Celem niniejszej pracy dyplomowej było zbadanie skuteczności, jak również efektywności wykorzystywanych algorytmów do generowania podpisów do obrazków.

Słowa kluczowe: podpisy obrazków, język naturalny, wizja komputerowa, zasoby komputerowe

Image caption generation

Abstract. In all areas of human life, images constitute an essential element of communication and information transmission, however, humans encounter difficulties in efficiently processing and utilizing this vast amount of visual data. Analyzing and interpreting images is a task that requires significant time and intellectual effort. Machine learning algorithms provide a solution to this problem by enabling automatic generation of captions for images, greatly facilitating the process of extracting information from them. However, their utilization presents another challenge: computer resources. To address the complex problem of generating captions for images, the employed algorithms are highly intricate, thus requiring increasingly larger amounts of computer resources. The aim of this thesis was to examine the effectiveness and efficiency of the utilized algorithms for generating captions for images.

Keywords: image captioning, natural language processing, computer vision, computer resources

Wersja w języku polskim

Politechnika Warszawska

…….....
miejscowość i data

……..
imię i nazwisko studenta

……..
numer albumu

…….……
wydział kierunek studiów

OŚWIADCZENIE

Świadomy/-a odpowiedzialności karnej za składanie fałszywych zeznań oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie, pod opieką kierującego pracą dyplomową.
Jednocześnie oświadczam, że:

- niniejsza praca dyplomowa nie narusza praw autorskich w rozumieniu ustawy z dnia 4 lutego 1994 roku o prawie autorskim i prawach pokrewnych (Dz.U. z 2021 r., poz. 1062) oraz dóbr osobistych chronionych prawem cywilnym,
- niniejsza praca dyplomowa nie zawiera danych i informacji, które uzyskałem/-am w sposób niedozwolony,
- niniejsza praca dyplomowa nie była wcześniej podstawą żadnej innej urzędowej procedury związanej z nadawaniem dyplomów lub tytułów zawodowych,
- wszystkie informacje umieszczone w niniejszej pracy, uzyskane ze źródeł pisanych i elektronicznych, zostały udokumentowane w wykazie literatury odpowiednimi odnośnikami,
- znam regulacje prawne Politechniki Warszawskiej w sprawie zarządzania prawami autorskimi i prawami pokrewnymi, prawami własności przemysłowej oraz zasadami komercjalizacji.

…….....
(podpis studenta)

Spis treści

1. Wstęp

W ostatnich latach, wraz z rozwojem technologii cyfrowej i internetu, liczba dostępnych obrazów, zdjęć i diagramów znacząco wzrosła. Zarówno w mediach społecznościowych, stronach internetowych, jak i w różnych obszarach działalności, takich jak medycyna, nauka czy przemysł, obrazy stanowią istotny element komunikacji i przekazu informacji. Jednak człowiek napotyka trudności w efektywnym przetwarzaniu i wykorzystywaniu tej ogromnej ilości wizualnych danych. Analiza i interpretacja obrazów są zadaniem wymagającym znacznego nakładu czasu i wysiłku intelektualnego. Człowiek musi poświęcić wiele uwagi i sił, aby opisać, co przedstawia dany obraz, uwzględniając detale, kontekst, emocje czy inne istotne elementy. Ten proces może być jeszcze bardziej uciążliwy w przypadku dużych zbiorów wizualnych danych, gdzie manualne generowanie opisów staje się nieefektywne i czasochłonne. W odpowiedzi na te wyzwania, rozwój technik sztucznej inteligencji i uczenia maszynowego otwiera nowe perspektywy w automatycznym generowaniu podpisów do obrazków. Algorytmy uczenia maszynowego pozwalają na tworzenie modeli komputerowych, które są w stanie analizować obrazy, wyodrębniać z nich cechy, rozpoznawać obiekty i kontekst, a następnie generować opisy tekstowe na podstawie tej wiedzy. Dzięki temu automatyczne generowanie podpisów staje się realnym rozwiązaniem, które może przyspieszyć i ułatwić proces analizy obrazów. Oprócz oszczędności czasu i wysiłku, automatyczne generowanie podpisów do obrazków ma również potencjalne znaczenie dla osób niewidomych. Dla nich dostęp do informacji wizualnych jest ograniczony lub niemożliwy. Generowanie tekstowych opisów obrazów staje się zatem narzędziem, które umożliwia im lepsze zrozumienie i korzystanie z wizualnych treści. Dzięki automatycznym podpisom, osoby niewidome mogą otrzymać opisy obrazów, które pośrednio przekazują informacje wizualne, pozwalając im na lepszą orientację i uczestnictwo w kulturze wizualnej.

1.1. Cel i motywacja pracy

Celem niniejszej pracy było sprawdzenie skuteczności oraz efektywności aktualnych rozwiązań wykorzystywanych do automatycznego generowania opisów obrazów cyfrowych. Szczególna uwaga została zwrócona na poziom wykorzystywanych zasobów komputerowych w stosunku do poprawy samej skuteczności generowania opisów wraz z uwzględnieniem czasu treningowego. Narastającym problemem coraz bardziej rozwijających się algorytmów uczenia maszynowego jest ich rosnące zapotrzebowanie na zasoby komputerowe. Wraz z postępem w dziedzinie sztucznej inteligencji, modele oparte na głębokim uczeniu stają się coraz bardziej skomplikowane i wymagają większej mocy obliczeniowej oraz większych ilości danych treningowych. Jednym z głównych czynników wymagań zasobów komputerowych jest rozmiar i złożoność sieci neuronowych. Modele, takie jak splotowe sieci neuronowe czy rekurencyjne sieci neuronowe, mogą składać się z setek tysięcy lub nawet milionów parametrów. Trenowanie tych modeli wymaga dużych zbiorów danych oraz intensywnego obliczeniowo procesu optymalizacji tych parametrów. Kolejnym aspektem jest wykorzystanie sprzętu. W przypadku dużych i złożonych sieci

neuronowych konieczne jest korzystanie z zaawansowanych jednostek przetwarzania graficznego lub specjalizowanych procesorów tensorowych, które są bardziej efektywne w wykonywaniu operacji macierzowych charakterystycznych dla uczenia maszynowego. Korzystanie z takiego specjalistycznego sprzętu może stanowić wyzwanie ze względu na wysokie koszty i dostępność. Ponadto w samych publikacjach dotyczących proponowanych rozwiązań często nie ma zawartych informacji o efektywności architektury w kontekście zasobów komputerowych.

2. Teoria uczenia maszynowego

Informatyka z dnia na dzień, coraz śmielej wkracza we wszelkie dziedziny ludzkiego życia. Dzięki jej rozwojowi wiele żmudnych, uciążliwych, czy też niebezpiecznych prac wykonywanych przez człowieka może zostać zautomatyzowane. Szczególny wkład w ten gwałtowny rozwój ma konkretna dziedzina informatyki: sztuczna inteligencja. Pozwala ona na rozwiązywanie skomplikowanych zadań, takich jak przetwarzanie obrazów cyfrowych, czy też analiza języka naturalnego. Ze względu na dużą ilość informacji konieczną do przeanalizowania, która zawarta jest, czy to w obrazach, czy w tekstach pisanych przez człowieka oraz ogromną różnorodność tychże danych, niezwykle problematyczne jest sformułowanie uniwersalnych algorytmów, które podjęłyby rozwiązanie problemów, jak na przykład lokalizacja obiektów w obrazach, czy ustalenie semantyki danego zdania. Z pomocą przychodzą tutaj algorytmy uczenia maszynowego należące do rodziny algorytmów sztucznej inteligencji. Pozwalają one przy pomocy uniwersalnego podejścia sformułować rozwiązanie danego problemu na podstawie historycznych danych.

2.1. Sieci neuronowe

Szczególnym rodzajem algorytmów uczenia maszynowego są sieci neuronowe. Zostały one stworzone na wzór neuronów obecnych w ludzkich organizmach. Składają się one z neuronów, które przetwarzają dane i przekazują do kolejnych neuronów razem tworzących warstwy. Każdy neuron posiada wagę, która jest ustalana w wyniku trenowania całej sieci przy pomocy danych dotyczących aktualnego problemu. Tak jak w przypadku ludzkiego mózgu, neurony w sieciach można komponować na wiele sposób. Do różnego rodzaju dziedzin, stosowane są wszelkiego rodzaju architektury odpowiednio dostosowane do problemów z nimi związanych. Głównym problemem wykorzystywania sieci neuronowych jest doprowadzenie stanu wag poszczególnych neuronów do takiego, który pozwoli na osiągnięcie jak najlepszych wyników w rozwiązywanym problemie. Jednakże sieci potrafią posiadać nawet miliony neuronów, z których każdy posiada wagę wymagającą odpowiedniego dopasowania. W tym celu stosuje się technikę obliczania błędu otrzymanych wyników, a następnie dopasowywania parametrów na jego podstawie. Błąd obliczany jest przy pomocy tak zwanej funkcji straty poprzez porównanie otrzymanych wartości z wartościami oczekiwanymi. Ze względu na konieczność tego porównania istotnym problemem dziedziny uczenia maszynowego jest uzależnienie od zbiorów danych. Istnieje wiele rodzajów funkcji straty, a do najpopularniejszych należą:

- średni błąd kwadratowy – mierzy on średnią kwadratową różnicę między przewidywanymi a rzeczywistymi wartościami. Jest bardzo popularny w przypadku zadań regresji polegających na przewidywaniu zbioru wartości,
- średni błąd bezwzględny – funkcja bardzo podobna do średniego błędu kwadratowego, ale zamiast kwadratu różnicy, oblicza on wartość bezwzględną różnicy między przewidywanymi a rzeczywistymi wartościami,
- entropia krzyżowa – niezwykle popularna funkcja straty w przypadku zadań dotyczących klasyfikacji. Można wyróżnić dwie jej odmiany: binarną oraz wieloklasową, z

których pierwsza jest wykorzystywana w przypadku jedynie dwóch klas wyjściowych, a druga dla szerszego zbioru,

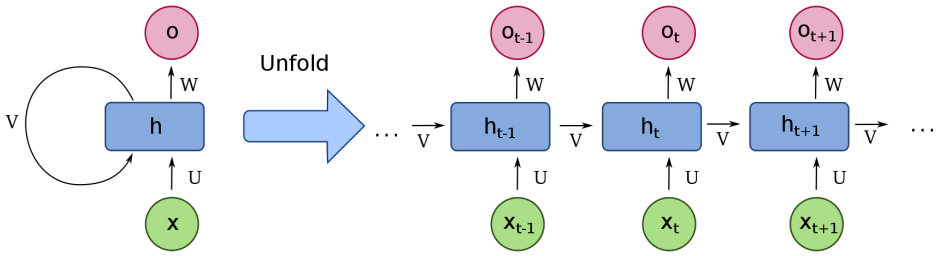
- Hinge loss – często używana w kontekście klasyfikacji binarnej, szczególnie w problemach związanych z maszynami wektorów nośnych [1]. Mierzy ona, jak bardzo model jest pewny poprawnej klasy dla danego przykładu, a jednocześnie penalizuje, gdy to przekonanie jest niewłaściwie duże.

Na podstawie tak otrzymywanych wartości funkcji straty, sieć neuronowa stara się ją zminimalizować, a co za tym idzie, dostosować swoje wagi do jak najlepszego rozwiązania. W tym celu najczęściej wykorzystywana jest metoda najszybszego spadku. Otrzymana wartość gradientu jest aplikowana do poszczególnych neuronów sieci, w celu dopasowania ich wag. Wielokrotne generowanie wyników, a następnie dostosowywanie wartości parametrów, w celu minimalizacji funkcji straty, pozwalają na dopasowanie się sieci neuronowej do danego problemu opisanego poprzez zbiór danych wejściowych oraz oczekiwanych. Ze względu na konieczność porównania wyników do danych oczekiwanych w wykorzystywaniu sieci neuronowych istotnym wyzwaniem jest generalizacja architektury, która pozwoliłaby na osiągnięcie jak najlepszych wyników bez względu na rodzaj danych wejściowych.

2.2. Przetwarzanie języka naturalnego

W dziedzinie przetwarzania języka naturalnego ogromną popularnością cieszą się algorytmy uczenia maszynowego, a w szczególności rekurencyjne sieci neuronowe. Klasyczne sieci neuronowe przetwarzają dane w sposób jednokierunkowy, co oznacza, że dane wejściowe są przetwarzane w sposób liniowy, a informacja o poprzednich wynikach nie jest brana pod uwagę przy przetwarzaniu kolejnych danych. Ze względu na naturę języka ludzkiego, pojedyncze słowa najczęściej nie niosą ze sobą całości przekazywanych informacji – konieczne jest wzięcie pod uwagę również kontekstu zdania, czy nawet całego tekstu. Taką możliwość dają rekurencyjne sieci neuronowe [2]. Przy ich pomocy dane wejściowe przetwarzane są w postaci szeregu, a informacja dotycząca poprzedniego elementu jest uwzględniania przy przetwarzaniu następnego elementu. Umożliwia to również nieograniczanie długości przetwarzanych danych – możliwe jest przetwarzanie kilkuwyrazowych tekstów lub nawet całych książek przy użyciu tej samej architektury sieci neuronowej. Jest to znaczące usprawnienie w porównaniu do klasycznych sieci neuronowych wykorzystujących warstwy w pełni połączone, jak również sieci splotowe w przypadku, których dane wejściowe muszą być tych samych rozmiarów. Dla dziedziny przetwarzania obrazów nie jest to istotny problem, ponieważ trywialne jest zwiększenie lub zmniejszenie rozdzielczości zdjęcia, natomiast w przypadku chęci zmniejszenia długości przetwarzanego tekstu bardzo wysoce prawdopodobne jest to, iż utracone zostałyby niezwykle istotne części przetwarzanych danych zaburzające całościowy przekaz. Schemat architektury rekurencyjnej sieci neuronowej został przedstawiony na rysunku 2.1. Główną jej różnicą w porównaniu do sieci jednokierunkowych jest wykorzystanie sprzężenia zwrotnego. Wartości wyjściowe z poprzednich obliczeń są wykorzystywane jako wartości wejściowe dla kolejnych obliczeń. Dzięki temu możliwe jest uwzględnienie kontekstu prze-

twarzanych danych, jak również nieograniczanie długości danych wejściowych – każdy kolejny element sekwencji jest interpretowany razem z wartością otrzymaną z poprzednich iteracji. W przypadku klasycznej sieci, aby przetworzyć dany szereg, uwzględniając poszczególne elementy, konieczne byłoby znanie długości sekwencji, w celu dostosowania ilości neuronów w architekturze. Takie rozwiązanie jest niepraktyczne w przypadku przetwarzania języka naturalnego, gdzie długość sekwencji jest zmienna, ponieważ każda zmiana architektury wymaga jej ponownego wytrenowania.



Rysunek 2.1. Architektura rekurencyjnej sieci neuronowej. Źródło: [3]

Rekurencyjna sieć neuronowa była jednym z pierwszych i najpopularniejszych rozwiązań wykorzystywanych przy przetwarzaniu języka naturalnego. Przy jej pomocy można rozwiązać skutecznie wiele zadań, jednakże nie jest wolna od wad. Największym problemem z nią związanym jest duża niestabilność objawiająca się poprzez eksplozję lub zanikanie gradientu. Spowodowane jest to tym, iż w obliczeniach sieci rekurencyjnej za każdym razem uwzględniane są wyniki z poprzedniego przetworzenia. W przypadku chęci przetwarzania długich tekstów, mnożenie kolejnych wartości gwałtownie zwiększa lub zmniejsza wartość gradientu, co znacząco zaburza ostateczny wynik (przykładowo wartość 1,01 podniesiona do tysięcznej potęgi daje wynik ponad 10 tysięcy). W celu zaadresowania tychże problemów powstały rozszerzenia sieci rekurencyjnej. Najpopularniejsze z nich to: Long Short-term Memory [4] oraz Gated Reccurent Unit [5]. Dzięki skomplikowaniu obliczeń oraz przekazywania danych w postaci kontekstu do kolejnych elementów szeregu, zjawisko eksplozji lub zanikania gradientu nie ma aż tak znaczącego wpływu na ostateczne rezultaty, ale mimo to nie jest ono całkowicie wyeliminowane.

2.3. Atencja

Atencja to kluczowy mechanizm w dziedzinie sieci neuronowych, który pierwszy raz został zaproponowany w 2014 roku [6], w celu poprawy działania modelu przetwarzania języka naturalnego skupiającego się na tłumaczeniu tekstu na różne języki. Mechanizm ten został zaproponowany, w celu usprawnienia danych kontekstowych, jakie każdy z elementów przetwarzanej sekwencji posiadał. Idea atencji opiera się na tym, że każda część danych wejściowych może mieć różne znaczenie dla wyniku. Zamiast traktować całą sekwencję jako jedną całość, atencja pozwala modelowi skupić się na istotnych fragmentach danych wejściowych w zależności od kontekstu. W praktyce atencja jest

zazwyczaj implementowana jako dodatkowy moduł w sieci neuronowej. Moduł ten oblicza wagi dla poszczególnych części danych wejściowych, które określają, jak bardzo te części powinny być brane pod uwagę przy generowaniu wyniku. Wagi te są obliczane na podstawie podobieństwa między danymi wejściowymi a aktualnym stanem modelu.

Szczególną odmianą mechanizmu atencji jest tak zwana atencja własna, która została zaproponowana po raz pierwszy przez autorów architektury Transformera [7], który okazał się rewolucyjnym rozwiązaniem w dziedzinie przetwarzania języka naturalnego, w głównej mierze dzięki zastosowaniu właśnie atencji własnej. Głównym aspektem odróżniającym ją od klasycznej wersji jest fakt, iż wagi konkretnych elementów obliczane są indywidualnie, biorąc pod uwagę wartości wszystkich pozostałych parametrów, jak i jego samego, skąd pochodzi nazwa samoistnej atencji. Pozwala to na uwzględnienie jedynie istotnych wartości w kontekście konkretnego elementu, a nie tak jak w przypadku klasycznego podejścia, jedynie w kontekście globalnym. Znacząco usprawnia to działanie modelu przetwarzającego teksty zawierające wiele elementów. Dodatkowym usprawnieniem często wykorzystywanym wraz z atencją własną jest mechanizm wielogłowej atencji. Polega on na równoległym wykorzystaniu kilku modułów atencji, a następnie połączeniu ich wyników w ostateczny rezultat. Takie podejście posiada wiele zalet, którymi są między innymi:

- wychwytywanie różnorodnych wzorców – każdy moduł atencji może skupić się na innych aspektach danych wejściowych, co pozwala na wykrycie bardziej złożonych wzorców,
- generalizacja – ze względu na równoległe wykorzystanie wielu modułów, model jest w stanie lepiej radzić sobie z różnorodnością danych wejściowych, co pozwala na osiąganie dobrych wyników w wielu różnorodnych zadaniach.

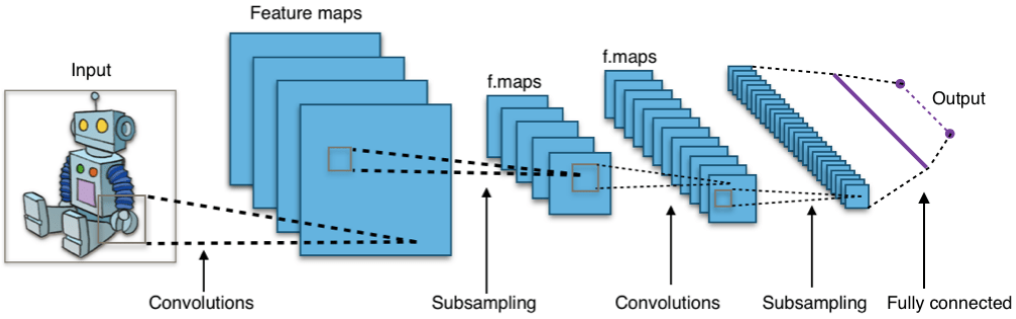
Mimo tak wielu zalet mechanizm atencji posiada również wiele wad, a są to między innymi:

- duża ilość parametrów – ze względu na konieczność uwzględnienia kontekstu każdego z elementów, moduł atencji musi zaalokować dużą ilość parametrów, co znacząco zwiększa złożoność modelu, a co za tym idzie czas potrzebny na przetworzenie danych, jak i ilość pamięci koniecznej do zaalokowania architektury. W przypadku wykorzystywania wielogłowej atencji problem ten dodatkowo się nasila ze względu na wykorzystywanie równoległych modułów.
- Wyniki otrzymane poprzez wykorzystanie modułu atencji mogą być trudne do wyjaśnienia i zinterpretowania – w przeciwieństwie do tradycyjnych sieci neuronowych, w których znaczenie każdej cechy wejściowej można łatwo określić poprzez sprawdzenie odpowiednich wag, mechanizm uwagi przypisuje znaczenie dynamicznie w czasie działania. Ten dynamiczny charakter utrudnia zapewnienie jasnych wyjaśnień procesu decyzyjnego modelu, utrudniając przejrzystość i interpretowalność.
- Duże ilości danych – ze względu na złożoność mechanizmu oraz dużą ilość parametrów konieczne jest wykorzystanie dużej ilości danych treningowych, aby osiągnąć zadowalające efekty. Istnieją modyfikacje mechanizmu atencji, a w szczególności architektury Transformera [8], którym udaje się osiągnąć zadowalające wyniki przy

wykorzystaniu mniejszej ilości danych, jednakże są to rozwiązania niestandardowe, które wymagają dodatkowych zabiegów w celu osiągnięcia zadowalających wyników, co pokazuje kolejną wadę wykorzystywania mechanizmu atencji – złożoność implementacyjną.

