Raport – Uczenie Maszynowe

Piotr Tutak, 286260, Informatyka Techniczna – Niestacjonarne

1. Cel projektu

Celem projektu jest stworzenie modelu rozpoznawania obrazów, który będzie rozpoznawał i odróżniał od siebie wymienione rodzaje obrazów:

- ryba
- pies
- radio
- piła
- kościół
- waltornia
- śmieciarka
- piłka golfowa
- stacja benzynowa
- spadochron

Użytym zbiorem danych do analizy był Imagenette ze strony:
https://github.com/fastai/imagenette

Dodatkowym celem jest porównanie najlepszych uzyskanych wyników z wynikami sieci GoogLeNet, która wygrała konkurs ILSVRC w 2014 roku na najlepszą architekturę do rozpoznawania obrazów.

2. Użyte narzędzia

Narzędzia, w których został zrealizowany projekt to przede wszystkim MATLAB w wersji 2018b wraz z dodatkowym pakietem: Deep Learning Toolbox.

3. Opis metody

Obecnie do skutecznego rozpoznawania obrazów używa się w zasadzie jedynie sztucznych głębokich sieci neuronowych, dlatego też opis metody skupi się przede wszystkim na tej metodzie.

3.1. Głębokie sieci neuronowe

Wyróżniamy różne rodzaje głębokich sieci neuronowych, do najważniejszych z nich należą:

- Sieci konwolucyjne (Convolutional Neural Networks – CNN)
- Sieci rekurencyjne (Long Short Term Memory – LSTM)

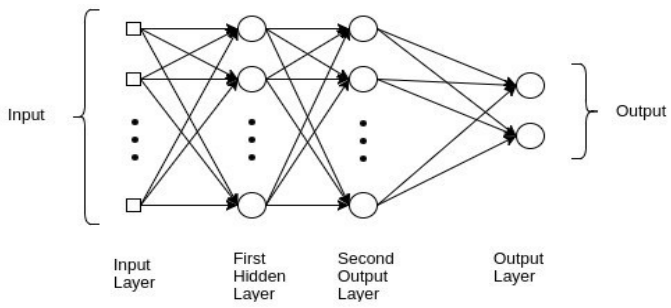
W rozpoznawaniu obrazów obecnie największą popularnością cieszą się sieci konwolucyjne. Obecnie wszystkie architektury biorące udział w corocznym konkursie rozpoznawania obrazów opierają się w jakimś stopniu na głębokich sieciach konwolucyjnych.

3.2. Sieci konwolucyjne

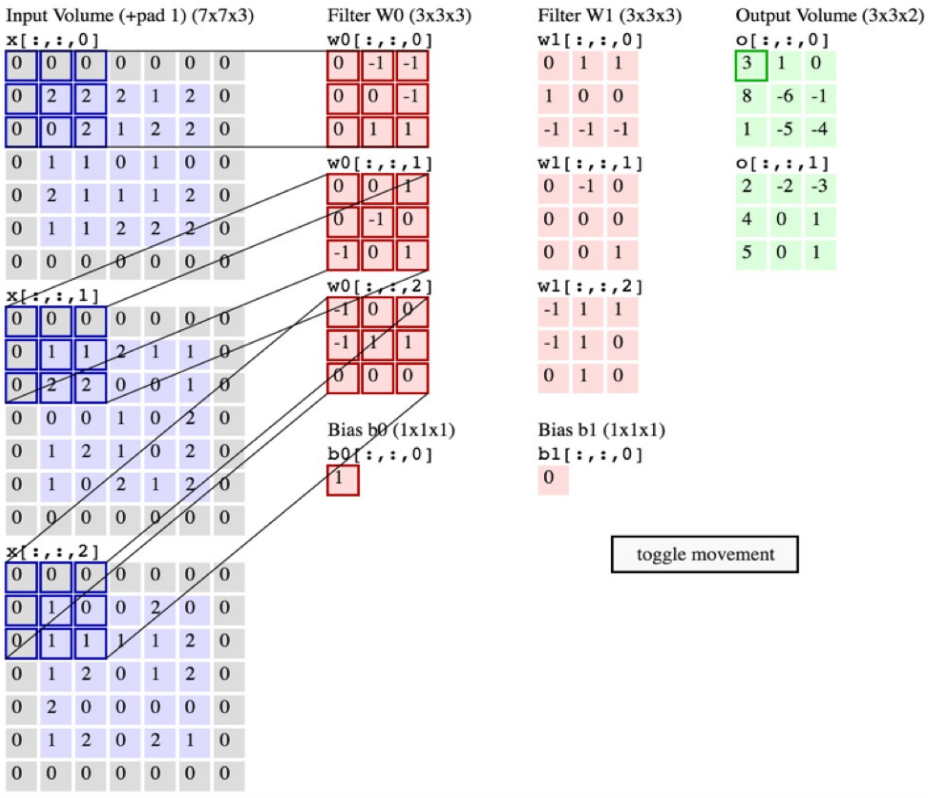
Sieci konwolucyjne, jak sama nazwa wskazuje, opierają się w dużej mierze na operacji „konwolucji”. Dodatkowymi operacjami, które są przeprowadzane w takich sieciach są najcześciej tak zwany „pooling”, oraz normalizacja.