2.4. Wizja komputerowa

Przetwarzanie wszelkiego rodzaju obrazów cyfrowych w postaci zdjęć czy filmów jest niezwykle problematyczne, ponieważ pojedyncze zdjęcie wykonane w jakości HD posiada prawie milion pikseli. Dodatkowo biorąc pod uwagę wiele kombinacji, w których zdjęcie może być wykonane: różne ułożenie przedmiotu, kadr oświetlenie otoczenie, dane konieczne do przetworzenia oraz sformułowanie na ich podstawie algorytmów rozwiązujących dużą część problemów z nimi związanych jest praktycznie niemożliwe. Z tychże względów ogromną popularnością wśród rozwiązywania problemów związanych z wizją komputerową cieszą się algorytmy uczenia maszynowego, a w szczególności splotowe sieci neuronowe. Schemat podstawowej architektury takiej sieci służącej do klasyfikacji obrazu został przedstawiony na rysunku 2.2.



Rysunek 2.2. Architektura splotowej sieci neuronowej. Źródło: [9]

Charakteryzują się one różnymi rodzajami warstw. Do podstawowych typów warstw należą:

- warstwa splotowa – jest to podstawowa warstwa, która składa się z zestawu filtrów splotowych wykorzystujących operację splotu. Filtry te przesuwają się po obrazie wejściowym i wykrywają w nim wzorce o różnym stopniu skomplikowania.
- Warstwa aktywacji – wyniki operacji splotu są przetwarzane za pomocą funkcji aktywacji, takiej jak tangens hiperboliczny, co pozwala na generowanie tak zwanych map cech dla każdego filtra z warstwy splotowej.
- Warstwa grupująca – rozmiar otrzymanych map cech wygenerowanych przez warstwę splotową jest redukowany poprzez grupowanie wartości w oknie i zastępowanie ich pojedynczą wartością reprezentującą cechy w danym obszarze. Najczęściej stosowane są dwa rodzaje operacji grupujących: max-pooling i average-pooling.

Powyższe warstwy najczęściej występują wielokrotnie w obrębie jednej sieci, co pozwala na wydobywanie wzorców z przetwarzanego obrazu na różnym poziomie kontekstu. Dalsze przetwarzanie oraz grupowanie map cech obrazu daje możliwość uwzględnienia szerszego

kontekstu, jaki jest zawarty w obrazie niż jedynie w obrębie najbliższego sąsiedztwa danego piksela.

2.5. Mechanizm atencji w przetwarzaniu obrazów

2.6. Wstępne przetwarzanie danych

W procesie przygotowywania danych do przetwarzania przez sieci neuronowe konieczne jest dokładne ujednolicenie i przekształcenie danych w formę, która jest łatwo przyswajalna przez komputer. Dla obrazów cyfrowych, które są reprezentowane jako macierze kodów RGB, wystarcza zazwyczaj znormalizowanie wartości pikseli oraz ujednolicenie ich wymiarów w celu ułatwienia obliczeń. W przypadku danych o niestandardowym formacie pojawia się dodatkowy krok, jakim jest konwersja obrazów do formatu RGB. Częstą dodatkową techniką, która pozwala na zwalczanie problemów przetrenowania oraz małej ilości danych treningowych jest przycinanie obrazów. Stosuje się do tego kilka technik:

- przycinanie losowe,
- przycinanie z wykorzystaniem algorytmów wykrywania krawędzi.

Wybór odpowiedniej techniki w głównej mierze jest zależny od potrzeb wynikających z docelowego problemu, który ma zostać rozwiązany poprzez wykorzystanie sieci neuronowej. Przy pomocy tejże techniki możliwe jest również wielokrotne wykorzystanie tego samego zdjęcia jako zupełnie innego obrazu, ze względu na przycięcie i zawarcie na nim innych elementów, co znacząco zwiększa ilość dostępnych danych treningowych.

W przypadku wartości tekstowych, które są wartością wejściową całej architektury, proces przetwarzania jest bardziej skomplikowany. Tekst w języku naturalnym posiada różnorodność, obejmującą zmienność długości słów, różne końcówki, a także inne nieregularności. Aby uporządkować te rozbieżności, stosuje się kilka technik:

- normalizacja wielkości liter – wszystkie litery w tekście są zamieniane na małe litery.
- Usunięcie wszystkich znaków niebędących słowami takich jak kropki, przecinki, emotikony.
- Usunięcie słów znajdujących się na stop liście, czyli zbiorze słów o nikłym znaczeniu dla kontekstu zdania, takich jak spójniki, przyimki lub nawet często używane słowa.
- W przypadku chęci osiągnięcia znaczącej generalizacji często stosowane jest również odrzucenie rzadko występujących słów w celu zmniejszenia dostępnego słownika, a co za tym idzie, zwiększenia skuteczności uczenia.

Tak przetworzony tekst nadal jest w postaci słów i liter, co jest problematycznym formatem przetwarzania dla komputera – porządanymi danymi są macierze i wektory, którymi operują wszelkie algorytmy uczenia maszynowego. W celu osiągnięcia takowego formatu, stosuje się technikę zwaną tokenizacją, z której można wyodrębnić kilka różnych podejść. Do najpopularniejszych należą:

- tokenizacja znaków – jest najprostszą techniką polegającą na utworzeniu wektorów, gdzie pojedynczą wartością jest każdy znak przetwarzanego zdania. Takie podejście

znacząco zwiększa ilość przetwarzanych danych, co nie jest konieczne, ponieważ pojedyncze litery ze słów nie nie niosą ze sobą znaczących informacji – informacja zawarta jest w całości słowa, a nawet w kontekście kilku słów lub całego zdania. Dużym atutem tego podejścia jest brak konieczności stosowania pozostałych technik wstępnego przetwarzania tekstu, co znacząco upraszcza proces.

- tokenizacja słów – polega na wyodrębnieniu token na podstawie słów. To właśnie do tej techniki najczęściej stosowane są wcześniej wspomniane metody wstępnego przetwarzania tekstu. W odróżnieniu od tokenizacji znaków, gdzie liczba liter w słowniku jest znacząco ograniczona, ilość słów może osiągać ogromne wartości – Wielki słownik ortograficzny PWN zawiera 140 tysięcy haseł [10], dlatego konieczne jest zastosowanie pewnych ograniczeń tak jak wspomniane wcześniej odrzucanie rzadko występujących słów.
- tokenizacja digramowa – polega na znalezieniu par liter i traktowaniu ich jako pojedynczy token. Wykorzystanie tejże techniki pozwala na uwzględnienie o wiele szerszego kontekstu niż w podstawowej tokenizacji znaków jednocześnie nie ograniczając się do konkretnego słownika słów, co umożliwia generowanie danych spoza słownika.

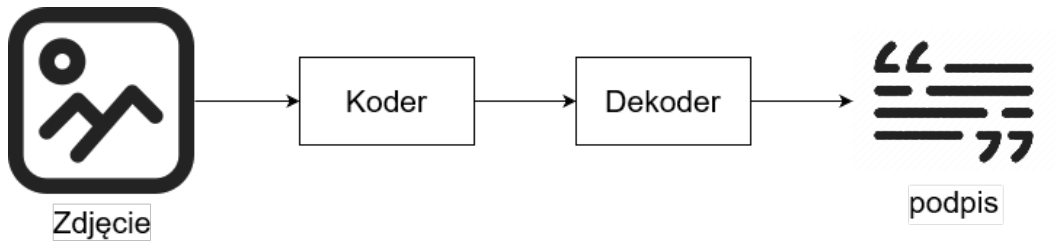
W przetwarzaniu języka dodatkową trudność wprowadza fakt, iż nie istnieje jeden uniwersalny język, a wiele popularnych języków posiada swoje własne specyficzne cechy, jak na przykład zupełnie inny alfabet, czy brak spacji między słowami. Rozwiązaniem tego problemu może być próba rozpoznania języka, a następnie zastosowanie odrębnej techniki przetwarzania dla konkretnej rodziny językowej.

3. Generowanie podpisów do obrazów – aktualne rozwiązania

W celu wygenerowania podpisu do obrazu przy użyciu algorytmów uczenia maszynowego wykorzystuje się sieci neuronowe. Sam proces generowania podpisów można rozbić na dwa etapy: przetwarzanie obrazu oraz generowanie tekstu. Z tego względu najpopularniejsze podejścia wykorzystują połączenia sieci wykorzystywanych do przetwarzania obrazów wraz z sieciami wykorzystywanymi do przetwarzania języka naturalnego. Takie podejście wyodrębnia dwa główne moduły w całej architekturze sieci:

- moduł kodujący obraz – jego danymi wejściowymi jest obraz w postaci macierzy pikseli, natomiast wyjściem jest wektor sprowadzony do postaci zgodnej z formatem wejściowym modułu dekodującego,
- moduł dekodujący zakodowany wcześniej obraz – wejściem tego modułu jest wyjście modułu kodującego, natomiast wyjściem jest ostatecznie wygenerowane zdanie.

Poglądowy schemat takowej architektury został przedstawiony na rysunku 3.1.



Rysunek 3.1. Typowa architektura generatora podpisów. Opracowanie własne

3.1. Moduły kodujące

Ze względu na niezwykle dużą skuteczność, jak również popularność, najczęściej wykorzystywaną siecią neuronową jako moduł kodujący jest splotowa sieć neuronowa. Główną jej modyfikacją, w celu dostosowania do problemu, jakim jest generowanie podpisów, jest dopasowanie ostatnich warstw w taki sposób, aby ostateczny wektor wyjściowy był kompatybilny z typowym formatem danych wejściowych sieci wykorzystywanych w dziedzinie przetwarzania języka naturalnego. Ze względu na specyfikę rozwiązywanego problemu częstym wyborem są te typy sieci, które niezwykle dobrze radzą sobie z generalizacją rozwiązywanego problemu. Jest to istotne w przypadku chęci generowania podpisów, ponieważ umożliwia generowanie znacznie barwniejszych i niosących więcej informacji podpisów.

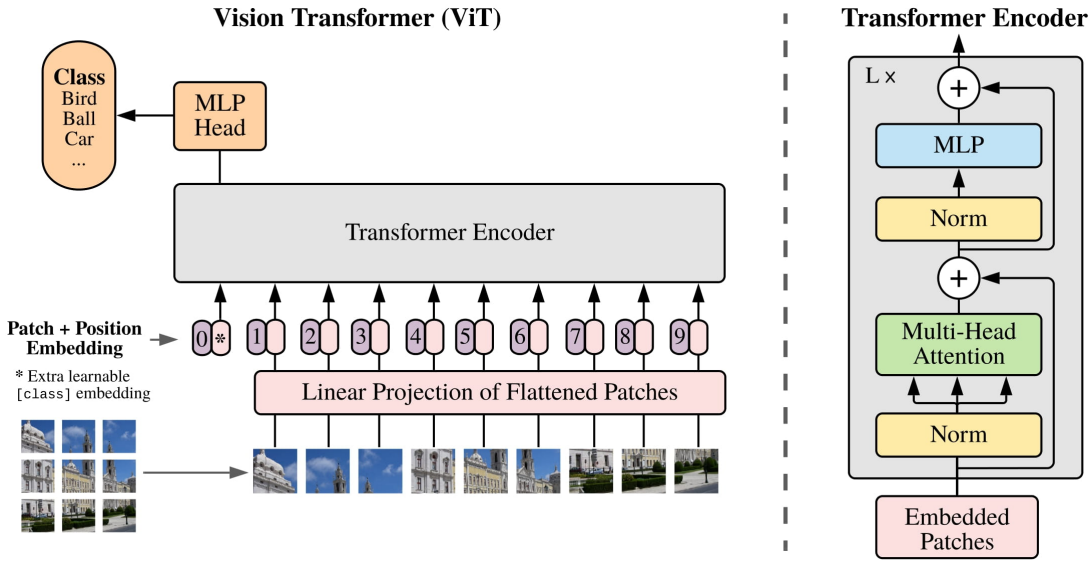
3.1.1. Bardzo głębokie sieci splotowe

Jednym z częściej wykorzystywanych rodzajów sieci splotowej są głębokie sieci neuronowe. Charakteryzują się one dużą liczbą warstw splotowych oraz małymi wielkościami filtrów. Aktualnie istnieje wiele architektur wykorzystujących takie podejście. Jedną z pierwszych była sieć VGGNet [11]. Dzięki zastosowaniu dużej liczby warstw splotowych możliwe było klasyfikowanie obrazów o dużej rozdzielczości z bardzo wysoką skuteczno-

ścią. Dużym minusem tego rodzaju sieci są wymagane zasoby. Ze względu na dużą ilość warstw konieczna jest alokacja dużej ilości pamięci w jednostce GPU, a samo przetworzenie danych przez sieć wymaga wielu obliczeń.

3.1.2. Transformer wizyjny

Transformer wizyjny [12] to model przetwarzania obrazów, który różni się od tradycyjnych architektur zawierających sieci splotowe. Opiera się on na transformerze znanym głównie z zastosowań w przetwarzaniu języka naturalnego. Przy jego pomocy obraz jest przetwarzany poprzez podzielenie na nienachodzące się bloki, a każdy blok reprezentowany jest jako token, podobnie jak słowa w zdaniu. Tak wygenerowane elementy są traktowane jako sekwencja wejściowa dla modelu transformera. Poglądowy diagram przedstawiający architekturę transformera wizyjnego został przedstawiony na rysunku 3.2.



Rysunek 3.2. Diagram architektury transformera wizyjnego. Źródło: [12]

Dane wyjściowe z kodera transformera przetwarzane są w bardzo podobny sposób jak w przypadku danych wyjściowych głębokich warstw sieci splotowych – przy pomocy w pełni połączonych warstw, wektor wyjściowy jest sprowadzany do postaci wektora zawierającego odpowiednie klasy, co zostało przedstawione na diagramie przy pomocy bloku „MLP Head" oraz „Class". W dziedzinie generowania podpisów do obrazków wspomniane bloki najczęściej zostają pominięte, dzięki czemu danymi wyjściowymi jest wektor zawierający informacje o obrazie, który jest wykorzystywany jako wejście dla modułu dekodującego.

3.2. Moduły dekodujące

Dzięki sprowadzeniu obrazu do formatu zwykłego wektora poprzez moduł kodujący możliwe jest wykorzystanie większości rozwiązań dedykowanych do zwykłych zadań przetwarzania języka naturalnego. Jako analogiczną dziedzinę można tutaj wyróżnić zagadnienie związane z generowaniem odpowiedzi na pytania – główną różnicą w przypadku generowania podpisów jest sprowadzenie obrazu do wartości możliwej do przetworzenia

przez moduł dekodujący. W przypadku zagadnień związanych z generowaniem odpowiedzi, przetwarzany jest tekst, natomiast w przypadku generowania podpisów są to obrazy. Dalsza część architektury, czyli moduł dekodujący, jest analogiczna w obu przypadkach. Częstym wyborem jako moduł dekodujący jest podstawowa rekurencyjna sieć neuronowa – RNN [2]. W przypadku takiej sieci wektor będący wyjściem modułu kodującego służy jako pierwszy element wejściowy modułu dekodującego. Naturalnie możliwe jest wykorzystywanie bardziej zaawansowanych odmian sieci rekurencyjnej jak LSTM [4] oraz GRU [5]. Również wykorzystywane są moduły atencji [6] [13], w których przypadku szczególnie użyteczne są głębokie sieci neuronowe. Wyniki z poprzednich warstw splotowych służą jako kontekst, z którego może korzystać moduł atencji.

3.2.1. Transformer

Tak jak w przypadku pozostałych problemów związanych z przetwarzaniem języka naturalnego, również w kontekście generowania podpisów do obrazów, popularnością cieszy się sieć Transformer [7]. Główną różnicą w porównaniu do klasycznych sieci rekurencyjnych jest przetwarzanie przez transformer sekwencji w całości, a nie pojedynczych jej elementów. Takie działanie jest możliwe dzięki zastosowaniu modułu atencji oraz zakodowania pozycyjnego, czyli dodania informacji o pozycji elementu w sekwencji do jego wartości. Sama atencja pozwala na określenie znaczenia danych elementów sekwencji w kontekście jej całości. Jest to znacząca przewaga nad sieciami rekurencyjnymi, które biorą pod uwagę jedynie aktualnie przetwarzany element oraz dane wygenerowane na podstawie wszystkich wcześniej przetworzonych elementów. Wykorzystanie tych rozwiązań również niweluje całkowicie problem eksplozji lub zanikania gradientu, który występuje w przypadku sieci rekurencyjnych. Autorzy transformera nie wymyślili jako pierwsi metody atencji, jednakże znacząco ją usprawnili poprzez zastosowanie atencji własnej, która uwzględniała wartości atencji nie globalnie, a w kontekście pojedynczych elementów do pozostałych wartości sekwencji, co znacząco wpłynęło na ogólną wydajność architektury. Dodatkowo zastosowali kilka modułów atencji pracujących równolegle nazywanych wielogłową atencją. Taki mechanizm pozwalał na dużo większe możliwości generalizacji otrzymywanych wyników, ze względu na wiele źródeł danych pochodzących z poszczególnych modułów, które następnie są łączone w ostateczny wynik.

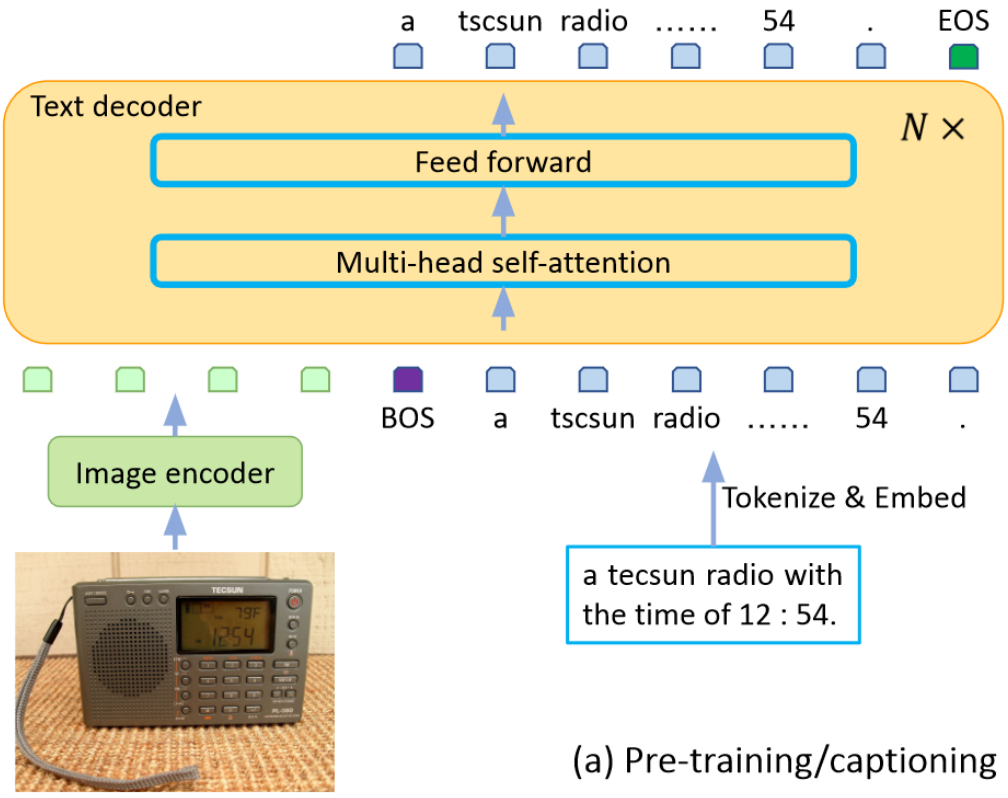
3.3. Neuronowy generator podpisów obrazów – NIC

Architektura NIC [14] (ang. Neural Image Caption) wykorzystuje połączenie splotowej sieci neuronowej [15] oraz sieć LSTM, w celu generowania podpisów. Osiąga ona wartość BLEU-4 na poziomie 0,27 dla zbioru MS COCO [16]. Jest to stosunkowo stare rozwiązanie pochodzące z 2015 roku, ale mimo to skuteczność tejże architektury nie odbiega znacząco od wyników aktualnych rozwiązań. Samo połączenie splotowej sieci neuronowej wraz z siecią rekurencyjną w architekturze NIC zostało zaimplementowane w nieskomplikowany sposób: pierwszym elementem wejściowym sieci LSTM jest zakodowany obraz otrzymany poprzez przetworzenie go przez sieć splotową. Autorzy pracy zastosowali technikę transferu wiedzy i ograniczyli trenowanie sieci jedynie do modułu dekodującego

– obrazy zostały zakodowane tylko raz, po czym posłużyły za dane treningowe dla sieci LSTM. Przy takim podejściu kluczowy był odpowiedni wybór modułu kodującego, dlatego autorzy zdecydowali się na wybór modelu osiągającego wysokie wyniki w wielu różnych dziedzinach przetwarzania obrazów. Taki model można uznać za wysoce generalizujący, a co za tym idzie, posiadający lepsze predyspozycje adaptacji do innych zadań, co jest idealnym przypadkiem wykorzystania jako moduł w sieci generowania podpisów, czyli zadania odbiegającego znacząco od klasycznych zastosowań sieci splotowych.

3.4. Generatywny transformator obrazu na tekst – GIT

Bardziej rozbudowana architektura GIT [17] (ang. Generative Image-to-text Transformer), w porównaniu do architektury NIC, na zbiorze danych MS COCO osiągnęła wartość metryki BLEU-4 równą 0,43 – wynik o 50% lepszy. Wykorzystuje ona model Florence [18] jako moduł kodujący wraz z klasycznym Transformerem wykorzystanym jako moduł dekodujący. Rozwiązanie to pochodzi z 2022 roku i aktualnie osiąga jedne z najlepszych wyników w dziedzinie generowania podpisów do obrazków. Autorzy główną uwagę skupili na skonstruowaniu prostej architektury wykorzystującej tylko jeden moduł kodujący i dekodujący – wiele architektur w odróżnieniu od modelu GIT wykorzystuje kilka modułów pracujących równolegle. Dane w pierwszej kolejności przetwarzane są przez moduł kodujący, którego wynikiem jest macierz zawierające informacje o szczegółach zawartych w obrazie. Następnie tak otrzymane dane przetwarzane są przez kilka modułów transformera – schemat przetwarzania danych został przedstawiony na rysunku 3.3.

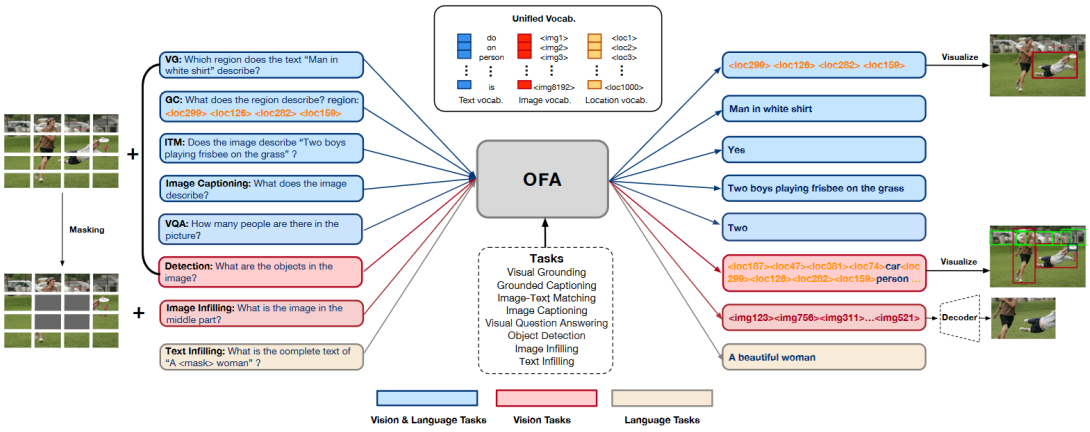


Rysunek 3.3. Diagram przedstawiający sposób przetwarzania danych przez model GIT. Źródło: [17]

Tak dobre wyniki zostały osiągnięte przez autorów między innymi dzięki dobraniu odpowiedniego modelu jako modułów kodującego. Architektura Florence została zaprojektowana z myślą o generalizacji przetwarzania obrazów, w celu umożliwienia zastosowania jej do wielu różnych zadań z dziedziny wizji komputerowej, co jest niezwykle istotne w przypadku generowania podpisów do obrazów. Również bardzo istotnym aspektem było wykorzystanie ogromnego zbioru danych, w celu wstępnego wytrenowania modułu kodującego – liczył on około 800 tysięcy zdjęć, które pochodziły z wielu różnych zbiorów danych [19] [20] [21] [22] [23] w tym między innymi zbiór MSCOCO [16]. Warto również wspomnieć, iż model GIT został stworzony z myślą o rozwiązywaniu nie tylko problemu generowania podpisów do obrazków, ale również odpowiadania na pytania w odniesieniu do zdjęć oraz generowania opisów do obrazów wideo.

3.5. Raz na zawsze – OFA

Architektura OFA [24] (ang. Once for all) została stworzona z myślą o wielozadaniowości, dzięki czemu jest w stanie rozwiązać wiele zadań związanych z przetwarzaniem obrazów cyfrowych, w tym generowania podpisów. Model ten nie jest ograniczony jedynie do przetwarzania obrazów, ponieważ opiera się on na metodzie seq2seq, polegającej na przetwarzaniu pewnej sekwencji do innej sekwencji. Jego danymi wejściowymi są dane językowe, którym może towarzyszyć obraz, z tego względu oprócz macierzy pikseli danego obrazka konieczne jest również podanie tekstu w postaci formuły informującej architekturę o konieczności wygenerowania podpisu jak na przykład „co opisuje obraz?”. Wszelkie typy zadań rozwiązywane przez architekturę OFA zostały zobrazowane na rysunku 3.4 stworzonym przez autorów sieci.



Rysunek 3.4. Diagram przedstawiający możliwe typy zadań rozwiązywane przez architekturę OFA. Źródło: [24]

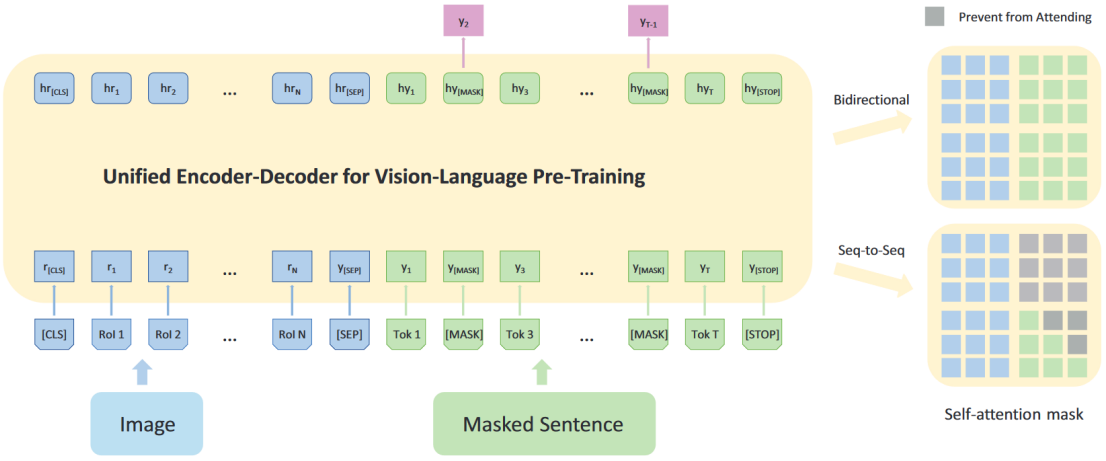
Architektura osiągnęła skuteczność na poziomie 0,44 dla metryki BLEU-4 na zbiorze danych MS COCO, co jest wynikiem porównywalnym do architektury GIT. Niemniej jednak wynik ten jest bardziej imponujący niż w przypadku modelu GIT, ponieważ architektura OFA jest w stanie rozwiązać wiele zadań związanych nie tylko z przetwarzaniem obrazów bez konieczności ponownego dostrajania modelu.

3.6. Ujednolicone wstępne uczenie modeli generowania podpisów – VLP

Architektura VLP [25] (ang. Vision-language Pre-training model) różni się tym od pozostałych rozwiązań, iż nie składa się z oddzielonych od siebie modułu kodującego oraz dekodującego – przetwarzanie obrazu oraz generowanie tekstu odbywa się w ramach tej samej jednostki wykorzystującej zmodyfikowaną architekturę transformera. Dodatkowo architektura została stworzona z myślą o wykorzystywaniu jej do różnego rodzaju zadań. Sami autorzy wyodrębnili dwa: generowanie podpisów oraz odpowiadanie na pytania w odniesieniu do obrazu. Z tego względu samo uczenie architektury przebiegało w trzech etapach:

- przetworzenie obrazu przy pomocy sieci splotowej rozpoznającej obiekty znajdujące się na obrazie.
- Zakodowanie zdania wraz z obrazem w postaci wektora, który posłużył jako wartość wejściowa dla architektury transformera.
- Dostrojenie architektury transformera w celu dostosowania jej do konkretnego zadania.

Mimo iż autorzy zastosowali technikę przetwarzania jednocześnie zdania oraz obrazu, konieczne było w pierwszej kolejności zakodowanie danych wizualnych, co zostało wykonane przy pomocy sieci splotowej – diagram przedstawiający architekturę wykorzystaną w przypadku uczenia modelu został przedstawiony na rysunku3.5.



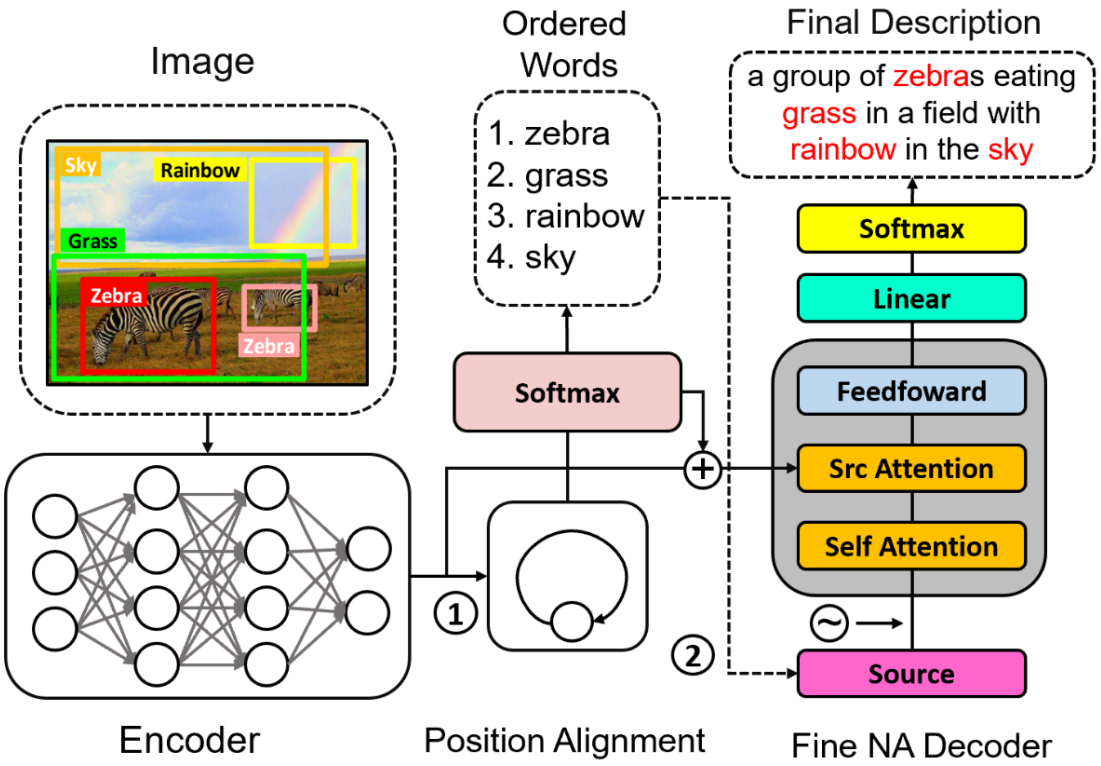
Rysunek 3.5. Diagram przedstawiający architekturę modelu VLP w trakcie jegouczenia. Źródło: [25]

Architektura osiągnęła skuteczność na poziomie 0,39 dla metryki BLEU-4 na zbiorze MS COCO. Jest to wynik niższy niż osiągnięty przez architektury GIT oraz OFA, jednakże różnica nie jest drastyczna.

3.7. Szybkie generowanie podpisów obrazów z wyrównaniem pozycji – FNIC

Architektura FNIC [26] (ang. Fast Neural Image Caption) wyróżnia się na tle pozostałych rozwiązań niewykorzystaniem klasycznego podejścia w przetwarzaniu języka naturalnego,

które oparte jest na sieci rekurencyjnej, czyli przetwarzaniu kolejnych elementów sekwencji. Takie podejście jest niezwykle skuteczne, ponieważ pozwala na generowanie kolejnych elementów z uwzględnieniem informacji o pozostałych już wygenerowanych pozycjach, czy też całej sekwencji w przypadku wykorzystywania modułu atencji. Jednakże wiążę się to z pojedynczym przetwarzaniem każdego z elementów, a co za tym idzie, znacząco wpływa na czas potrzebny do takiego przetworzenia. Architektura FNIC, w celu przyspieszenia generowania podpisów, wykorzystuje technikę zwaną wyrównaniem pozycji. Schemat przedstawiający architekturę modelu FNIC został przedstawiony na rysunku 3.6.



Rysunek 3.6. Diagram przedstawiający architekturę modelu FNIC. Źródło: [26]

W pierwszej kolejności obraz przetwarzany jest przy pomocy sieci splotowej do postaci wektora, co jest najpopularniejszą metodą przetworzenia obrazu do postaci wejściowej do dalszej analizy. Następnie informacje o obrazie są przetwarzane przez sieć rekurencyjną GRU, w celu wygenerowania pojedynczych słów, które mogą opisywać obraz – głównym celem tego modułu jest to, aby wygenerowane słowa były ułożone w odpowiedniej kolejności względem sensu zdania oraz docelowego opisu. Następnie każde ze słów przetwarzane jest równolegle – w ten sposób wygenerowane części zdań łączone są w ostateczny wynik. Tenże zabieg drastycznie przyspiesza generowanie podpisów, ponieważ każde słowo jest generowane równolegle, a nie pojedynczo. Jednakże kosztem jest dość spory spadek skuteczności danej architektury. Wyniki otrzymane przez model FNIC są porównywalne do klasycznych rozwiązań wykorzystujących moduł kodujący w postaci sieci splotowej wraz z modułem dekodującym w postaci sieci rekurencyjnej przy jednoczesnym kilkukrotnie krótszym czasie potrzebnym na generowanie podpisów. Należy tutaj również

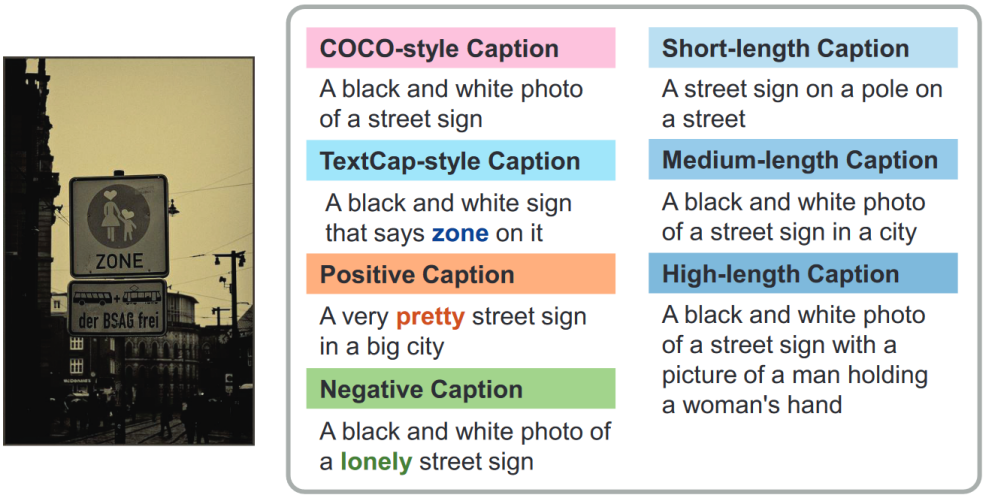
zauważyć, iż przyspieszenie najprawdopodobniej wiążę się z koniecznością wykorzystania o wiele większych zasobów komputerowych – równoległe przetwarzanie wielu elementów wymaga o wiele większej mocy obliczeniowej. Niestety autorzy pracy nie uwzględnili informacji o wykorzystywanych zasobach, a jedynie o relatywnym przyspieszeniu generowania podpisów w porównaniu do innych rozwiązań.

3.8. Kontrolowane podpisy obrazów przy pomocy podpowiedzi – ConCap

Architektura ConCap [27] (ang. Controllable Captioner) podejmuje problem, jakim jest generowanie podpisów do obrazków o różnym charakterze. W większości rozwiązań proponowanych w dziedzinie generowania podpisów do obrazów otrzymywane opisy niosą za sobą proste informacje o tym, co przedstawia dany obrazek. Przykład takich tekstów został przedstawiony na rysunku 3.7. Często w przypadku generowania podpisu użytkownik ma konkretny cel, na przykład chce opisać, co się na nim znajduje albo jakie emocje przedstawia.

Rysunek 3.7. Przykładowe obrazy wraz z prostymi podpisami. Opracowanie własne.

Przedstawione powyżej opisy jasno wyrażają, co znajduje się na zdjęciach, ale brakuje w nich informacji np. o emocjach, otoczeniu itp. W głównej mierze opis sprowadza się do zidentyfikowania obiektów oraz czynności, jakie wykonują. Architektura ConCap pozwala na dodanie wraz ze zdjęciem zdania opisującego, jakiego tekstu byśmy oczekiwali, na przykład pozytywnego, czy też negatywnego. Przykłady różnego rodzaju opisów zostały przedstawione na rysunku 3.8



Rysunek 3.8. Różnego rodzaju opisy wygenerowane przez model ConCap. Źródło: [27]

Architektura ConCap składa się z między innymi transformera wizyjnego oraz sieci BERT [28]. Niestety autorzy nie wskazali, jaką wydajność osiąga ich rozwiązanie, jednakże można przypuszczać, iż ze względu na wykorzystanie kilku niezwykle rozbudowanych oraz obciążających obliczeniowo architektur, użycie tego rozwiązania wymaga sporych zasobów komputerowych. Sami autorzy w celach uczenia architektury wykorzystali 32 karty graficzne Nvidia V100, które są specjalistycznymi jednostkami obliczeniowymi. Skuteczność architektury waha się w zależności od rodzaju opisów, jakie były generowane, co jest dość zrozumiałe, ponieważ opisy pochodzące ze zbioru MS COCO są proste i podobieństwo wygenerowanych wartości w przypadku chęci wygenerowania na przykład pozytywnych opisów może znacznie bardziej się różnić niż w przypadku klasycznej generacji. Niemniej jednak architektura dla zbioru MS COCO generując proste opisy, osiągnęła wartość metryki BLEU-4 równą 0,41, co jest wynikiem porównywalnym do pozostałych architektur jak OFA czy GIT. W przypadku generowania długich opisów architektura osiągnęła najsłabszy wynik wynoszący 0,27, co i tak jest niezwykle dobrym rezultatem.

3.9. Wydajność aktualnych rozwiązań

Publikowanych jest coraz więcej artykułów dotyczących uczenia maszynowego, jednakże stosunkowo mniej publikowanych jest treści podejmujących problem, jakim są wysokie wymagania sprzętowe potrzebne do stworzenia takiego rozwiązania, jak również jego wykorzystywania. Najczęściej w przypadku opisywania autorskiej architektury sieci neuronowej, autorzy podają ramy czasowe poświęcone na trenowanie architektury wraz z nazwą wykorzystywanej jednostki GPU. W większości wcześniej przedstawionych rozwiązań takie właśnie podejście zastosowali autorzy. Nawet w przypadku rozwiązania FNIC, którego celem było zwiększenie wydajności generowania podpisów, autorzy ograniczyli się jedynie do podania wyników względnych poprzez porównanie, w jakim stopniu ich rozwiązanie działało szybciej od pozostałych architektur. Często wyniki te otrzymywane są przy pomocy wielu zaawansowanych jednostek GPU, które są ogólnodostępne,

jednakże koszty ich wykorzystywania są znacząco większe, porównując do jednostek obliczeniowych znajdujących się w komputerach osobistych. Konkretne informacje o długości trenowania danej architektury, czy też czasu potrzebnego na wygenerowanie wyników, w trakcie wykorzystywania rozwiązania w docelowych warunkach, najczęściej nie zostają podane przez autorów i tak również było we wszystkich przykładach wcześniej wspomnianych w niniejszej pracy.

Tak konkretne dane pojawiają się w artykułach badających wydajności już opracowanych rozwiązań. Przykładem publikacji, które poruszają problem wysokich wymagań sprzętowych rozwiązań dotyczących neuronowych sieci splotowych oraz rekurencyjnych są:

- Analiza wydajności splotowych sieci neuronowych bazujących na GPU [29],
- Optymalizacja wydajności rekurencyjnych sieci neuronowych wykorzystujących GPU [30].

Niestety ze względu na o wiele mniejszą popularność dziedziny generowania podpisów do obrazków, ilość publikacji dotyczących konkretnie tego problemu jest stosunkowo mniejsza. Przykładem rozwiązania pochylającego się nad problemami wydajności wykorzystywanych rozwiązań w dziedzinie generowania podpisów do obrazków jest publikacja [31] skupiająca się na problemie dużych wymagań pamięciowych sieci dotyczących samego przechowywania zapisanych parametrów sieci neuronowych na urządzenia, co jest szczególnie problematyczne w przypadku urządzeń mobilnych. Wspomniany artykuł niestety nie porusza kwestii dotyczących wydajności czasowej zastosowanego rozwiązania oraz wykorzystywania pamięci RAM w trakcie użytkowania architektury.