a) Konwolucja

Operacja konwolucji to najważniejsza operacja przeprowadzana w sieciach konwolucyjnych. W tradycyjnych sieciach neuronowych takich jak wielowarstwowy perceptron, każdy neuron z następnej warstwy jest połączony z każdym neuronem z poprzedniej warstwy. W przypadku warstw o dużej ilości wejść powoduje to problemy związane z wariancją, czyli sieć szybko się przeucza z powodu zbyt dużej liczby parametrów. Sieć nie generalizuje, ale uczy się szczegółów. Kolejnym problemem jest bardzo duża liczba obliczeń, które trzeba wykonać przy przetwarzaniu danych z warstwy na warstwę. Te problemu ujawniają się szczególnie w przypadku obrazów, gdzie mamy bardzo dużo neuronów wejściowych (każdy piksel jest wejściem) oraz często wiele kanałów kolorów (np. RGB).



Sieć konwolucyjna opiera się na innym założeniu. Zamiast uczyć wszystkie wagi dla wszystkich neuronów tworzy się „wagi ruchome” zwane „filtrami”, które są współdzielone przez wszystkie neurony w danej warstwie, o mniejszym rozmiarze. Przykłady filtrów:



Każdy filtr dokonuje operacji dla wszystkich neuronów wejściowych, jeśli wejścia posiadają 3 kanały, to i filtr posiada trzy kanały, jak w przypadku pokazanym powyżej.

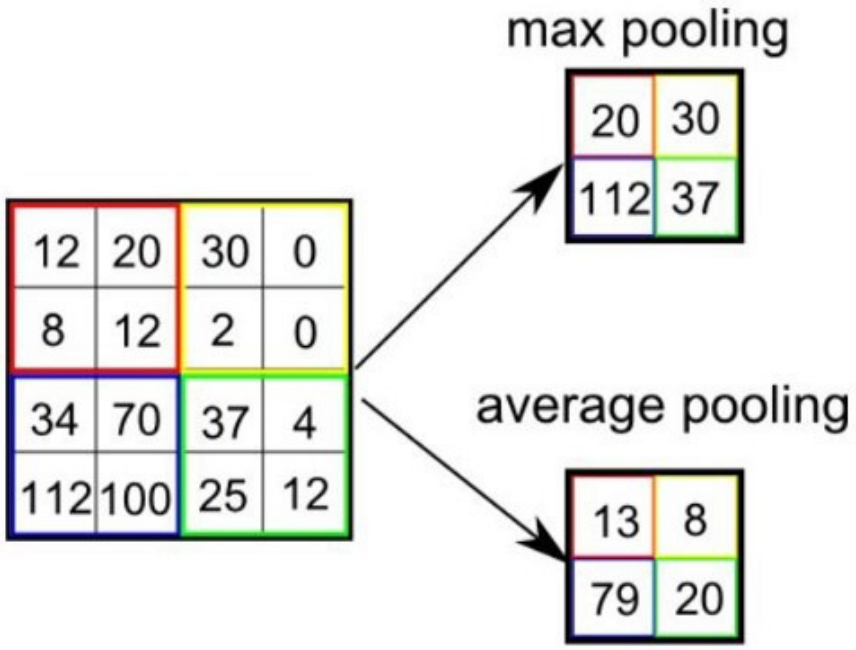
Na tym przykładzie widać również pewną szczególną właściwość operacji konwolucji, mianowicie obraz wyjściowy, po operacji konwolucji, bez dodatkowych innych przekształceń, będzie mniejszy, niż obraz wejściowy. Dla przykładu, filtr o rozmiarze 3 x 3 dokona operacji konwolucji na obrazie wejściowym o rozmiarze 3 x 3 tylko raz, czego efektem będzie obraz o rozmiarze 1 x 1. By zachować rozmiar wejściowy niezmieniony, należy dokonać operacji „paddingu”, czyli odpowiedniego powiększenia obrazu wejściowego tak, by obraz wyjściowy był tych samych rozmiarów jak obraz wejściowy. Najczęściej padding polega na powiększeniu obrazu wejściowego o obramowanie złożone z zer. Innym ważnym parametrem operacji konwolucji jest tak zwany „stride”, czyli ilość pikseli, o którą należy przesunąć filtr, w przechodzeniu z jednej operacji konwolucji na drugą. Stride równy 2 oznacza, że operacja konwolucji będzie omijać 1 piksel przy przesuwaniu filtra. Efektem tego będzie też mniejszy obraz wyjściowy (dokładnie 4 razy mniejszy, po 2 razy wzdłuż każdej z osi).