Innym przykładem publikacji poruszającej problem optymalizacji sieci neuronowych wykorzystywanych do generowania podpisów jest rozwiązanie [32] wykorzystujące moduł określający rodzaj części mowy, jaki reprezentuje wygenerowane słowo. Autorzy dużą uwagę skupili na wydajności zaproponowanego rozwiązania, jednakże mimo prezentacji wyników długość przetwarzania pojedynczego zdjęcia, nie przedstawili oni danych dotyczących wykorzystywanych zasobów komputerowych oraz jakie konkretnie jednostki obliczeniowe zostały przez nich wykorzystane.

Publikacja Harsita Rampala i Amana Mohanty [33] jest najlepszym przykładem publikacji z dziedziny generowania podpisów do obrazków skupiającej się na wydajności wykorzystywanych rozwiązań. Autorzy, w celu optymalizacji wykorzystywanych zasobów sieci rekurencyjnych oraz splotowych zastosowali wiele technik kompresji modeli, które pozwoliły na znaczne zredukowanie wymagań pamięciowych badanych rozwiązań oraz usprawnienie czasu potrzebnego na ich przetwarzanie. Jednak przedstawione dane zostały ograniczone jedynie do długości czasu inferencji poszczególnych rozwiązań oraz zabrakło informacji o wykorzystanych jednostkach obliczeniowych.

Ze względu na to, iż generowanie podpisów do obrazów łączy dwie dziedziny – wizja komputerowa oraz przetwarzanie języka naturalnego – możliwe jest oszacowanie, w jakich proporcjach zmieniać będzie się sama wydajność wykorzystywanych poszczególnych modułów kodujących oraz dekodujących w architekturze poprzez połączenie czasów jednego modułu wraz z czasami drugiego modułu otrzymanymi z istniejących badaniach.

Mimo wszystko takich wyników nie można uznać za dokładne ze względu na konieczność wprowadzenia modyfikacji w konkretnych architekturach modułów, jak również zupełnie różne środowiska testowe wykorzystywane w konkretnych badaniach. Dodatkowym problemem jest brak informacji o skuteczności danej architektur w kontekście zastosowanych zasobów, co jest niezwykle istotnym aspektem, ponieważ w przypadku osiągnięcie niezwykle wydajnej architektury jednocześnie może ona osiągać o wiele słabszą dokładność generowanych wyników.

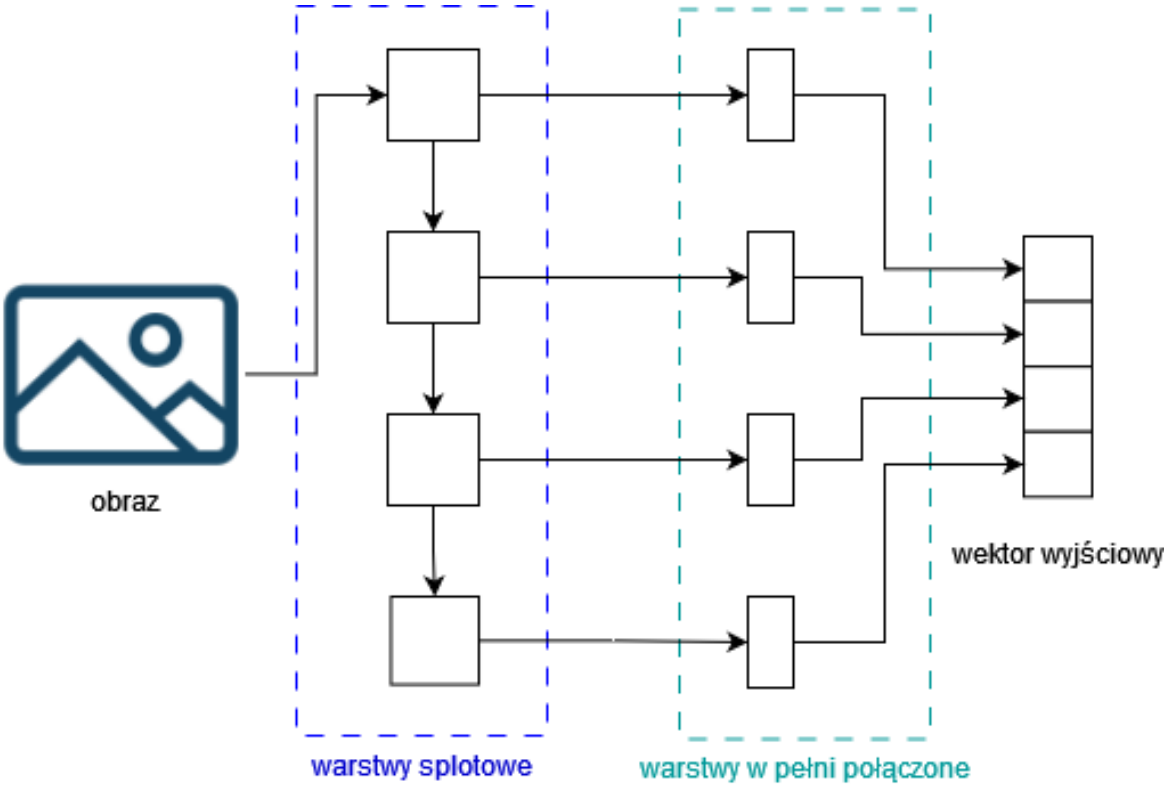
4. Opis zastosowanego rozwiązania

W przypadku wykorzystywania sieci splotowych wraz z sieciami rekurencyjnymi, w celu stworzenia architektury zajmującej się generowaniem podpisów do obrazów, najczęstszym typem połączenia tychże dwóch modułów jest wykorzystanie wartości wyjściowej sieci splotowej jako pierwszego elementu sekwencji przetwarzanej przez sieć rekurencyjną. Jest to niezwykle proste rozwiązanie, przy pomocy którego możliwe jest osiągnięcie sensownych rezultatów. Głównym problemem takiego podejścia jest uwzględnienie jedynie informacji końcowej generowanej przez sieć splotową. Pomijany jest ogrom danych przetwarzanych w warstwach pośrednich. Sieć rekurencyjna mogłaby znacząco skorzystać na uwzględnieniu wartości przetwarzanych w pośrednich warstwach splotowych, których wynikiem są tak zwane mapy szczegółów. Zawierają one kontekst informacji znajdujących się na obrazie o różnym poziomie uszczegółowienia.

W celu zaadresowania tego problemu oraz wykorzystania informacji, jakie generuje sieć splotowa w wyższych warstwach, w niniejszej pracy zostało wykorzystane zmodyfikowane rozwiązanie łączące moduły kodujący i dekodujący z uwzględnieniem danych z pośrednich warstw splotowych modułów kodujących.

Macierze generowane przez pośrednie warstwy splotowe zostały wyciągnięte, spłaszczone, zredukowane i połączone w jeden wektor, co pozwoliło na wykorzystanie go jako wartości wejściowej dla sieci rekurencyjnej. Takie rozwiązanie umożliwia uwzględnienie wartościowych danych, które mogą mieć istotne znaczenie dla ostatecznego wyniku.

Wektory pochodzące z warstw splotowych, otrzymane poprzez ich spłaszczenie są zbyt długie, by mogły być wykorzystane jako wartości wejściowe dla sieci rekurencyjnej, dlatego konieczna jest ich redukcja. W tym celu wykorzystane zostały warstwy w pełni połączone, które również są wykorzystywane w ostatnich warstwach sieci splotowej. Taki zabieg pozwala na otrzymanie ostatecznego wektora o pożądanej długości z zachowaniem istotnych informacji pochodzących z map szczegółów. Dużym minusem takiego podejścia jest konieczność dodatkowych obliczeń przeprowadzanych przez warstwy w pełni połączone. W celu ograniczenia zwiększania zużycia zasobów obliczeniowych, dla każdej mapy szczegółów została wykorzystana tylko jedna warstwa w pełni połączona. Takie podejście może nie być optymalny – wektory map szczegółów są znaczącej długości, a przy redukcji poprzez tylko jedną warstwę w pełni połączoną może dojść do utraty dużej części informacji, jednakże znacząco ograniczy to zużycie zasobów obliczeniowych. Dodatkowo takie rozwiązanie pozwoli sprawdzić, w jakim stopniu pojedyncze warstwy w pełni połączone wpływają na wydajność generowania podpisów, jak również skuteczność. Schemat modułu kodującego został przedstawiony na rysunku 4.1.



Rysunek 4.1. Diagram przedstawiający architekturę modułu kodującego. Opracowanie własne

Każdy wektor wyjściowy warstwy splotowej został zredukowany do takiego samego rozmiaru, który wyniósł 1024. Następnie zostały one połączone w jeden wektor, który posłużył jako wartość wejściowa modułu dekodującego. Schemat przedstawia jedynie poglądowy sposób działaniu modułu kodującego. Ze względu na wykorzystanie wielu różnych sieci splotowych w ramach modułu kodującego, mapy szczegółów zostały wydobyte na różnych ich poziomach.

4.1. AlexNet

Sieć AlexNet [34] składa się z kilku warstw zawierających następujące elementy:

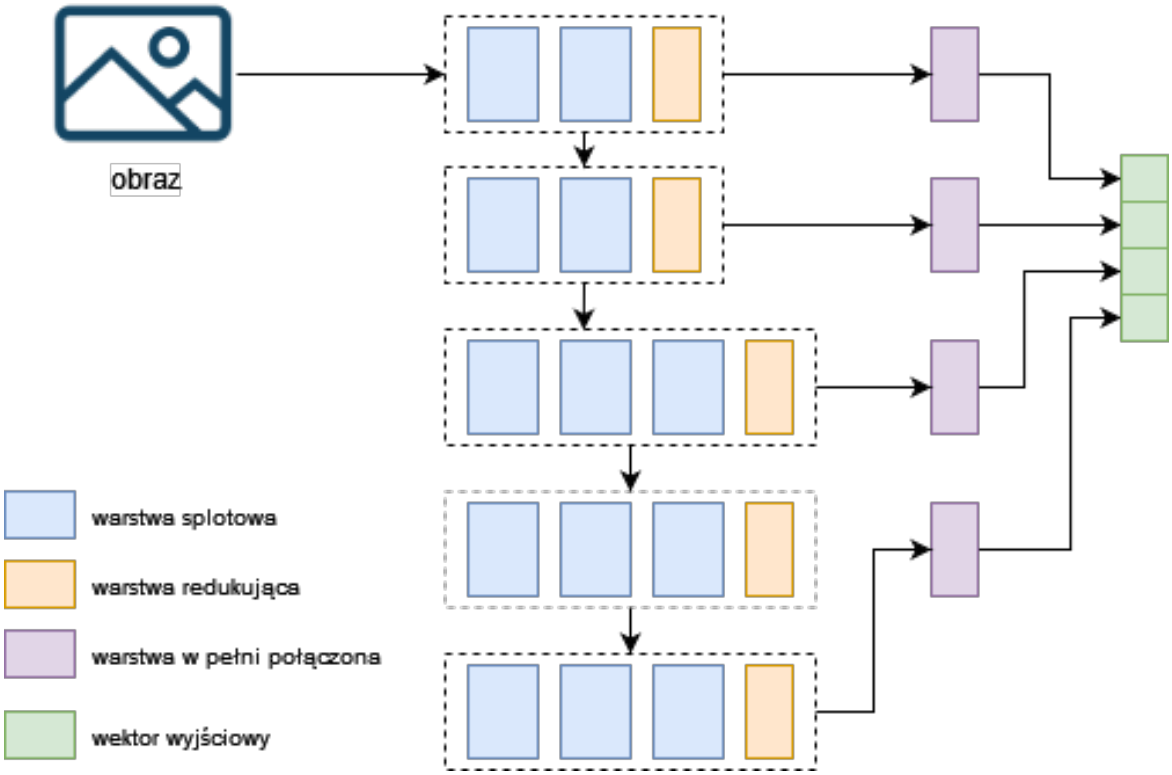
- warstwa splotowa,
- funkcja aktywacji ReLU,
- warstwa redukująca.

Warstwa redukująca następnie jest połączona ponownie z warstwą splotową. Cała sieć zawiera pięć takich bloków. Wartości wyjściowe z warstw redukujących zostały wykorzystane jako wektory wejściowe dla warstw w pełni połączonych, których wyjście tworzy ostateczny wektor wyjściowy. Jako że ostateczny wektor składa się jedynie z czterech wektorów składowych, przedostatni blok został pominięty.

4.2. VGG

Sieć VGG [11] posiada nieco bardziej rozbudowaną architekturę niż sieć AlexNet. Podstawowe bloki również składają się z warstwy splotowej, funkcji aktywacji ReLU oraz

warstwy redukcyjnej, jednakże główną różnicą jest zastosowanie większej ilości warstw splotowych w późniejszych poziomach całej sieci, co można zaobserwować na rysunku 4.2.



Rysunek 4.2. Diagram przedstawiający architekturę sieci VGG. Opracowanie własne.

Pierwsze dwa bloki są złożone z dwóch warstw splotowych, natomiast kolejne z trzech. Z takiej konfiguracji można wyodrębnić pięć głównych bloków, tak samo, jak w przypadku sieci AlexNet. W celu utworzenia ostatecznego wektora wyjściowego pominięty został przedostatni blok.

4.3. ResNet

Architektura ResNet [35] znacząco odbiega swoją budową od poprzednich sieci. Posiada ona cztery główne bloki składające się z pomniejszych bloków, które z kolei zawierają trzy warstwy splotowe wraz z warstwami normalizującymi. W celu utworzenia wektora wyjściowego nie było konieczności pomijania żadnego bloku ze względu na to, iż architektura składa się z czterech zasadniczych bloków. Pozwoliło to na utworzenie ostatecznego wektora składającego się z danych pochodzących z czterech różnych warstw sieci ResNet.

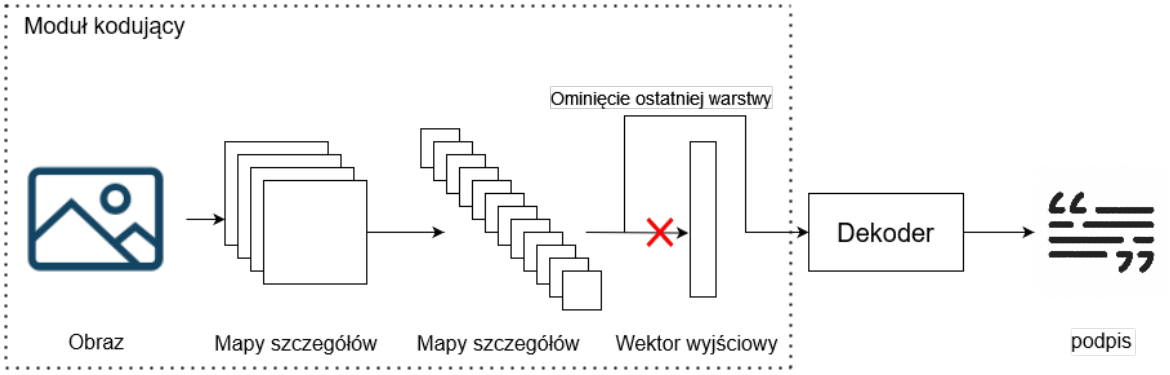
4.4. GoogleNet

Sieć GoogleNet [36] podobnie do pozostałych sieci splotowych składa się z kilku bloków posiadających różne konfiguracje warstw splotowych. Elementem rozpoznawalnym tejże sieci jest blok Inception. Sieć posiada aż 9 takich bloków. Wektor wyjściowy został utworzony z map szczegółów pochodzących z bloków pierwszego, trzeciego, szóstego oraz

dziewiątego. Taki wybór w najlepszy sposób uwzględnia różne poziomy szczegółowości obrazu – od najbardziej ogólnych pochodzących z wczesnych etapów sieci do najbardziej szczegółowych.

4.5. Klasyczne podejście

W celu sprawdzenia skuteczności zastosowanego rozwiązania zostało ono porównane z klasycznym połączeniem modułu kodującego z dekodującym. W tym przypadku dane wyjściowe pochodzące z sieci splotowej zostały wykorzystane jako pierwszy element sekwencji wejściowej sieci rekurencyjnej. W celu osiągnięcia takowego połączenia konieczna była modyfikacji architektury sieci splotowych przedstawiona na rysunku 4.3.



Rysunek 4.3. Diagram przedstawiający architekturę wykorzystującą klasyczne połączenie modułów kodującego oraz dekodującego. Opracowanie własne.

Z architektury sieci splotowej została usunięta ostatnia warstwa odpowiedzialna za przekształcenie danych do ostatecznego wektora odpowiadającego klasom rozpoznawanych obiektów. Takie podejście ograniczające modyfikację modułu kodującego jedynie do ostatniej warstwy pozwala wykorzystać ich wstępnie wytrenowane wagi poprzez tak zwaną technikę transferu wiedzy.

4.6. Transfer wiedzy

Szeroką popularnością w dziedzinie trenowania modeli sieci neuronowych cieszy się technika przenoszenia wiedzy. Jest to technika uczenia maszynowego, w której model uczony jest na jednym zadaniu, a następnie wykorzystuje tę wiedzę do lepszego rozwiązania innego, zazwyczaj podobnego zadania. Może zostać ona podzielona na dwa kluczowe kroki:

- Początkowe uczenie modelu na dużym zbiorze danych i zadaniu, na którym jest dostępna duża ilość informacji.
- Dostosowanie modelu do nowego zadania. Parametry modelu są dostosowywane do specyfiki nowego zadania, a wagi nauczone podczas początkowe uczenia są używane jako punkt wyjścia.

W przypadku generowania podpisów do obrazków technikę przenoszenia wiedzy można zastosować poprzez wykorzystanie wcześniej wyuczonych modeli sieci splotowych w ra-

mach modułu kodującego. Trenowanie splotowych sieci neuronowych od zera na dużym zbiorze danych może być czasochłonne i wymagać dużej ilości zasobów obliczeniowych. Wykorzystanie wstępnie wytrenowanych modeli pozwala uniknąć tego procesu, ponieważ modele te są już przeszkolone do ekstrakcji cech, co jest ich głównym zadaniem w całej architekturze. Z tego względu dane treningowe zostały poddane wcześniejszemu przetworzeniu przez moduł kodujący, a trening polegał na uczeniu jedynie modułu dekodującego w przypadku rozwiązania wykorzystującego proste połączenie między modułami. Rozwiązanie angażujące dodatkowe warstwy w pełni połączone wymaga również ich wyuczenia, jednakże architektura została skomponowana w ten sposób, aby nie ingerowała w wewnętrzną strukturę sieci splotowych, a co za tym idzie, transfer wiedzy również w tym przypadku może zostać wykorzystany.

5. Opis przeprowadzonych eksperymentów

W celu generowania podpisów do obrazków wykorzystana została architektura składającą się z modułu kodującego oraz modułu dekodującego. Ich połączenie zostało przedstawione w poprzednim rozdziale, jednakże w celu sprawdzenia, w jakim stopniu zaproponowane rozwiązanie miało wpływ na otrzymane wyniki, zostało ono porównane z architekturą wykorzystującą klasyczne podejście. W eksperymentach zostały wykorzystane ich różne konfiguracje. Dodatkowo wyniki zostały porównane z architekturami dedykowanymi do zadania generowania podpisów. Wykorzystane moduły kodujące:

- AlexNet [34] – jedna z najbardziej wpływowych sieci w historii przetwarzania obrazów, cechuje ją duża liczba parametrów, co pozwala na wyodrębnienie dużej liczby cech.
- GoogLeNet [36] – sieć charakteryzuje się modułami inception mającymi na celu efektywne wykorzystanie zasobów obliczeniowych.
- VGGNet [11] – bardzo głęboka sieć neuronowa wykorzystująca małe filtry splotowe pozwalających na wyodrębnienie cech o różnym stopniu skomplikowania.
- ResNet [35] – bardzo głęboka sieć neuronowa charakteryzująca się blokami rezydualnymi, które umożliwiają efektywne uczenie się sieci neuronowej o dużej liczbie warstw.
- VIT [12] - transformer wizyjny będący rozszerzeniem modelu transformer. Implementuje on mechanizm atencji, w celu wyodrębniania poszczególnych cech obrazu. Ze względu na jego znacząco różniącą się architekturę w porównaniu do klasycznych sieci splotowych został on wykorzystany tylko dla rozwiązania klasycznego. Podejście wykorzystujące dane z wewnętrznych warstw splotowych nie jest możliwe do zastosowania, ponieważ transformer wizyjny nie posiada takowych warstw.

Wykorzystane moduły dekodujące:

- RNN [2] – podstawowa rekurencyjna sieć neuronowa.
- LSTM [4] – jest rozszerzeniem podstawowej rekurencyjnej sieci neuronowej, której głównym celem jest rozwiązanie problemu zanikającego gradientu.
- GRU [5] – posiada mniej parametrów niż LSTM, co w teorii pozwala na szybsze i wydajniejsze uczenie się.
- Transformer [7] – umożliwia na analizowanie pełnej sekwencji w jednym momencie poprzez wykorzystanie modułu atencji.

Każda kombinacja modułu kodującego wraz z modułem dekodującym została sprawdzona pod względem:

- wydajności uczenia poprzez sprawdzenie czasu potrzebnego na przetworzenie przez architekturę treningowego zbioru danych,
- wydajności generowania podpisów, poprzez sprawdzenie czasu potrzebnego na przetworzenie testowego zbioru danych,
- efektywności poprzez sprawdzenie skuteczności generowania podpisów.

Wydajność testowania oraz skuteczność generowania podpisów została również spraw-

dzona dla dedykowanej architektury GIT [17] wykorzystującej model Florence [18] jako moduł kodujący wraz z klasycznym transformerem jako modułem dekodującym. W tym celu wykorzystane zostały wagi udostępnione przez autorów – pozwoliło to na całkowite pominięcie etapu uczenia tejże sieci.

5.1. Trenowanie pełnej architektury

W celu sprawdzenia jak duże znaczenie ma wykorzystanie wstępnie wytrenowanych modułów kodujących, ich wydajność treningowa została porównana do wydajności uczenia pełnej architektury. Porównane zostały wszystkie wcześniej wymienione kombinacje modułów kodujących wraz z modułami dekodującymi. Oprócz tego do zbioru modułów dekodujących dodany został również podstawowy wariant splotowej sieci neuronowej. Ponownie zostały wykorzystane dwa sposoby połączenia modułów:

- zaproponowane rozwiązanie wykorzystujące dane z wewnętrznych warstw splotowych,
- klasyczne podejście wykorzystujące jedynie wartości wyjściowe z ostatniej warstwy splotowej.

5.2. Wstępne przetwarzanie danych

Dane wejściowe w postaci macierzy pikseli obrazów cyfrowych zostały normalizowane, a ich wymiary ujednolicone w celu uproszczenia obliczeń. Natomiast przetwarzanie języka naturalnego odbyło się poprzez zastosowanie tokenizacji – w tym celu został wykorzystany wcześniej wytrenowany model Distilbert [37].

5.3. Metryki

Oprócz parametrów dotyczących wydajności trenowania oraz testowania podanych rozwiązań sprawdzona została również skuteczność otrzymanych wyników przy pomocy wybranych metryk.

5.3.1. BLEU

BLEU (ang. Bilingual Evaluation Understudy) [38] – metryka oryginalnie wywodząca się z dziedziny tłumaczenia maszynowego, jednakże ze względu na taki sam format danych, idealnie pasuje również do ewaluacji wyników generowania podpisów do obrazów. Zakres wartości metryki zawiera się w przedziale od 0 do 1, gdzie najlepszą wartością jest górna granica.

5.3.2. METEOR

METEOR (ang. Metric for Evaluation of Translation with Explicit ORdering) [39] – metryka będąca rozwinięciem metryki BLEU. Odznacza się lepszą oceną korelacji na poziomie zdań, a nie tak jak w przypadku BLEU, na poziomie całego korpusu. Zakres wartości metryki zawiera się w przedziale od 0 do 1, gdzie najlepszą wartością jest górna granica.

5.3.3. CIDEr

CIDEr (ang. Consensus-based Image Description Evaluation) [40] – dedykowana metryka do oceny jakości generowanych opisów obrazów. Poprzez zastosowanie algorytmu TF-IDF [41] pozwala na ocenę jakości generowanych podpisów w oparciu o podobieństwo do wszystkich podpisów, a nie tylko do jednego. Zakres jej wartości jest większy niż w przypadku pozostałych metryk, ponieważ wynosi od 0 do 10, gdzie wartość 10 oznacza najlepszy wynik.

5.3.4. SPICE

SPICE – wykorzystuje ona rozbiór logiczny zdania w celu stworzenie jego grafu, które są następnie porównywane z grafami pozostałych zdań. Dzięki temu metryka ta jest w stanie ocenić podobieństwo poprzez strukturę zdania, a nie tylko występowanie danych słów. Tak jak w przypadku metryk BLEU i METEOR zakres wartości zawiera się w przedziale od 0 do 1, gdzie najlepszą wartością jest górna granica.

W przypadku wyników architektur wykorzystujących różne kombinacje modułów kodujących oraz dekodujących zostały one przedstawione poprzez tylko dwie metryki: BLEU i CIDEr. Pozostałe wskaźniki zostały wykorzystane jedynie w przypadku porównania wyników bezpośrednio z wartościami otrzymanymi przez autorów dla dedykowanych rozwiązań. Ograniczenie zostało wprowadzone, w celu zachowania przejrzystości wyników, a także ze względu na fakt, iż metryki BLEU i CIDEr powinny być w pełni wystarczające, aby ocenić jakość generowanych podpisów.

5.4. Wykorzystane środowiska

W celu porównania wpływu użytych jednostek obliczeniowych na efektywność badanych architektury wykorzystane zostały różne karty graficzne:

- NVIDIA GeForce GTX 1650 4GB – jednostka GPU,
- NVIDIA Tesla T4 16 GB – jednostka GPU,
- Intel Core i7-1280P – jednostka CPU.

W przypadku karty graficznej Tesla T4 wszelkie testy zostały wykonane za pośrednictwem platformy Google Colab umożliwiającej bezpłatny dostęp do zaawansowanych jednostek obliczeniowych. Pozostałe testy zostały wykonane na komputerze osobistym

Docelowa architektura została zaimplementowana przy użyciu gotowych modeli sieci neuronowych zawartych w bibliotekach PyTorch [42] oraz Transformers [43] dostępnych w języku programowania Python.

5.5. Zbiory danych

Istnieje wiele zbiorów danych zawierających obrazy wraz z ich opisami. Jednymi z najpopularniejszych są:

- MS COCO [16] – składa się on z ponad 120 tysięcy zdjęć.
- Flickr [44] – posiada on mniej zdjęć niż zbiór MS COCO, ponieważ jego szeroki wariant zawiera ich 30 tysięcy, natomiast węższy 8 tysięcy.

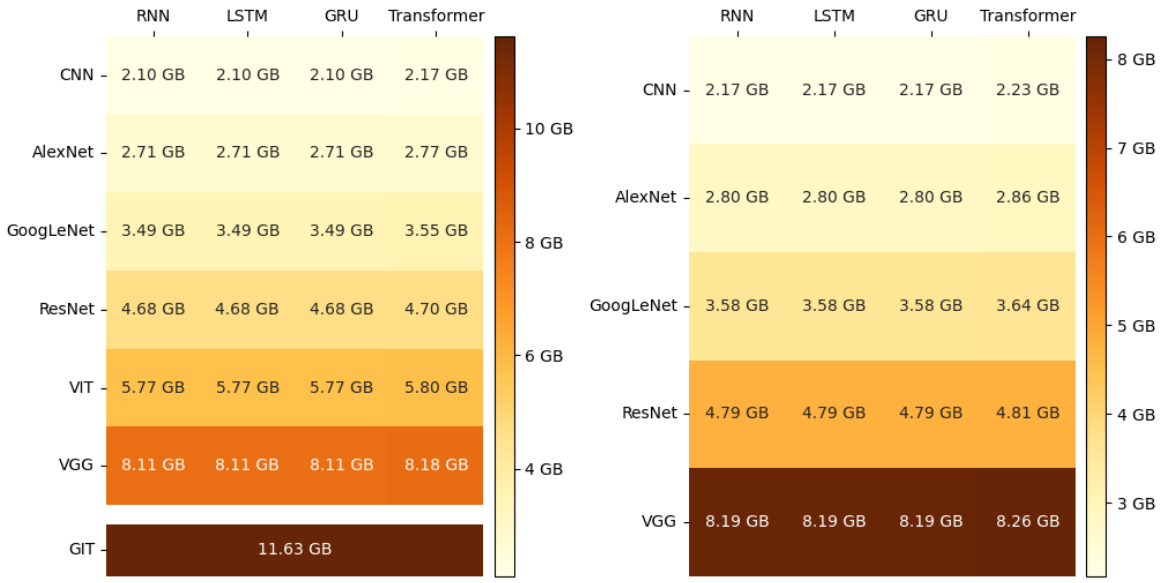
Ze względu na konieczność wytrenowania dużej liczby architektur niezwykle istotne jest dokonanie tego w sensownym oknie czasowym, co może być niewykonalne przy wykorzystaniu zbioru MS COCO. Z tego względu został wykorzystany węższy wariant zbioru Flickr zawierający 8 tysięcy zdjęć. Ten wybór pozwolił również sprawdzić skuteczność modelu GIT bez wcześniejszego go wytrenowania, posługując się jedynie modelem udostępnionym przez autorów, który został wytrenowany na zbiorze MS COCO. Zbiór danych został podzielony na:

- zbiór treningowy – 6 tysięcy zdjęć,
- zbiór testowy – 1 tysiąc zdjęć,
- zbiór walidacyjny – 1 tysiąc zdjęć.

Wszystkie kombinacje wykorzystanych architektur zostały wyuczone przy pomocy treningowego zbioru danych. Dodatkowo w trakcie treningu jakość modelu była sprawdzana przy pomocy zbioru walidacyjnego. Dane były przetwarzane w seriach liczących po 32 elementy. Przetwarzanie było powtarzane 500 razy w przypadku metody klasycznej, natomiast dla zaproponowanego rozwiązania przetwarzanie odbywało się 250 razy.

6. Wyniki

Poniższe dane przedstawione na rysunku 6.1 prezentują ilość pamięci RAM wyrażonej w gigabajtach potrzebnej do uruchomienia poszczególnych architektur dla rozwiązania klasycznego oraz wykorzystującego wektor składający się z danych z map szczegółów sieci splotowych.



(a) Klasyczne połączenie.

(b) Połączenie wykorzystujące mapy szczegółów.

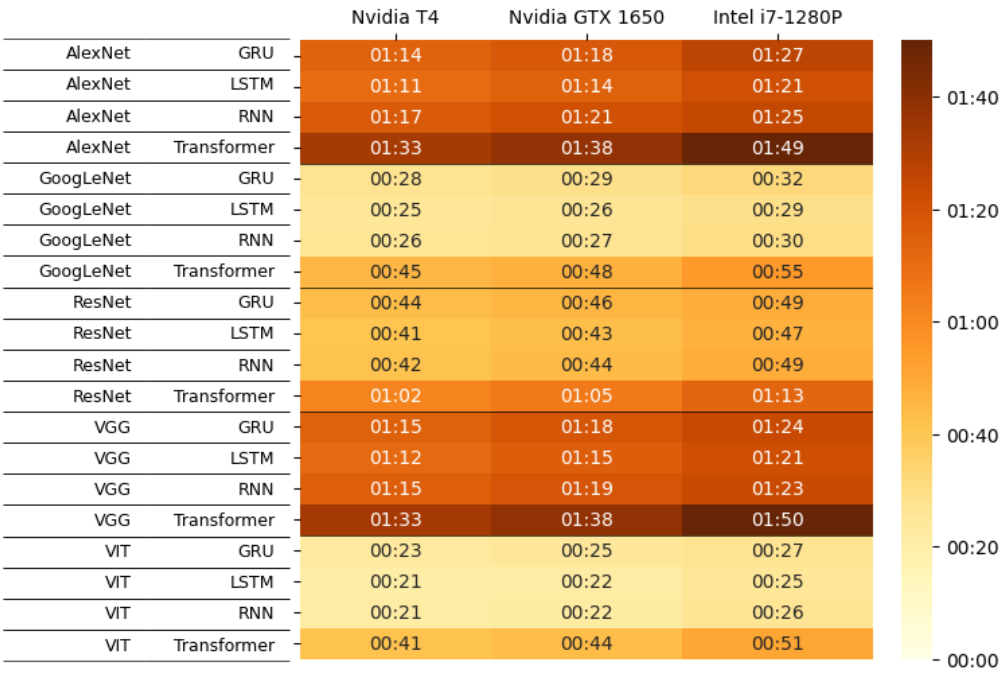
Rysunek 6.1. Pamięć RAM potrzebna do uruchomienia testowanych architektur wyrażona w gigabajtach. Opracowanie własne.

Można zauważyć, iż największe różnice w ilości potrzebnej pamięci RAM są pomiędzy poszczególnymi modułami kodującymi. W przypadku modułów dekodujących zauważalna różnica jest pomiędzy wykorzystaniem transformera a pozostałymi sieciami rekurencyjnymi. Najpewniej wynika to z tego, iż sieci rekurencyjne przetwarzają tylko jeden element w danym czasie w przeciwieństwie do transformera, który przy pomocy modułu atencji analizuje całą sekwencję w jednym momencie, przez co konieczne jest jej załadowanie do pamięci komputera. Istotnym jest również fakt, iż połowa modułów kodujących potrzebuje ponad 4 GB pamięci, a co za tym idzie, nie było możliwe ich wykorzystanie przy pomocy karty graficznej Nvidia GeForce GTX 1650, ponieważ posiada ona tylko cztery gigabajty pamięci. Tak duża ilość potrzebnej pamięci dotyczy bardzo głębokich sieci oraz wykorzystujących transformer wizyjny. Powodem takiego stanu rzeczy jest fakt, iż w przypadku bardzo głębokich sieci splotowych architektura posiada ogromną liczbę parametrów ze względu na dużą liczbę małych filtrów oraz warstw. Natomiast transformer wizyjny przetwarza obraz do postaci sekwencji, która musi być załadowana w całości, co również znacząco wpływa na rozmiar potrzebnej pamięci. Warto również zauważyć, iż wykorzystanie danych pochodzących z map szczegółów sieci splotowych, tylko nieznacznie wpłynęło na zwiększenie wykorzystywanej pamięci RAM. Wpływ tutaj miało

wykorzystanie warstw w pełni połączonych, które w porównaniu do warstw splotowych nie są aż tak obciążające i zajmują stosunkowo mniejszą ilość pamięci.

6.1. Wydajność uczenia modułów dekodujących dla połączenia klasycznego

Dane przedstawione na rysunku 6.2 prezentują czas potrzebny na przetworzenie jednej epoki treningowej poszczególnych kombinacji modułów kodujących oraz dekodujących wykorzystujących wcześniej wytrenowane moduły kodujące przy pomocy klasycznego połączenia.



Rysunek 6.2. Czas potrzebny na przetworzenie jednej epoki treningowej poszczególnych kombinacji modułów kodujących oraz dekodujących wykorzystujących wcześniej wytrenowane moduły kodujące. Dane przedstawione w skali logarytmicznej. Opracowanie własne.

Ponownie tak jak w przypadku potrzebnej pamięci do alokacji architektury tutaj również najbardziej zauważalne są różnice czasowe pomiędzy poszczególnymi modułami kodującymi. W przypadku modułów dekodujących różnice są widoczne jedynie pomiędzy wykorzystaniem transformera a pozostałymi sieciami rekurencyjnymi. Co ciekawe, długość przetwarzania nie jest zależna od stopnia skomplikowania sieci. Ze względu na wcześniejsze wygenerowanie danych wejściowych do modułów dekodujących poprzez przetworzenie obrazów przy pomocy wstępnie wyuczonych modułów kodujących, czas potrzebny na przetworzenie jednej epoki treningowej jest zależny od wielkości ostatniej warstwy modułu kodującego, który jest następujący dla poszczególnych sieci:

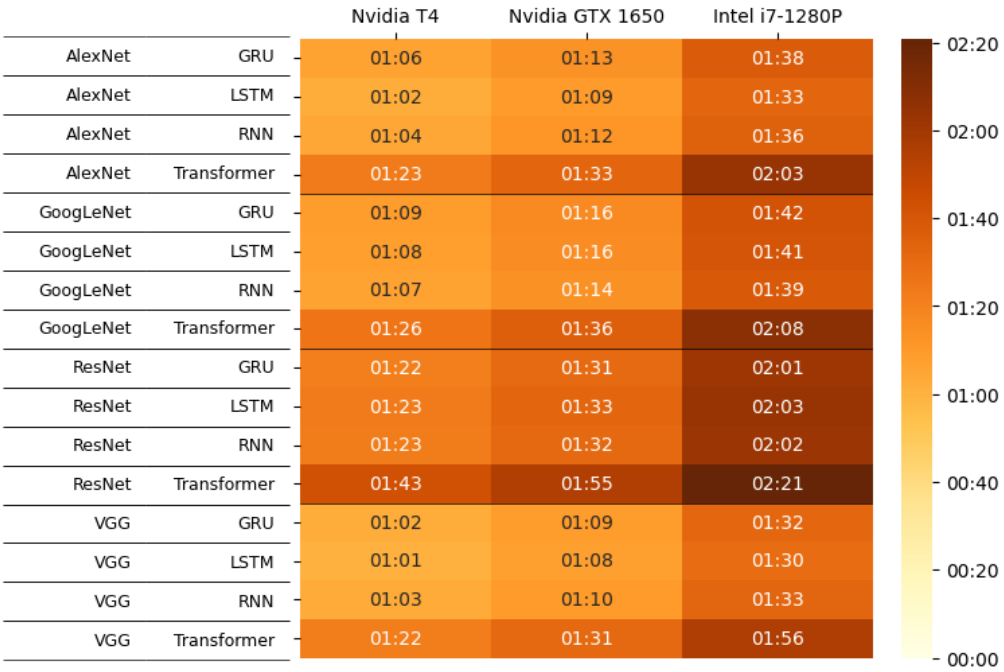
- AlexNet – 4096,
- GoogLeNet – 1024,
- VGGNet – 4096,
- ResNet – 2048,

- VIT – 768.

Biorąc to pod uwagę, łatwo można zauważyć, iż sieć z najmniejszą warstwą końcową, czyli VIT, potrzebuje najmniej czasu na przetworzenie jednej epoki treningowej, a sieci z największą warstwą końcową, czyli AlexNet oraz VGGNet, potrzebuje go najwięcej. W przypadku trenowania również modułu kodującego bez zastosowania wstępnie wyuczonych modeli możliwe byłoby dostosowanie wielkości ostatniej warstwy, w celu zmniejszenia czasu potrzebnego na przetworzenie danych przez moduł dekodujący.

6.2. Wydajność uczenia architektury dla zaproponowanego połączenia

W przypadku wykorzystania dodatkowych danych pochodzących z pośrednich warstw splotowych konieczne było zastosowanie warstw w pełni połączonych, które również musiały zostać wytrenowane wraz z modułem dekodującym. Dane przedstawione na rysunku 6.3 prezentują czas potrzebny na przetworzenie jednej epoki treningowej poszczególnych kombinacji modułów kodujących oraz dekodujących wykorzystujących wcześniej wytrenowane moduły kodujące przy pomocy zaproponowanego połączenia.



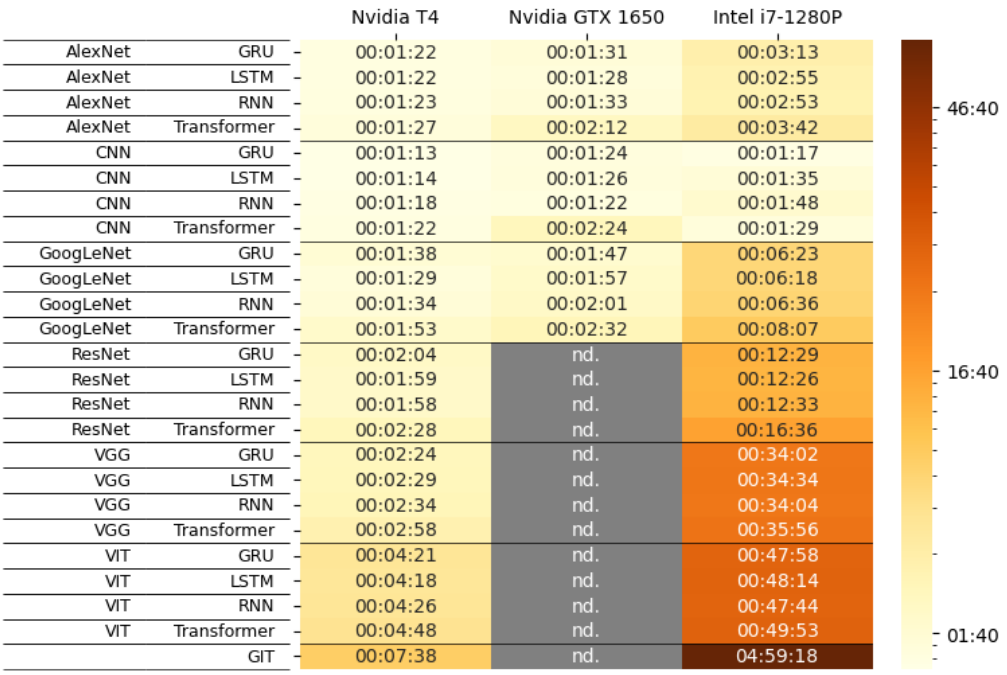
Rysunek 6.3. Czas potrzebny na przetworzenie jednej epoki treningowej poszczególnych kombinacji modułów kodujących oraz dekodujących wykorzystujących wcześniej wytrenowane moduły kodujące przy pomocy zaproponowanego połączenia. Opracowanie własne.