b) Pooling

Kolejną operacją, bardzo często dokonywaną w sieciach konwolucyjnych, jest operacja „poolingu”. Jest kilka rodzajów „poolingu”, ale najważniejsze, i najczęściej używane są dwa: „max pooling” i „average pooling”. Pooling polega na obliczeniu wartości z pewnej z góry określonej grupy pikseli i przekazanie ich jako wartości wejściowych dla kolejnej warstwy.

Max Pooling polega na wybraniu wartości maksymalnej z określonej grupy pikseli, average pooling polega na obliczeniu wartości średniej z pewnej grupy pikseli.

Grupę pikseli określa się najczęściej jako rozmiar kwadratu obejmującego piksele w danym obrazie. Max pooling o rozmiarze 2 oznacza wybranie wartości największej z kwadratu 2 x 2.



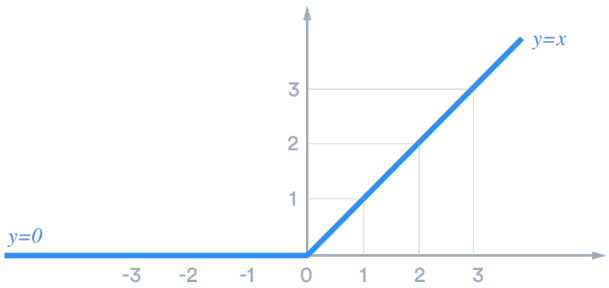
Pooling podobnie jak „stride”, powoduje pomniejszenie obrazu wyjściowego.

c) Normalizacja

Normalizacja jest używana by uzyskać stabilność procesu uczenia i uniezależnić nieco wagi każdej warstwy od warstw sąsiednich. Polega ona na odjęciu wartości średnich funkcji aktywacji neuronów z grupy przetwarzanych aktualnie przykładów, zwanych wsadem (*batch)* użytych w procesie uczenia, od wartości wag wejściowych następnej warstwy i podzieleniu ich przez odchylenie standardowe wartości funkcji aktywacji danego wsadu.

d) Funkcja aktywacji

Funkcja aktywacji to funkcja, która jest obliczana na wyjściu warstwy neuronów. Wynik funkcji aktywacji jest przekazywany do następnej warstwy. Najczęściej używaną funkcją aktywacji jest funkcja ReLU, która w ostatni czasie zyskała ogromną popularność i wykazuje bardzo dobre wyniki w procesie uczenia. Jest to połączenie funkcji liniowej stałej:



e) Optymalizator

W procesie uczenia stosowane są różne optymalizatory, jednakże w tym projekcie wybrane zostały dwa najbardziej podstawowe i popularne:

- SGDM
- ADAM

Optymalizator ADAM jest określany jako połączenie dwóch innych optymalizatorów, czyli AdaGrad i RMSProp. Optymalizator jest odpowiedzialny w procesie uczenia, za aktualizację wag sieci neuronowej.