Ze względu na generowanie wektorów wejściowych o takich samych rozmiarach dla każdego z modułów kodujących, różnica pomiędzy ich poszczególnymi typami nie jest tak widoczna, jak w przypadku klasycznego podejścia. W przypadku sieci AlexNet oraz VGG, których rozmiar wyjściowych wektorów pozostał niezmienny, wzrost czasu jest na poziomie około 10%, co jest przewidywalną różnicą ze względu na konieczność wyuczenia również warstw w pełni połączonych odpowiedzialnych za redukcję rozmiaru wektorów

pochodzących z warstw splotowych. Tak jak w poprzednim przypadku, ponownie można zauważyć różnicę pomiędzy sieciami rekurencyjnymi a transformerem. Również warto zwrócić uwagę na to, iż wykorzystanie jednostki GPU miało pewien wpływ na wydajność treningu, ale nie była to drastyczna poprawa – w większości przypadków spadek czasu wyniósł około 25% w przypadku podstawowej jednostki GPU, a około 30% w przypadku jednostki specjalistycznej w porównaniu do jednostki CPU. Można stwierdzić, iż wykorzystanie danych z warstw splotowych miało wpływ na długość treningu, mimo iż jest to jedynie kilka sekund przetwarzania epoki, biorąc pod uwagę fakt, iż w przypadku trenowania sieci konieczne jest przetworzenie danych kilkaset razy, różnica ta staje się znacząca – przetworzenie 250 epok daje różnicę na poziomie 45 minut.

6.3. Wydajność uczenia modułów kodujących oraz dekodujących

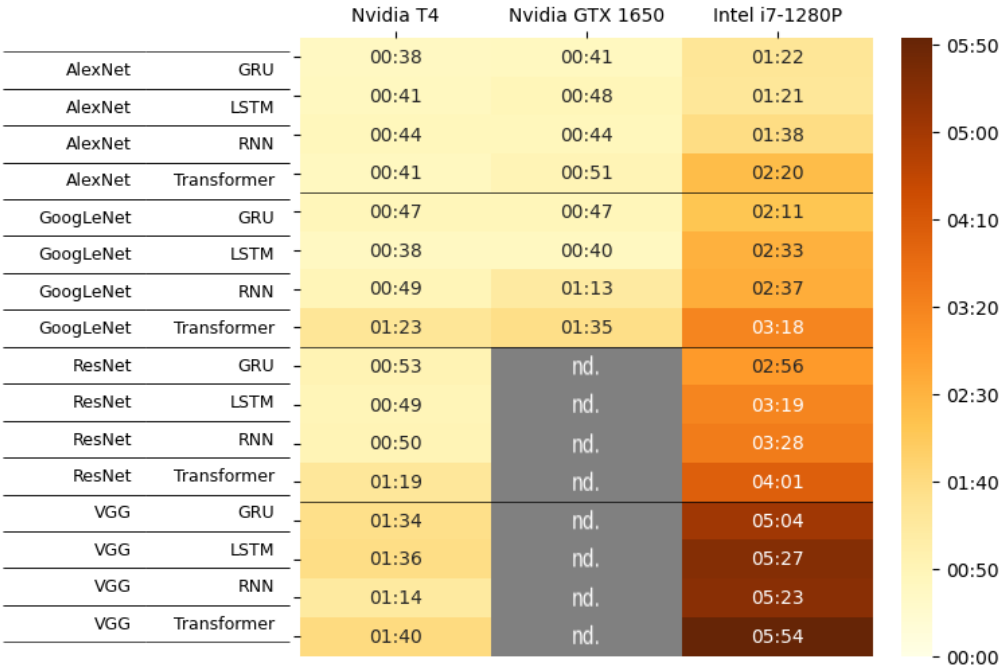
Dane przedstawione na rysunku 6.4 prezentują czas potrzebny na przetworzenie jednej epoki treningowej poszczególnych kombinacji modułów kodujących oraz dekodujących.



Rysunek 6.4. Czas potrzebny na przetworzenie jednej epoki treningowej poszczególnych kombinacji modułów kodujących oraz dekodujących. Opracowanie własne.

Niestety ze względu na ograniczenie pamięci RAM karty graficznej Nvidia GTX 1650 nie było możliwe przeprowadzenie treningu sieci zajmujących więcej niż 4 GB pamięci, czyli ResNet, VGG, VIT oraz GIT. W przypadku czasu potrzebnego na przetworzenie danych przy pomocy jednostki CPU rośnie on w sposób wykładniczy. Biorąc pod uwagę, iż założeniem przeprowadzonych eksperymentów było przetworzenie zbioru treningowej 500 razy, dla połowy sieci staje się to niewykonalne. Już w przypadku sieci GoogleNet połączonej z jedną z sieci rekurencyjnych, gdzie jedna epoka treningowa zajmuje ponad 6 minut, powtórzenie jej 500 razy zajęłoby 53 godziny. Natomiast w przypadku najdłużej przetwarzającej

sieci, czyli GIT, czas ten wyniósłby ponad 100 dni ciągłego przetwarzania. W przypadku pozostałych sieci, gdzie czas kilkudziesięciu godzin jest w wielu momentach akceptowalnym rzędem wielkości, to tak wydłużony czas uczenia znacząco utrudnia reagowanie na wszelkie błędy oraz problemy. Sytuacja jest znacznie lepsza w przypadku wykorzystania jednostki GPU. Pomiędzy modelem dostępnym w wielu komputerach stacjonarnych, Nvidia GTX 1650, a modelem dedykowanym do obliczeń na kartach graficznych, Nvidia T4, różnice są stosunkowo niewielkie. W przypadku architektury GIT przetworzenie danych 500 razy zajęłoby około 65 godzin, co w porównaniu do czasu potrzebnego w przypadku wykorzystania jednostki CPU jest osiągalne. Dodatkowo warto wspomnieć, iż korelacja pomiędzy liczbą parametrów ostatniej warstwy modułu kodującego a czasem potrzebny na trening nie pojawia się w tym przypadku. Wynika to z faktu, iż czas trenowania modułu kodującego rośnie o wiele znaczniej wraz z większym skomplikowaniem wykorzystanej architektury. Z tego względu również różnice pomiędzy architekturami wykorzystującymi klasyczne połączenie modułów a tymi wykorzystującymi dodatkowe dane z warstw splotowych nie są aż tak zauważalne – wzrost czasu treningu jest nieznaczny w porównaniu do całościowego czasu przetwarzania, co można zauważyć na rysunku 6.5.



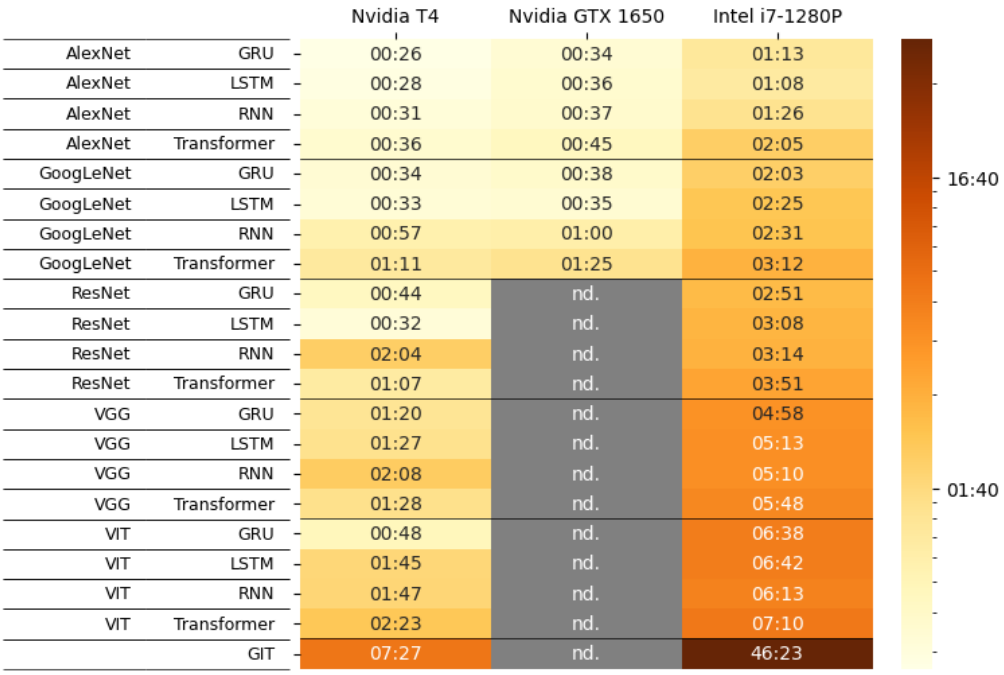
Rysunek 6.5. Czas potrzebny na przetworzenie jednej epoki treningowej poszczególnych kombinacji modułów kodujących oraz dekodujących wykorzystujących dane z warstw splotowych. Opracowanie własne.

Różnica w czasie trenowania pomiędzy klasyczną techniką połączenia modułu kodującego i dekodującego a zaproponowanym rozwiązaniem wykorzystującym dane z warstw splotowych jest bardzo zbliżona w obu przypadkach – trenowania modułu dekodującego oraz całej architektury. Jest to przewidywalny rezultat, ponieważ w obu podejściach dodat-

kowe obciążenie jest takie samo – konieczność wyuczenia dodatkowych warstw w pełni połączonych.

6.4. Wydajność generowania podpisów

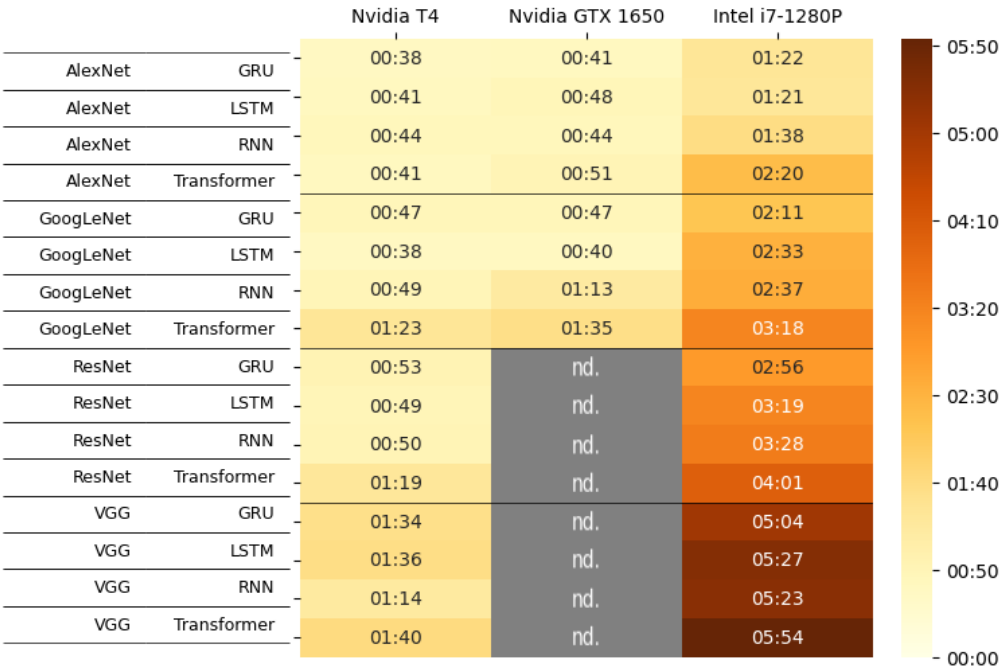
Na rysunku 6.6 zostały przedstawione czasy potrzebne na wygenerowanie podpisów obrazków pochodzących z testowego zbioru danych dla poszczególnych architektur.



Rysunek 6.6. Czas potrzebny na wygenerowanie podpisów obrazków pochodzących z testowego zbioru danych dla poszczególnych architektur, które zostały wyuczone przy użyciu wcześniej wytrenowanych modułów kodujących, wykorzystując klasyczne połączenie modułów. Dane przedstawione w skali logarytmicznej. Opracowanie własne.

W przypadku generowania podpisów konieczne jest wykorzystanie bezpośrednio modułu kodującego, ponieważ w warunkach naturalnych przetwarzane zdjęcie nie jest wcześniej znane. Z tego względu wyniki dla modeli wykorzystujących wcześniej wyuczone moduły kodujące oraz modele trenowane od zera są takie same – w obu przypadkach obraz przetwarzany jest w ten sam sposób. Wskutek tego nie było możliwe przeprowadzenie eksperymentów przy użyciu karty graficznej Nvidia GTX 1650, tych sieci, których alokacja przekracza 4 GB pamięci. Czas wymagany do przetworzenia zbioru testowego, tak jak w przypadku uczenia architektur, wraz z wykorzystaniem bardziej rozbudowanego modułu kodującego wzrasta, a w przypadku modułów dekodujących znaczące różnice są widoczne jedynie pomiędzy transformerem a pozostałymi sieciami. Ponownie najbardziej wymagającą architekturą okazało się rozwiązanie GIT. Przy jego pomocy przetworzenie tysiąca zdjęć zajęłoby ponad 45 minut, co daje niecałe 3 sekundy na zdjęcie – taki czas w wielu przypadkach można uznać za akceptowalny, jednakże ponownie pokazuje to, iż prostsze sieci są bardziej przystępne dla osób z ograniczonymi zasobami sprzętowymi. Najszybsza

okazała się sieć AlexNet połączona z siecią GRU – w tym przypadku przetworzenie całego zbioru testowego zajmuje mniej niż pół minuty. W przypadku wykorzystania danych z wewnętrznych warstw splotowych modułu kodującego, w celu wygenerowania wektora wejściowego modułu kodującego czas generowania podpisów nie zmienił się znacząco, co można zauważyć na rysunku 6.7.



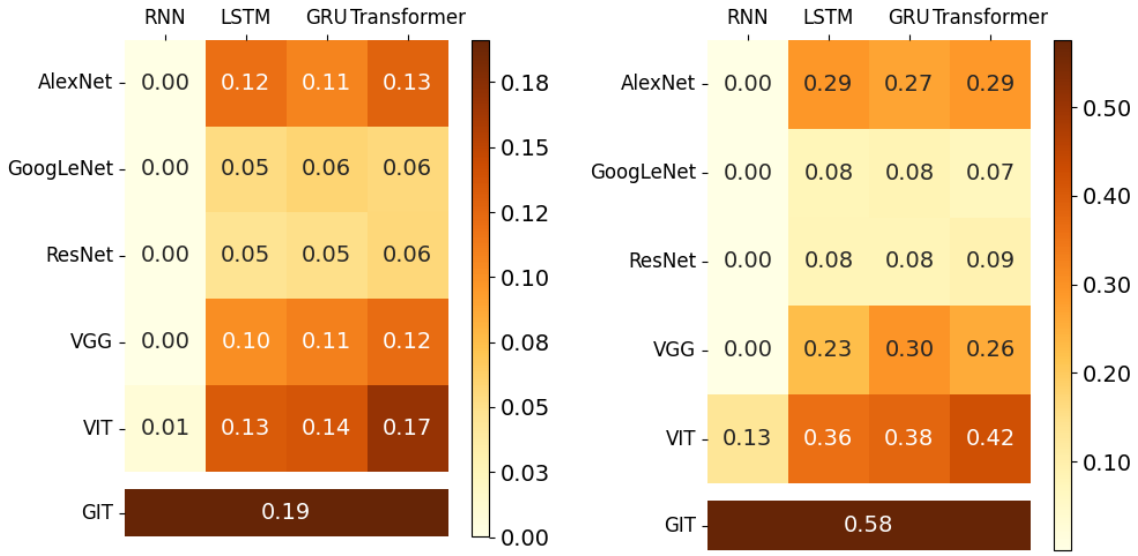
Rysunek 6.7. Czas potrzebny na wygenerowanie podpisów obrazków pochodzących z testowego zbioru danych dla poszczególnych architektur, które zostały wyuczone przy użyciu wcześniej wytrenowanych modułów kodujących, wykorzystując dane z warstw splotowych. Dane przedstawione w skali logarytmicznej. Opracowanie własne.

Tak jak w pozostałych przypadkach, główna różnica czasowa pomiędzy klasycznym podejściem a wykorzystaniem warstw splotowych wynika z konieczności przetworzenia danych przy pomocy dodatkowych warstw w pełni połączonych. Wzrost czasu wynosi kilka sekund i nie jest to znacząco różnica, biorąc pod uwagę długość przetwarzania całego zbioru testowego.

6.5. Skuteczność architektury wykorzystującej klasyczne połączenie

Wydajność architektur jest niezwykle istotna, ale by móc w pełni ocenić ich przydatność, konieczne jest sprawdzenie skuteczności generowania podpisów. W tym celu wykorzystane zostały metryki BLEU oraz CIDEr. Dane przedstawione na rysunku 6.8 prezentują wartości tychże metryk dla architektury GIT oraz poszczególnych kombinacji modułów kodujących i dekodujących. Wyraźnie widoczna jest bardzo duża skuteczność architektury GIT. W przypadku metryki BLEU jest ona większa jedynie o 0,02 od drugiej najlepszej architektury – kodera VIT połączonego z transformerem, ale w przypadku metryki CIDEr różnica ta jest już znacznie większa, ponieważ wynosi aż 0,16. Może to świadczyć o lepszej

jakości generowanych podpisach. W przypadku wykorzystywania sieci RNN osiągnięcie sensownych wyników było niemożliwe.

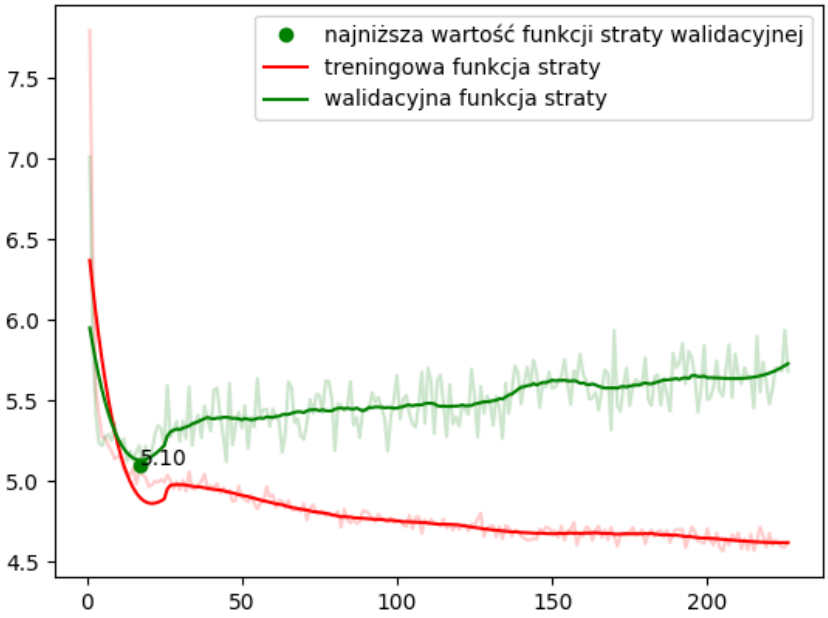


(a) Wartości metryki BLEU

(b) Wartości metryki CIDEr

Rysunek 6.8. Wartości metryk BLEU oraz CIDEr dla poszczególnych kombinacji modułów kodujących i dekodujących. Opracowanie własne

W prawie wszystkich przypadkach funkcja straty zbioru walidacyjnego szybko osiągała swoje minimum, zaczynała rosnąć, co najczęściej sygnalizuje przeuczenie modelu – można to zauważyć na rysunku 6.9.

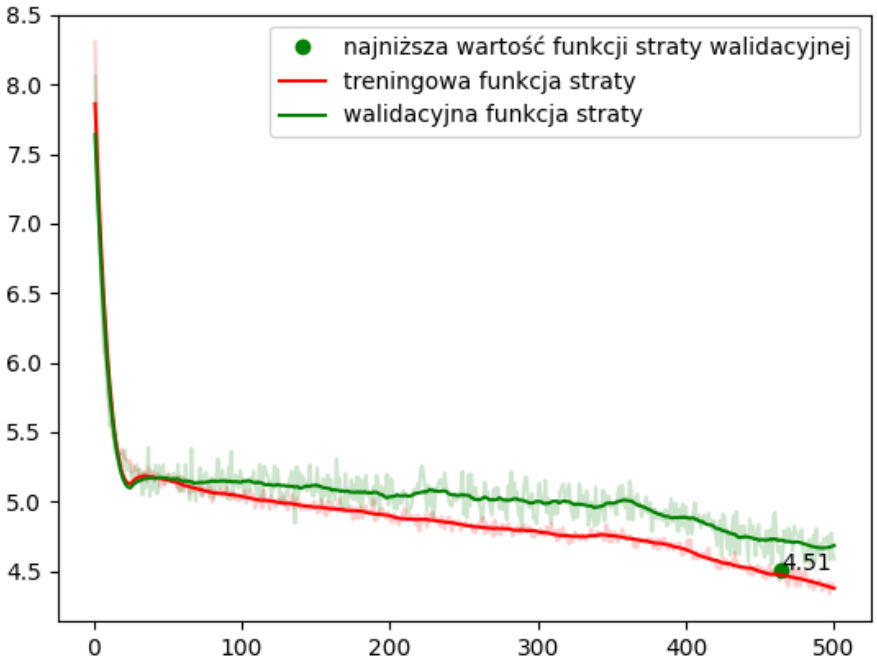


Rysunek 6.9. Wykres wartości funkcji straty treningowej oraz walidacyjnej dla architektury składającej się z sieci AlexNet i RNN. Opracowanie własne.