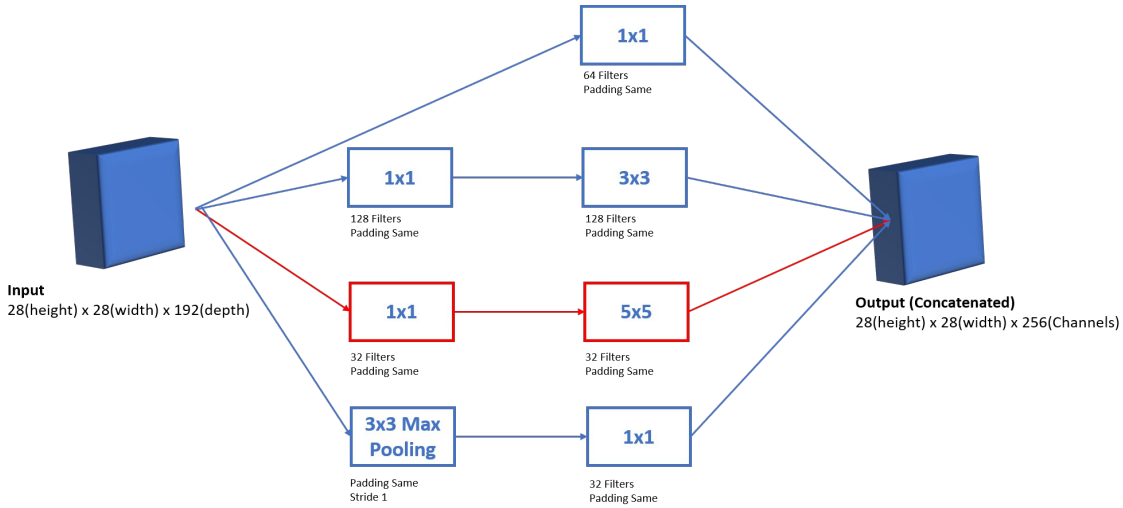
3.3. GoogLeNet

GoogLeNet to bardzo znana sieć neuronowa, która zwyciężyła w ILSVRC w 2014 roku na najlepszą architekturę. Charakteryzuje się bardzo szybkimi czasami odpowiedzi na zadane wartości wejściowe z jednoczesną bardzo dużą dokładnością.

To wszystko zostało osiągnięte dzięki zastosowaniu tzw. modułu incepcyjnego.

Moduł incepcyjny to połączone ze sobą warstwy sieci neuronowej nie szeregowo, ale równolegle. Pomaga on w reprezentacji różnych właściwości obrazu bez tracenia informacji z jednoczesnym zachowaniem bardzo dużej szybkości przetwarzania dzięki redukcji wymiarów. Innowacją zastosowaną w tym module jest używanie konwolucji o rozmiarze 1 (1 x 1) w celu redukcji wymiarów przed konwolucją o rozmiarze 5 (5 x 5). Umożliwia to znaczne przyspieszenie obliczeń.

Poglądowy schemat modułu incepcyjnego zastosowanego w GoogLeNet jest:



Głównym zadaniem wytrenowania sieci GoogLeNet w projekcie jest punkt odniesienia. Można ją wraz z wagami w prosty sposób zaimportować w MATLAB'ie i wytrenować tylko ostatnie warstwy w celu dostosowania do nowego zbioru danych.

Konstrukcja sieci GoogLeNet z podziałem na poszczególne warstwy przedstawia się następująco:

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

Warstwy o nazwie „inception” to wspomniane już wyżej moduły incepcyjne.

4. Realizacja

Realizacja projektu została wykonana w programie MATLAB przy użyciu dodatku Deep Learning Toolbox. Wszystkie pliki ze zbioru uczącego zostało podzielone na 2 zbiory. Zbiór uczący i zbiór walidacyjny w stosunku 70 : 30. Projekt użytej sieci neuronowej był wzorowany na sieciach VGG używanych w konkursach ILSVRC (ImageNet Large-Scale Visual Recognition Challenge).

W sieciach były używane głównie filtry konwolucyjne o rozmiarze 3 (pole 3 x 3), warstwy max-pooling o rozmiarze 2. Wszystkie funkcje aktywacji użyte w sieci to funkcje ReLU.

Po każdej warstwie konwolucji i przed funkcją aktywacji została umieszczana warstwa normalizacyjna.

W przypadku niektórych sieci wobec wszystkich obrazów w procesie uczenia zostały stosowane losowe przesunięcia oraz powiększenia w celu zwiększenia efektywnego zbioru danych, oraz po to, by uniknąć przeuczania się sieci.

W procesie uczenia stosowana była regularyzacja L2, czyli kara zastosowana do funkcji celu w przypadku dużych wartości wag użytych filtrów. Ma to zapobiegać problemowi z wariancją, czyli przetrenowaniu sieci.

Wartość współczynnika uczenia była dostosowywana z czasem i pomniejszana o odpowiedni procent wartości wyjściowej, wraz z postępującym procesem uczenia sieci.

Sieci miały różną liczbę warstw oraz różną liczbę użytych filtrów.

5. Analiza wyników

Wyniki sieci GoogLeNet po podmienieniu wag ostatnich dwóch warstw i wyuczeniu sieci uplasowały się w wartościach 98,6%.

Wyniki poszczególnych sieci są widoczne poniżej.

Wyróżnione zostały następujące parametry:

- optimizer – użyty optymalizator
- learn_rate – początkowy współczynnik uczenia
- patience – liczba epok po której, jeśli nie nastąpi poprawa wyników uczenie sieci dobiega końca
- drop_period – liczba epok po której współczynnik uczenia zostaje pomnożony przez drop_rate_factor
- drop_rate_factor – współczynnik zmniejszający współczynnik uczenia poprzez mnożenie
- image_augmentation – wartość określająca czy wobec obrazów wejściowych zostały zastosowane dodatkowe operacje na wejściu jak translacja, powiększenie o wartości losowe z zakresów:
  - skalowanie: [0.9000 1.1000]
  - translacja: [-30 30] – wartości podane w pikselach

name	optimizer	learn_rate	patience	drop_period	drop_rate_factor	image_augmentation	accuracy
network_64_0764	SGDM	0,001	5	4	0,5	No	0,6408
network_65_6561	ADAM	0,0001	5	2	0,5	No	0,6566
network_71_6943	SGDM	0,001	3	3	0,5	Yes	0,7169
network_74_1656	ADAM	0,0001	5	2	0,5	No	0,7417
network_74_2675	ADAM	0,0001	4	4	0,65	Yes	0,7427
network_75_4395	ADAM	0,0001	5	2	0,5	No	0,7544
network_79_0828	SGDM	0,001	4	4	0,65	Yes	0,7908
network_82_242	SGDM	0,001	4	5	0,65	Yes	0,8224
network_82_3185	SGDM	0,001	4	6	0,65	Yes	0,8232
network_82_828	ADAM	0,0001	3	3	0,65	Yes	0,8283
network_83_1338	SGDM	0,001	3	3	0,5	Yes	0,8313
network_83_2611	SGDM	0,001	4	5	0,65	Yes	0,8326
network_83_949	SGDM	0,001	4	4	0,65	Yes	0,8395
network_84_3312	SGDM	0,001	4	5	0,65	Yes	0,8433
network_84_8153	SGDM	0,001	4	4	0,65	Yes	0,8482

Na podstawie tabeli, na pierwszy rzut oka można stwierdzić, że optymalizator ADAM gorzej sobie radzi niż SGDM. Jednakże należy pamiętać, że sieci różnią się od siebie nie tylko parametrami uczenia, ale też konstrukcją, czyli tym z jakich warstw są zbudowane.

Dla optymalizatora ADAM został zastosowany mniejszy współczynnik uczenia oraz krótszy czas na zmniejszenie tego współczynnika, gdyż ten optymalizator znacznie szybciej dochodzi do ustalonego wyniku dla danej wartości współczynnika uczenia niż SGDM.

Widać również, że dodanie losowych przekształceń do obrazów zdecydowanie poprawia wyniki sieci. Również współczynnik, który wpływał na zmniejszenie współczynnika uczenia o 35% zamiast 50% okazał się bardziej optymalny.

Poszczególne sieci różnią się od siebie ilością i rodzajem warstw, poniżej przedstawiono schematy niektórych sieci przedstawionych w tabeli:

network_74_1656	**network_74_2675**	**network_75_4395**	**network_79_0828**	**network_82_242**	**network_82_3185**
ImageInput, 256x256	ImageInput, 224x224	ImageInput, 256x256	ImageInput, 224x224	ImageInput, 224x224	ImageInput, 224x224
Conv, 3x3, 64	Conv, 3x3, 64	Conv, 3x3, 64	Conv, 3x3, 64	Conv, 3x3, 64	Conv, 3x3, 64
Dropout, 0.3	Conv, 3x3, 64	MaxPool, 2x2	Conv, 3x3, 64	Conv, 3x3, 64	Conv, 3x3, 64
MaxPool, 2x2	Conv, 1x1, 64	Conv, 3x3, 128	MaxPool, 2x2	MaxPool, 2x2	MaxPool, 2x2
Conv, 3x3, 128	MaxPool, 2x2	MaxPool, 2x2	Conv, 3x3, 64	Conv, 3x3, 64	Conv, 3x3, 64
MaxPool, 2x2	Conv, 3x3, 64	Conv, 3x3, 256	Conv, 3x3, 64	Conv, 3x3, 64	Conv, 3x3, 64
Conv, 3x3, 256	Conv, 3x3, 64	MaxPool, 2x2	MaxPool, 2x2	MaxPool, 2x2	MaxPool, 2x2
Dropout, 0.3	Conv, 1x1, 64	Conv, 3x3, 512	Conv, 3x3, 128	Conv, 3x3, 128	Conv, 3x3, 128
Conv, 3x3, 256	MaxPool, 2x2	MaxPool, 2x2	Conv, 3x3, 128	Conv, 3x3, 128	Conv, 3x3, 128
MaxPool, 2x2	Conv, 3x3, 128	Conv, 3x3, 512	MaxPool, 2x2	MaxPool, 2x2	MaxPool, 2x2
Conv, 3x3, 512	Conv, 3x3, 128	MaxPool, 2x2	Conv, 3x3, 256	Conv, 3x3, 256	Conv, 3x3, 256
Dropout, 0.3	Conv, 1x1, 128	FullyConnected, 4096	Conv, 3x3, 256	Conv, 3x3, 256	Conv, 3x3, 256
Conv, 3x3, 512	MaxPool, 2x2	FullyConnected, 10	MaxPool, 2x2	MaxPool, 2x2	MaxPool, 2x2
MaxPool, 2x2	Conv, 3x3, 256	SoftMax	Conv, 3x3, 512	Conv, 3x3, 512	Conv, 3x3, 512
Conv, 3x3, 512	Conv, 3x3, 256	ClassOutput, 10	AvgPool, 2x2	Conv, 3x3, 512	AvgPool, 2x2
Dropout, 0.3	Conv, 1x1, 256		FullyConnected, 512	AvgPool, 2x2	FullyConnected, 512
Conv, 3x3, 512	MaxPool, 2x2		Dropout, 0.35	FullyConnected, 512	Dropout, 0.35
MaxPool, 2x2	Conv, 3x3, 512		FullyConnected, 512	Dropout, 0.35	FullyConnected, 512
FullyConnected, 4096	Conv, 3x3, 512		FullyConnected, 10	FullyConnected, 512	FullyConnected, 10
Dropout, 0.3	Conv, 1x1, 512		SoftMax	FullyConnected, 10	SoftMax
FullyConnected, 4096	MaxPool, 2x2		ClassOutput, 10	SoftMax	ClassOutput, 10
FullyConnected, 10	FullyConnected, 768			ClassOutput, 10	
SoftMax	Dropout, 0.35				
ClassOutput, 10	FullyConnected, 768				
	FullyConnected, 10				
	SoftMax				
	ClassOutput, 10				