Architektura z wartością funkcji straty na poziomie 5 generowała zdania zawierające jedynie literę „a”, ponieważ jest ona jednym z najczęściej występujących słów w języku angielskim. Wyjątkiem w przypadku wykorzystania sieci RNN jest jej połączenie z transformerem wizyjnym. Dla takiego połączenia metryka BLEU wynosi jedynie 0,01, jednakże wartość CIDEr jest już znacznie wyższa i wynosi 0,13. Wynika to z tego, iż modelowi nie udało się wygenerować pełnych zdań, a jedynie pojedyncze słowa. Przykładowe wyniki zostały przedstawione na rysunku 6.10. Metryka BLEU bierze pod uwagę długość wygenerowanego podpisu, dlatego też w przypadku wygenerowania jedynie pojedynczego słowa, wartość tej metryki jest o wiele niższa w porównaniu do metryki CIDEr.

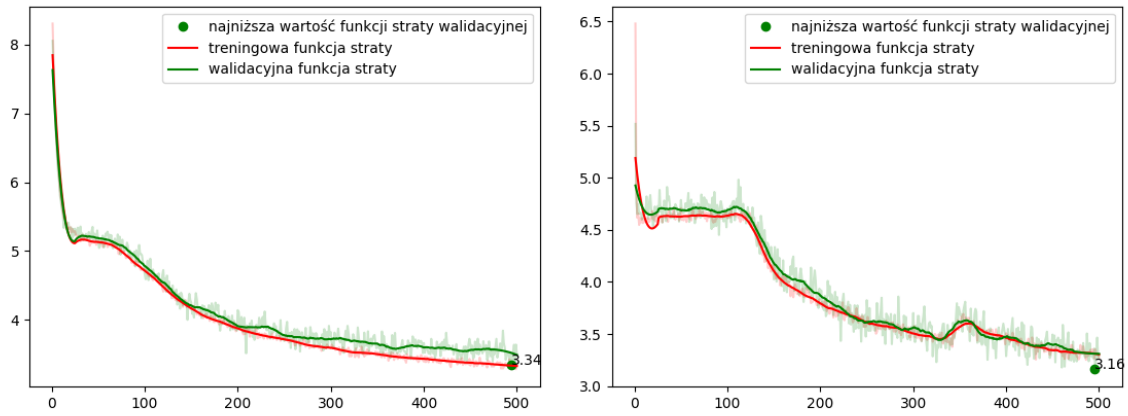
Rysunek 6.10. Przykładowe obraz wraz z podpisami wygenerowanymi przez architekturę wykorzystującą transformer wizyjny wraz z siecią RNN. Opracowanie własne.

W przypadku tego połączenia funkcja straty walidacyjnej wyniosła wartość 4,5, co można zaobserwować na rysunku 6.11.



Rysunek 6.11. Wykres wartości funkcji straty treningowej oraz walidacyjnej dla architektury składającej się z transformera wizyjnego i sieci RNN. Opracowanie własne.

Jest to stosunkowo lepszy wynik od pozostałych architektur zawierających sieć RNN, biorąc również pod uwagę fakt, iż w przeciwieństwie do pozostałych rozwiązań na wykresie funkcji straty widać dalszy potencjał na malenie jej wartości, a co za tym idzie na nawet lepsze wyniki. Niestety ze względu na ograniczenia czasowe narzucony został limit 500 epok treningowych, które architektura łącząca VIT oraz RNN osiągnęła. Jednocześnie warto zauważyć, że wyuczenie modelu zajęło podobny przedział czasowy, co architektura zawierająca sieć GRU oraz LSTM. W obu tych przypadkach średnia potrzebna na przetworzenie jednej epoki treningowej wyniosła mniej niż 30 sekund, co przy 500 epokach treningowych wynosi około 3 godzin, co jest bardzo zadowalającym czasem. Mimo widocznego potencjału sieci RNN w przypadku połączenia z transformerem wizyjnym nie jest to rozwiązanie, które można uznać za satysfakcjonujące, ponieważ w przypadku wykorzystania sieci RNN w połączeniu z innymi modułami kodującymi, nie udało się osiągnąć zadowalających wyników, a w przypadku wykorzystania transformerów wizyjnych, nie jest to rozwiązanie wydajne ze względu na potrzebę wykorzystania takich samych zasobów komputerowych jak w przypadku wykorzystania sieci GRU lub LSTM. Patrząc na krzywą funkcji straty architektury VIT wraz z siecią GRU widoczne na rysunku 6.12a można zauważyć jej spłaszczenie w ostatnich epokach treningowych, co nie jest widoczne w przypadku wcześniej omawianego połączenia VIT oraz RNN, jak również w przypadku VIT wraz z klasycznym transformerem, co zostało przedstawione na rysunku 6.12b.



(a) GRU **(b)** Transformer

Rysunek 6.12. Wykres wartości funkcji straty treningowej oraz walidacyjnej dla architektury składającej się z transformera wizyjnego wraz z różnymi modułami dekodującymi. Opracowanie własne.

Przypadek architektury zawierającej transformer wizyjny i klasyczny jest o wiele bardziej obiecujący niż w przypadku wykorzystania sieci RNN. Wartości funkcji straty są znacznie niższe, widoczna jest dalsza tendencja spadkowa krzywej, a co za tym idzie, możliwe byłoby dalsze poprawianie wyników. Niestety wykorzystywanie klasycznego transformera wiąże się z o wiele większym zapotrzebowaniem zasobów komputerowych. Przetworzenie jednej epoki treningowej architektury VIT i Transformer zajęło średnio 51 sekund, co jest wynikiem prawie dwa razu większym niż w przypadku wykorzystania sieci RNN, GRU lub

LSTM. Przykładowe podpisy wygenerowane przez tę architekturę zostały przedstawione na rysunku 6.13.

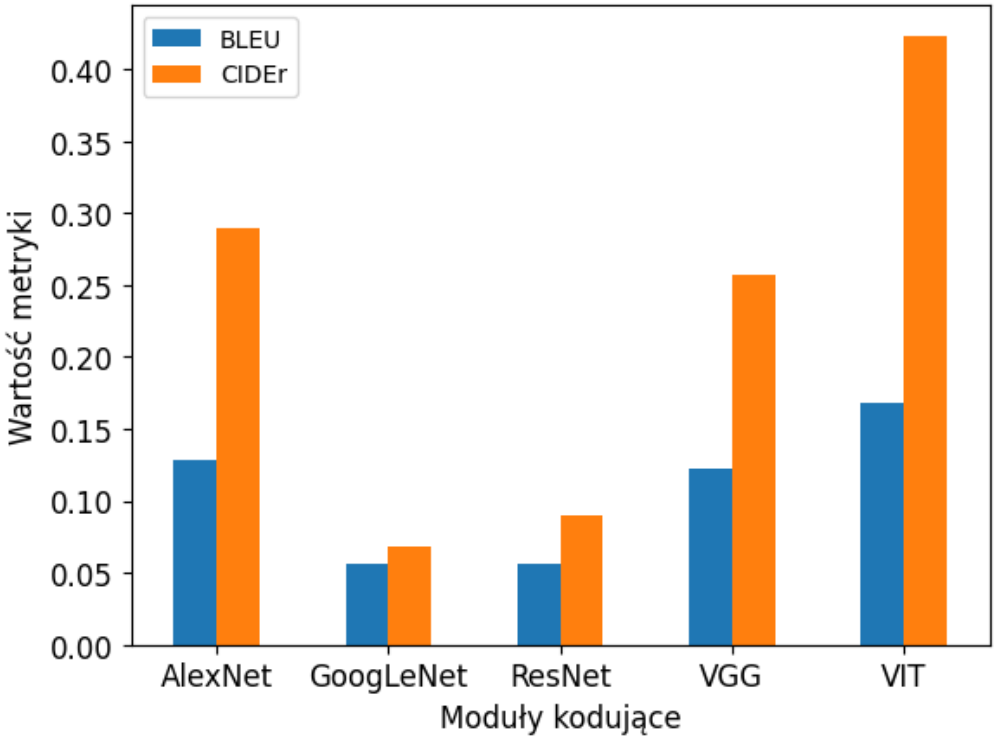
Rysunek 6.13. Przykładowe obrazy wraz z podpisami wygenerowanymi przez architekturę wykorzystującą transformer wizyjny wraz z klasycznym transformerem. Opracowanie własne.

Można zaobserwować, iż wygenerowane podpisy są o wiele dłuższe niż w porównaniu do architektury VIT i RNN, jednakże brakuje im logicznej spójności. W dużej mierze są one bez sensu, ale zawarte w nich rzeczowniki w większości odnoszą się do faktycznych obiektów znajdujących się na zdjęciach. Jest to dosyć logiczne ze względu na wykorzystanie wstępnie wyuczonych modułów kodujących, które były trenowane na zbiorze danych przeznaczonym do zadań klasyfikacyjnych. Wskazuje to na poprawne rozpoznawanie obiektów znajdujących się na obrazie, jednakże architektura nie potrafi ich połączyć w logiczną całości, za co odpowiedzialny jest moduł dekodujący generujący sekwencję. Mimo wykorzystania zaawansowanego modułu dekodującego, jakim jest transformer, jego danymi wejściowymi cały czas jest pojedynczy wektor reprezentujący obraz. Natomiast podpisy wygenerowane przez sieć GIT, widoczne na rysunku 6.14, są o wiele bardziej zadowalające, tworzą one zrozumiałe zdania i można zauważyć w nich identyfikacje odpowiednich relacji pomiędzy obiektami znajdującymi się na zdjęciach.

Rysunek 6.14. Przykładowe obrazy wraz z podpisami wygenerowanymi przez architekturę GIT. Opracowanie własne.

Tak dobry wynik w porównaniu do pozostałych architektur pokazuje duży wpływ modułu dekodującego na ostateczny rezultat oraz możliwości analizowania obrazu jako całą sekwencję, a nie tylko jako pojedynczy element początkowy. W przypadku tejże sieci warto również przypomnieć, iż w ramach testów wykorzystany został wcześniej wytrenowany model na zbiorze MS COCO, który nie był dostrojony do zbioru Flickr. Niestety tak dobra skuteczność oraz duży potencjał wiążą się z bardzo dużym zapotrzebowaniem na zasoby komputerowe. Wykorzystanie tego rozwiązania zajmuje bardzo dużo czasu i ze względu na ogromną liczbę parametrów nie jest możliwe jego użycie przy pomocy mniejszych kart graficznych, co znacząco wpływa na czas przetwarzania. W przypadku wykorzystania modelu wstępnie wyuczonego takie ograniczenia mogą nie być dużą przeszkodą, ale na pewno korzystając z modelu bez wstępnych wartości parametrów architektury, otrzymanie tak dobrego wyniku może okazać się nieosiągalne ze względu na ograniczenia zasobów komputerowych, czasowych lub budżetowych.

Warto również zwrócić uwagę na wyniki architektury wykorzystującej splotową sieć AlexNet. Rysunek 6.15 przedstawia wartości metryk wszystkich modułów kodujących w połączeniu z transformerem jako modułem dekodującym. Można na nim zauważyć bardzo dobry wynik w porównaniu do pozostałych architektur.

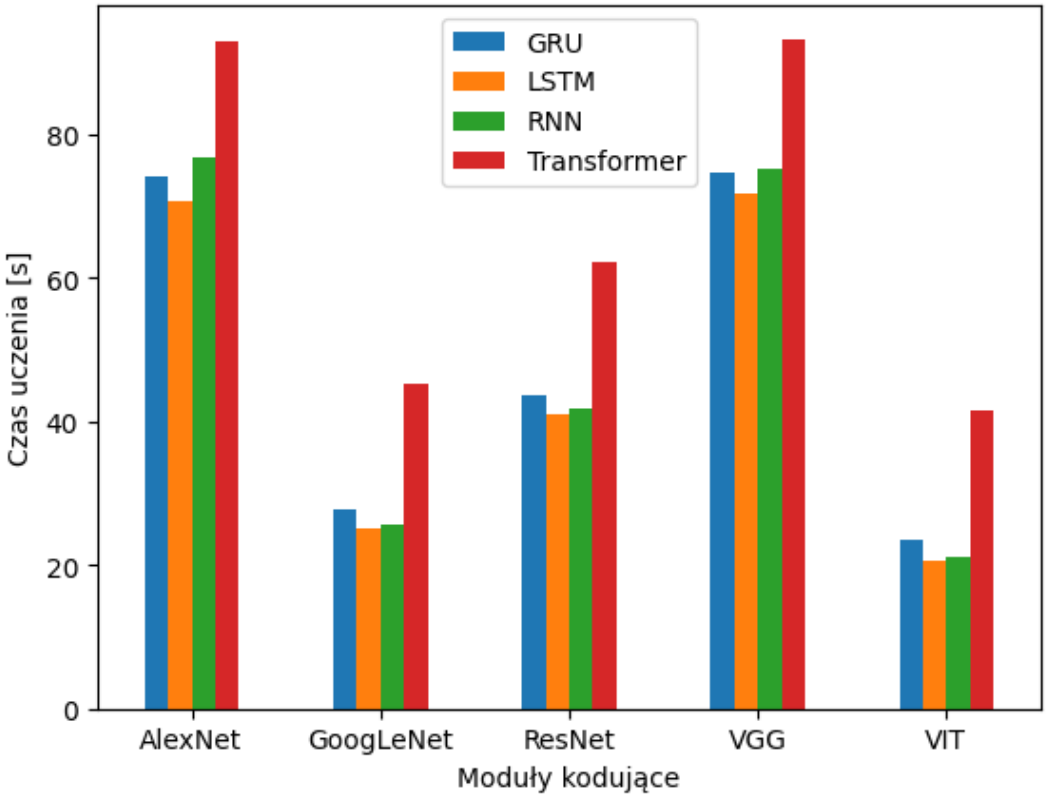


Rysunek 6.15. Wartości metryk BLEU oraz CIDEr dla poszczególnych modułów kodujących połączonych z transformerem jako modułem dekodującym. Opracowanie własne.

Sieć AlexNet ustąpiła jedynie miejsca architekturze GIT oraz kombinacjom modułów wykorzystujących transformer wizyjny. Wynika to z faktu, iż posiadała ona największą ostatnią warstwę w swojej architekturze. Zawierała ona ponad 4 tysiące parametrów – GoogLeNet, ResNet posiadały ich zdecydowanie mniej, ponieważ odpowiednio: 1024 oraz 2048. Przykładowe podpisy wygenerowane przez architekturę AlexNet wraz z transformerem zostały przedstawione na rysunku 6.16. Są one dosyć podobne jakościowo do tych otrzymanych przy pomocy VIT.

Rysunek 6.16. Przykładowe obraz wraz z podpisami wygenerowanymi przez architekturę wykorzystującą transformer wizyjny wraz z siecią RNN. Opracowanie własne.

Zaskakujący jest słabszy wynik architektury VGG, która posiadała taką samą liczbę parametrów jak AlexNet, a mimo wszystko okazała się mniej skuteczna w przypadku obu metryk. Ponownie można zwrócić uwagę na bardzo duży wpływ w analizowaniu obrazu poprzez wykorzystanie transformera wizyjnego, ponieważ pomimo posiadania najmniejszej liczby parametrów – jedynie 768 – architektury go wykorzystujące osiągnęły zdecydowanie najlepsze wyniki. Nie można jednocześnie zapomnieć o bardzo dużym wpływie liczby parametrów na czas potrzebny do wyuczenia danych architektur, które można zaobserwować na rysunku 6.17 dla karty graficznej Nvidia T4.

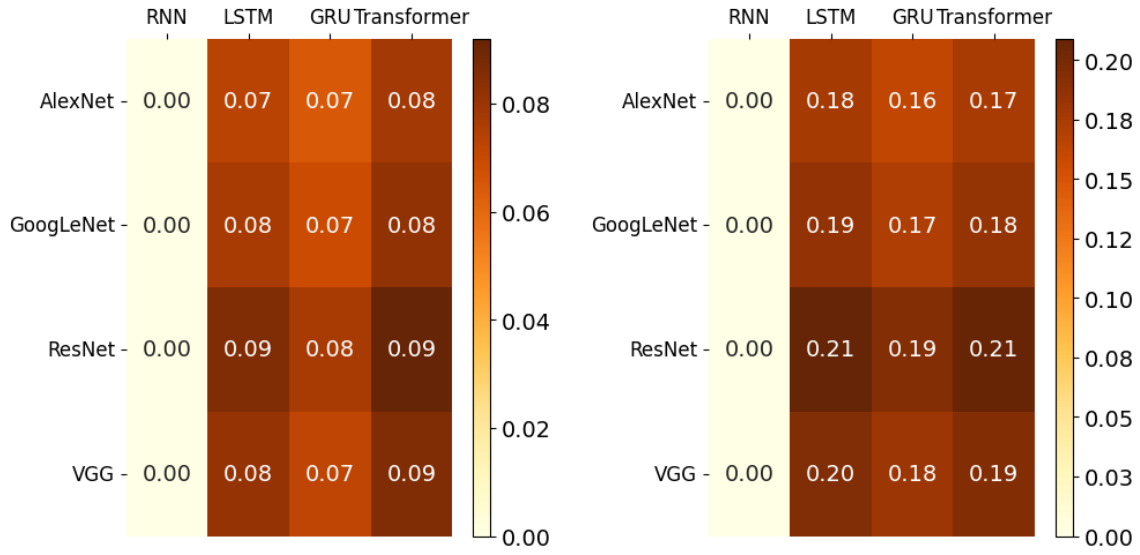


Rysunek 6.17. Czas przetwarzania jednej epoki treningowej przy użyciu kart graficznej Nvidia T4 dla poszczególnych modułów dekodujących wykorzystujących wstępnie wyuczone moduły kodujące. Opracowanie własne.

Widoczny jest bezpośredni wpływ liczby parametrów modułów kodujących na czas przetwarzania. Jest to przewidywalnym wynikiem, ponieważ w przypadku większej ilości danych, sieć rekurencyjna musi przetworzyć ich stosunkowo więcej. Interesującym jest fakt, iż nie miało to bezpośredniego przełożenia na lepsze wyniki. Pokazuje to, iż bardzo istotne jest przeanalizowanie obrazu w odpowiedni sposób i najpewniej zwiększenie liczby parametrów w modułach kodujących miałoby w pewnym stopniu wpływ na poprawę skuteczności generowania podpisów, ale porównując różnice w przypadku takich sieci jak GoogLeNet, ResNet, VGG nie są one drastyczne. Warte zweryfikowania na pewno byłoby zwiększenie tejże ilości w przypadku transformera wizyjnego.

6.6. Skuteczność architektury wykorzystującej zaproponowane połączenie

Skuteczność rozwiązania wykorzystującego dane pochodzące z pośrednich warstw splotowych została przedstawiona na rysunku 6.18.



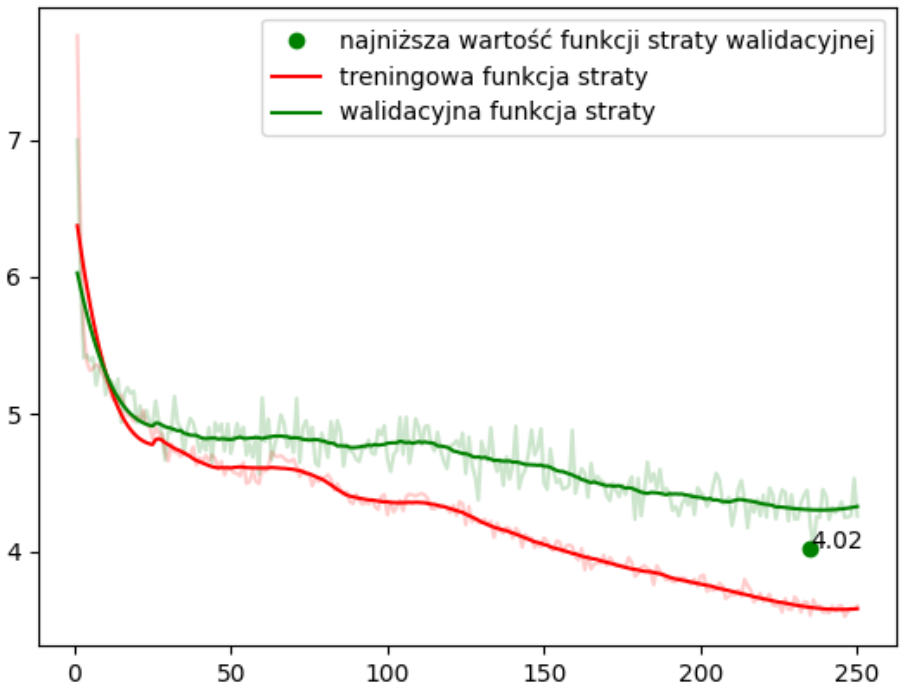
(a) Wartości metryki BLEU

(b) Wartości metryki CIDEr

Rysunek 6.18. Wartości metryk BLEU oraz CIDEr dla poszczególnych kombinacji modułów kodujących i dekodujących wykorzystujących zaproponowane połączenie. Opracowanie własne

Ponownie można zauważyć, iż sieć RNN nie poradziła sobie z tym zadaniem i nie osiągnęła sensownych rezultatów – dla każdej architektury splotowej wartość metryk wyniosła zero. W przypadku pozostałych modułów dekodujących otrzymane wyniki są do siebie zbliżone. Wyniki poszczególnych sieci splotowych są na bardzo zbliżonym poziomie, w przeciwieństwie do architektury z klasycznym połączeniem. Taki stan rzeczy wynika z faktu, iż wektor wyjściowy każdego modułu kodującego był takiego samego rozmiaru – w przypadku architektury klasycznego połączenia wektory te różniły się rozmiarem, co miało bezpośredni wpływ na skuteczność danej architektury. W przypadku wykorzystania danych z warstw pośrednich każdy wektor był długości 4096. Taki rezultat sugeruje, iż bardzo istotne jest zawarcie jak największej ilości danych przekazywanych do modułu dekodującego.

Na podstawie krzywych funkcji straty można zauważyć, iż w przypadku wykorzystania danych z warstw splotowych różnice pomiędzy wartościami zbioru walidacyjnego i treningowego są znacznie większe niż w przypadku klasycznego połączenia modułów, co można zaobserwować na rysunku 6.19, który przedstawia dane treningowe architektury składającej się z sieci AlexNet oraz GRU.



Rysunek 6.19. Wykres wartości funkcji straty treningowej oraz walidacyjnej dla architektury składającej się z sieci AlexNet i GRU. Opracowanie własne.

Mimo iż najlepsza wartość funkcji straty została osiągnięta przy końcu treningu, to już od połowy wartość walidacyjna nie spadała w takim samym tempie, co wartość funkcji straty na zbiorze treningowym. Może to świadczyć o zbyt małej generalizacji danych wejściowych. Można wysunąć wniosek, iż wykorzystywanie danych z warstw splotowych nie jest tak efektywne, ponieważ są to dane zbyt ogólne, niezawierające kluczowych informacji o charakterystyce obrazu.

Podpisy wygenerowane przez większość kombinacji zawierających GRU lub LSTM jako moduł dekodujący wraz z pozostałym modułami kodującymi są bardzo podobne do siebie, co jest przewidywalnym rezultatem na podstawie wartości metryk, jak również krzywych funkcji straty dla tychże rozwiązań. Przykładowe podpisy wygenerowane przez architekturę wykorzystującą AlexNet wraz z LSTM zostały przedstawione na rysunku 6.20.

Rysunek 6.20. Przykładowe obrazki wraz z podpisami wygenerowanymi przez architekturę wykorzystującą sieć VGG i GRU połączone przy pomocy rozwiązania wykorzystującego dane z pośrednich warstw splotowych. Opracowanie własne.

Można zauważyć, iż wygenerowane podpisy tworzą pełne zdania, jednakże znacząco odbiegają one od kontekstu zawartego na zdjęciach. Pojedyncze wyrazy faktycznie korespondują do obiektów znajdujących się na fotografiach, ale ich połączenie w logiczną całość jest nieodpowiednie. Taki stan rzeczy sugeruje, iż być może moduły kodujące za bardzo skupiają się na wyodrębnianiu informacji dotyczących identyfikacji konkretnych obiektów, a zbyt małe zaawansowanie modułów dekodujących nie pozwala na zbyt zaawansowane zinterpretowanie związków przyczynowo skutkowych znajdujących się na obrazkach.

6.7. Porównanie z innymi rozwiązaniami

Najlepszą z architektur łączących bezpośrednio moduły kodujące i dekodujące okazało się połączenie transformera wizyjnego wraz z klasycznym transformerem, które wykorzystywało proste połączenie modułów. W przypadku połączenie uwzględniającego dane z warstw splotowych najlepszym rozwiązaniem okazało się połączeniem sieci ResNet oraz Transformera, jednakże metryki przez nie otrzymane były prawie dwukrotnie mniejsze niż wcześniej wspomniane rozwiązanie. Mimo to wartości metryk otrzymane przy pomocy obu rozwiązań mocno odstają od tych otrzymanych przy pomocy dedykowanych rozwiązań takich jak GIT oraz OFA, co zostało przedstawione w tabeli 6.1. W przypadku rozwiązania GIT oraz OFA zostały tam umieszczone wyniki podane bezpośrednio przez autorów, jak również te otrzymane przy pomocy udostępnionych wstępnie wyuczonych modeli, które zostały wygenerowane poprzez wykorzystanie tego samego zbioru danych testowych, jak w przypadku pozostałych rozwiązań wykorzystujących połączenie modułu kodującego oraz dekodującego, co pozwoliło na bezpośrednie porównanie tychże rozwiązań, co okazuje się kluczowe ze względu na spore rozbieżności wyników otrzymanych przez autorów oraz w drodze testowania.

Tabela 6.1. Porównanie skuteczności rozwiązania wykorzystującego VIT oraz transformera wizyjnego z innymi rozwiązaniami.

Rozwiązanie	**BLEU**	**METEOR**	**CIDEr**	**SPICE**
VIT + Transformer	0,17	0,18	0,42	0,10
GIT (test)	0,19	0,17	0,58	0,13
GIT (artykuł)	0,40	0,30	1,31	0,23
OFA (test)	0,33	0,28	1,03	0,21
OFA (artykuł)	0,43	0,32	1,45	0,25

Wartości metryk otrzymane dla architektury GIT w przypadku BLEU oraz METEOR są nieznacznie wyższe od rozwiązania VIT + Transformer. W przypadku metryk CIDEr i SPICE różnice są bardziej zauważalne, co może świadczyć o lepszej zdolności GIT do zachowania sensu i semantyki w generowanych podpisach. Jednakże porównując wyniki GIT z tymi podanymi bezpośrednio przez autorów architektury, zauważalne są znaczne różnice. Wartości wszystkich metryk dla testowanego wariantu są prawie dwukrotnie mniejsze

niż te, które podali autorzy. Takie rozbieżności mogą wskazywać na słabą zdolność architektury do generalizacji na różne zbiory danych, ponieważ wyniki przez nich podane zostały otrzymane przy pomocy zbioru MS COCO, natomiast zbiorem wykorzystanym do otrzymania testowych wyników był Flickr8k. Na tak duże różnice również mogły mieć specyficzne warunki testowe lub wadliwość modelu przekazanego przez autorów. W przypadku architektury OFA, różnice między wartościami otrzymanymi w badaniach a tymi zawartymi w artykule są niewielkie. Wartości wszystkich metryk podane przez autorów są większe o kilka setnych, co sugeruje, że OFA utrzymuje stabilność wyników i skuteczność w różnych kontekstach. Może to wskazywać na lepszą generalizację i niezawodność architektury OFA w porównaniu do GIT, która również osiągnęła największe wartości w przypadku wszystkich podanych metryk. Jest to przewidywalnym rezultatem, ponieważ OFA została stworzona z myślą o generalizacji i zdolności rozwiązywania wielu różnych zadań z dziedziny przetwarzania obrazu oraz generowania tekstu.

Patrząc na to, iż tak duża przepaść dzieli rozwiązania wykorzystujące proste połączenie modułów kodującego i dekodującego do dedykowanych rozwiązań, można wysunąć wniosek, iż kluczowym jest odpowiednie wykorzystanie danych wejściowych. Dedykowane rozwiązania interpretują obraz jako sekwencję, co pozwala na bardziej złożoną analizę obrazu, a co za tym idzie na bardziej złożone generowanie podpisów. W przypadku wykorzystania klasycznego połączenia modułów kodującego i dekodującego obraz jest interpretowany jako pojedynczy element, co ogranicza możliwości analizy obrazu, ale było konieczne ze względu na rodzaj danych wejściowych sieci rekurencyjnych. W przypadku wykorzystania transformera, mimo jego możliwości przetwarzania sekwencji oraz wykorzystywania modułu atencji, w celu utrzymania modularnej architektury, danymi wejściowymi również był pojedynczy element sekwencji.

7. Wnioski

Najlepszym rozwiązaniem spośród wszelkich kombinacji modułów kodujących oraz dekodujących okazało się najbardziej zaawansowane połączenie, czyli transformer wizyjny wraz z klasycznym transformerem. Niestety takie rozwiązanie i tak odstaje od kompleksowych architektur stworzonych z myślą o generowaniu podpisów do obrazów, takich jak GIT i OFA. O wiele lepsza skuteczność tychże rozwiązań jednocześnie wiąże się z o wiele większymi wymaganiami sprzętowymi, które graniczą z możliwością ich wykorzystania. Szczególnie dotkliwe jest wymaganie alokacji dużej ilości pamięci, co skutkuje brakiem możliwości wykorzystania mniej zaawansowanych wersji jednostek GPU. Jest to główne ograniczenie, ponieważ w przypadku sieci, które były w stanie zostać uruchomione za pomocą mniejszej karty graficznej zostały przetworzone w podobnym czasie do tego przy wykorzystaniu znacznie bardziej zaawansowanej jednostki GPU. W przypadku dedykowanych architektur zarówno czas uczenia, jak i generowania podpisów jest znacząco większy niż w przypadku wykorzystywania mniej złożonych sieci. Pokazuje to, iż rozwój sztucznej inteligencji kieruje się w stronę coraz bardziej rozbudowanych i obciążających rozwiązań. Otrzymanie wyników bez wykorzystywania wstępnie wyuczonych modeli staje się coraz trudniejsze ze względu na potrzebę posiadania wielkiego zbioru danych oraz zasobów obliczeniowych. W przypadku architektury GIT czy OFA warto również pamiętać, że wstępnie wyuczone modele nie rozwiązują problemu późniejszych wymagań obliczeniowych potrzebnych w trakcie docelowego generowania podpisów. Jednoznacznie można stwierdzić, iż podstawowa sieć rekurencyjna nie jest w stanie sprostać zadaniu generowania podpisów do obrazów, a jej bardziej zaawansowani następcy jak LSTM czy GRU są o wiele lepszym wyborem. W przypadku sieci splotowych wybór najlepszego rozwiązania nie jest tak oczywisty. Wraz z lepszą skutecznością rosną wymagania sprzętowe. Tak duże wymagania obliczeniowe, jakie występują w procesie uczenia maszynowego, wywierają znaczący wpływ na czas potrzebny na wyuczenie modelu, co może stanowić jedno z kluczowych ograniczeń w dalszym rozwoju sztucznej inteligencji. Proces szkolenia modeli maszynowych wymaga ogromnych nakładów czasu zarówno na przetwarzanie, jak i adaptację do coraz bardziej rosnących zbiorów danych. Model uczenia maszynowego musi być zdolny do efektywnego przetwarzania ogromnych ilości informacji, a jednocześnie szybko dostosowywać się do zmieniających się warunków. Owa potrzeba adaptacji jest szczególnie ważna w kontekście dynamicznie rozwijającego się świata, gdzie ilość generowanych danych z dnia na dzień znacząco rośnie. Dzięki temu modele mogą lepiej reprezentować rzeczywistość i dostosowywać się do zmieniających się wzorców. Otrzymane wyniki w kwestii czasu potrzebnego na przetwarzanie architektur sieci neuronowych, jak również wymagania sprzętowe pokazują, iż może stanowić to duże ograniczenie w dalszym rozwoju tychże rozwiązań. Warto również wspomnieć, iż autorzy wspomnianych rozwiązań OFA, GIT czy FNIC szeroko stosują techniki transferu wiedzy, wykorzystując wstępnie wytrenowane modele na ogromnych zbiorach danych. Podejście to zyskuje na wielkiej popularności i można stwierdzić, że ze względu na coraz większe skomplikowanie rozwiązywanych zadań, jak również dążenie do generalizacji problemów

stawianych przed architekturami algorytmów uczenia maszynowego, technika transferu wiedzy będzie odgrywać coraz większą rolę w dalszym rozwoju sztucznej inteligencji. Taki kierunek rozwoju może stanowić spory problem w przypadku chęci posiadania pełnej kontroli nad wykorzystywanym modelem, ponieważ w przypadku korzystania z wag architektury z innego źródła może brakować nam informacji w przy pomocy jakich dokładnie danych owe wagi zostały wygenerowane, a być może mogły być to dane pochodzące z nielegalnych źródeł.

7.1. Perspektywy rozwoju

Ze względu na ograniczenia czasowe nie było możliwe zweryfikowanie, jak duży wpływ ma równoczesne trenowanie modułu kodującego wraz z modułem dekodującym na skuteczność otrzymanych rozwiązań. Można przypuszczać, iż skuteczność w pewnym stopniu by wzrosła, ale są to jedynie domysły, które warte są weryfikacji. W przypadku wykorzystywania danych z pośrednich warstw splotowych, wytrenowanie modelu od zera mogłoby zaowocować znacznie lepszymi wynikami, ponieważ architektura minimalizowałaby funkcję straty, biorąc pod uwagę szerszy zakres danych wyjściowych, a nie tak jak odbywa się to w przypadku klasycznego podejścia, jedynie dane pochodzące z ostatnich warstw. Bardzo sporym usprawnieniem całej architektury byłoby wykorzystanie mechanizmu atencji, co nie zostało zaimplementowane ze względu na ograniczenie się do prostych sieci rekurencyjnych. Być może wartym zweryfikowania byłyby również inne architektury modułów kodujących i dekodujących stworzone z myślą o większej wydajności.

Bibliografia

[1] A. Mammone, M. Turchi i N. Cristianini, „Support vector machines”, *Wiley Interdisciplinary Reviews: Computational Statistics*, t. 1, nr. 3, s. 283–289, 2009.

[2] J. L. Elman, „Finding structure in time”, *Cognitive science*, t. 14, nr. 2, s. 179–211, 1990.

[3] t. f. e. Wikipedia, *Recurrent neural network*, Dostęp zdalny (01.05.2023): `https://commons.wikimedia.org/wiki/File:Recurrent_neural_network_unfold.svg`, 2023.

[4] S. Hochreiter i J. Schmidhuber, „Long short-term memory”, *Neural computation*, t. 9, nr. 8, s. 1735–1780, 1997.

[5] K. Cho, B. van Merrienboer, C. Gulcehre i in., *Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation*, 2014. arXiv: `1406.1078 [cs.CL]`.

[6] D. Bahdanau, K. Cho i Y. Bengio, *Neural Machine Translation by Jointly Learning to Align and Translate*, 2016. arXiv: `1409.0473 [cs.CL]`.

[7] A. Vaswani, N. Shazeer, N. Parmar i in., „Attention is all you need”, *Advances in neural information processing systems*, t. 30, 2017.

[8] P. Xu, D. Kumar, W. Yang i in., *Optimizing Deeper Transformers on Small Datasets*, 2021. arXiv: `2012.15355 [cs.CL]`.

[9] t. f. e. Wikipedia, *convolutional neural network*, Dostęp zdalny (01.05.2023): `https://en.wikipedia.org/wiki/File:Typical_cnn.png`, 2023.

[10] E. Polański, *Wielki słownik ortograficzny PWN z zasadami pisowni i interpunkcji*. Polska: PWN, 2016.

[11] K. Simonyan i A. Zisserman, „Very deep convolutional networks for large-scale image recognition”, *arXiv preprint arXiv:1409.1556*, 2014.

[12] A. Dosovitskiy, L. Beyer, A. Kolesnikov i in., „An image is worth 16x16 words: Transformers for image recognition at scale”, *arXiv preprint arXiv:2010.11929*, 2020.

[13] Z. Zohourianshahzadi i J. K. Kalita, „Neural attention for image captioning: review of outstanding methods”, *Artificial Intelligence Review*, t. 55, nr. 5, s. 3833–3862, 2022.

[14] O. Vinyals, A. Toshev, S. Bengio i D. Erhan, „Show and tell: A neural image caption generator”, w *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, s. 3156–3164.

[15] S. Ioffe i C. Szegedy, *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*, 2015. arXiv: `1502.03167 [cs.LG]`.

[16] X. Chen, H. Fang, T.-Y. Lin i in., „Microsoft coco captions: Data collection and evaluation server”, *arXiv preprint arXiv:1504.00325*, 2015.

[17] J. Wang, Z. Yang, X. Hu i in., *GIT: A Generative Image-to-text Transformer for Vision and Language*, 2022. arXiv: `2205.14100 [cs.CV]`.

[18] L. Yuan, D. Chen, Y.-L. Chen i in., *Florence: A New Foundation Model for Computer Vision*, 2021. arXiv: `2111.11432 [cs.CV]`.

[19] P. Sharma, N. Ding, S. Goodman i R. Soricut, „Conceptual captions: A cleaned, hypernymed, image alt-text dataset for automatic image captioning", w *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2018, s. 2556–2565.

[20] V. Ordonez, G. Kulkarni i T. Berg, „Im2text: Describing images using 1 million captioned photographs", *Advances in neural information processing systems*, t. 24, 2011.

[21] R. Krishna, Y. Zhu, O. Groth i in., „Visual genome: Connecting language and vision using crowdsourced dense image annotations", *International journal of computer vision*, t. 123, s. 32–73, 2017.

[22] S. Changpinyo, P. Sharma, N. Ding i R. Soricut, „Conceptual 12m: Pushing web-scale image-text pre-training to recognize long-tail visual concepts", w *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, s. 3558–3568.

[23] X. Hu, Z. Gan, J. Wang i in., „Scaling up vision-language pre-training for image captioning", w *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2022, s. 17 980–17 989.

[24] P. Wang, A. Yang, R. Men i in., *OFA: Unifying Architectures, Tasks, and Modalities Through a Simple Sequence-to-Sequence Learning Framework*, 2022. arXiv: 2202.03052 [cs.CV].

[25] L. Zhou, H. Palangi, L. Zhang, H. Hu, J. Corso i J. Gao, „Unified vision-language pre-training for image captioning and vqa", w *Proceedings of the AAAI conference on artificial intelligence*, t. 34, 2020, s. 13 041–13 049.

[26] Z.-c. Fei, „Fast image caption generation with position alignment", *arXiv preprint arXiv:1912.06365*, 2019.

[27] N. Wang, J. Xie, J. Wu, M. Jia i L. Li, „Controllable image captioning via prompting", w *Proceedings of the AAAI Conference on Artificial Intelligence*, t. 37, 2023, s. 2617–2625.

[28] J. Devlin, M.-W. Chang, K. Lee i K. Toutanova, „Bert: Pre-training of deep bidirectional transformers for language understanding", *arXiv preprint arXiv:1810.04805*, 2018.

[29] X. Li, G. Zhang, H. H. Huang, Z. Wang i W. Zheng, „Performance analysis of GPU-based convolutional neural networks", w *2016 45th International conference on parallel processing (ICPP)*, IEEE, 2016, s. 67–76.

[30] J. Appleyard, T. Kocisky i P. Blunsom, *Optimizing Performance of Recurrent Neural Networks on GPUs*, 2016. arXiv: 1604.01946 [cs.LG].

[31] N. Wang, J. Xie, H. Luo i in., „Efficient image captioning for edge devices", w *Proceedings of the AAAI Conference on Artificial Intelligence*, t. 37, 2023, s. 2608–2616.

[32] A. Deshpande, J. Aneja, L. Wang, A. G. Schwing i D. Forsyth, „Fast, Diverse and Accurate Image Captioning Guided by Part-Of-Speech", w *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, czer. 2019.

[33] H. Rampal i A. Mohanty, *Efficient CNN-LSTM based Image Captioning using Neural Network Compression*, 2020. arXiv: 2012.09708 [cs.CV].

[34] A. Krizhevsky, I. Sutskever i G. E. Hinton, „ImageNet Classification with Deep Convolutional Neural Networks", *Advances in neural information processing systems*, t. 25, s. 1097–1105, 2012.

[35] K. He, X. Zhang, S. Ren i J. Sun, „Deep residual learning for image recognition", w *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, s. 770–778.

[36] C. Szegedy, W. Liu, Y. Jia i in., „Going Deeper with Convolutions", *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[37] V. Sanh, L. Debut, J. Chaumond i T. Wolf, *DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter*, 2020. arXiv: 1910.01108 [cs.CL].

[38] K. Papineni, S. Roukos, T. Ward i W.-J. Zhu, „Bleu: a method for automatic evaluation of machine translation", w *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, 2002, s. 311–318.

[39] S. Banerjee i A. Lavie, „METEOR: An automatic metric for MT evaluation with improved correlation with human judgments", w *Proceedings of the acl workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization*, 2005, s. 65–72.

[40] R. Vedantam, C. Lawrence Zitnick i D. Parikh, „Cider: Consensus-based image description evaluation", w *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, s. 4566–4575.

[41] G. Salton i C. Buckley, „Term-weighting approaches in automatic text retrieval", *Information processing & management*, t. 24, nr. 5, s. 513–523, 1988.

[42] A. Paszke, S. Gross, F. Massa i in., „PyTorch: An Imperative Style, High-Performance Deep Learning Library", w *Advances in Neural Information Processing Systems 32*, Curran Associates, Inc., 2019, s. 8024–8035. adr.: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[43] T. Wolf, L. Debut, V. Sanh i in., „Transformers: State-of-the-Art Natural Language Processing", w *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, Online: Association for Computational Linguistics, paź. 2020, s. 38–45. adr.: https://www.aclweb.org/anthology/2020.emnlp-demos.6.

[44] P. Young, A. Lai, M. Hodosh i J. Hockenmaier, „From image descriptions to visual denotations: New similarity metrics for semantic inference over event descriptions", *Transactions of the Association for Computational Linguistics*, t. 2, s. 67–78, 2014.

Spis rysunków