network_82_828	**network_83_1338**	**network_83_2611**	**network_83_949**	**network_84_3312**	**network_84_8153**
ImageInput, 224x224	ImageInput, 224x224	ImageInput, 224x224	ImageInput, 224x224	ImageInput, 224x224	ImageInput, 224x224
Conv, 3x3, 64	Conv, 3x3, 64	Conv, 3x3, 64	Conv, 3x3, 64	Conv, 3x3, 64	Conv, 3x3, 64
Conv, 3x3, 64	Conv, 3x3, 64	Conv, 3x3, 64	Conv, 3x3, 64	Conv, 3x3, 64	Conv, 3x3, 64
MaxPool, 2x2	Conv, 3x3, 64	MaxPool, 2x2	MaxPool, 2x2	MaxPool, 2x2	MaxPool, 2x2
Conv, 3x3, 64	MaxPool, 2x2	Conv, 3x3, 64	Conv, 3x3, 64	Conv, 3x3, 64	Conv, 3x3, 64
Conv, 3x3, 64	Conv, 3x3, 64	Conv, 3x3, 64	Conv, 3x3, 64	Conv, 3x3, 64	Conv, 3x3, 64
MaxPool, 2x2	Conv, 3x3, 64	MaxPool, 2x2	MaxPool, 2x2	MaxPool, 2x2	MaxPool, 2x2
Conv, 3x3, 128	Conv, 3x3, 64	Conv, 3x3, 128	Conv, 3x3, 128	Conv, 3x3, 128	Conv, 3x3, 128
Conv, 3x3, 128	MaxPool, 2x2	Conv, 3x3, 128	Conv, 3x3, 128	Conv, 3x3, 128	Conv, 3x3, 128
MaxPool, 2x2	Conv, 3x3, 128	MaxPool, 2x2	MaxPool, 2x2	MaxPool, 2x2	MaxPool, 2x2
Conv, 3x3, 256	Conv, 3x3, 128	Conv, 3x3, 256	Conv, 3x3, 256	Conv, 3x3, 256	Conv, 3x3, 256
Conv, 3x3, 256	MaxPool, 2x2	MaxPool, 2x2	Conv, 3x3, 256	Conv, 3x3, 256	Conv, 3x3, 256
MaxPool, 2x2	Conv, 3x3, 256	Conv, 3x3, 512	MaxPool, 2x2	MaxPool, 2x2	MaxPool, 2x2
Conv, 3x3, 512	Conv, 3x3, 256	AvgPool, 2x2	Conv, 3x3, 512	Conv, 3x3, 512	Conv, 3x3, 512
Conv, 3x3, 512	MaxPool, 2x2	FullyConnected, 512	Conv, 3x3, 512	AvgPool, 2x2	Conv, 3x3, 512
AvgPool, 2x2	Conv, 3x3, 512	Dropout, 0.35	AvgPool, 2x2	FullyConnected, 512	AvgPool, 2x2
FullyConnected, 768	Conv, 3x3, 512	FullyConnected, 512	FullyConnected, 1024	Dropout, 0.35	FullyConnected, 768
Dropout, 0.35	AvgPool, 2x2	FullyConnected, 10	Dropout, 0.5	FullyConnected, 512	Dropout, 0.35
FullyConnected, 768	FullyConnected, 1024	SoftMax	FullyConnected, 1024	FullyConnected, 10	FullyConnected, 768
FullyConnected, 10	Dropout, 0.4	ClassOutput, 10	FullyConnected, 10	SoftMax	FullyConnected, 10
SoftMax	FullyConnected, 1024		SoftMax	ClassOutput, 10	SoftMax
ClassOutput, 10	FullyConnected, 10		ClassOutput, 10		ClassOutput, 10
	SoftMax				
	ClassOutput, 10				

Na początku uczenia były stosowane sieci, podobne do sieci VGG z bardzo dużymi warstwami ostatnimi – FullyConnected 4096. Po czasie i pewnych eksperymentach oraz przeanalizowaniu wyników okazało się, że sieć przeucza się pod koniec procesu uczenia, oznaczało to problem z wariancją. W związku z tym szczególnie liczba neuronów ostatniej warstwy została znacznie ograniczona. Została też wprowadzona warstwa DropOut pomiędzy dwiema warstwami sieci z największą liczbą neuronów również po to by zapobiec nadmiernemu przeuczaniu się sieci.

Ostatecznie za optymalną architekturę została uznana sieć z dwiema warstwami konwolucyjnymi po 64, 64, 128, 256, i 512 neuronów, przedzielonych warstwami MaxPool o polu 2 x 2 piksele i przesunięciu „stride" również 2 x 2, zakończona warstwami FullyConnected 768 przedzielonych warstwą DropOut ze współczynnikiem prawdopodobieństwa wyłączenia neuronu równym 0.35.

Ta sieć uzyskała wynik 84,81%. Został w niej zastosowany optymalizator SGDM. Dla porównania najlepsza sieć o takiej samej konstrukcji dla optymalizatora ADAM osiągnęła wynik 82,83%. Dla przypomnienia sieć GoogLeNet uzyskała wynik 98,6%.

6. Wnioski

Na podstawie osiągniętych wyników można stwierdzić, że:

- Sieć GoogLeNet radzi sobie zdecydowanie lepiej niż stworzone sieci
- W przypadku uczenia sieci neuronowych bardzo ważne jest wstępne przetworzenie użytych obrazów, zastosowanie nawet drobnej losowości czy modyfikacji może znacznie poprawić wyniki
- Optymalizator ADAM szybciej się uczy, jednakże ostatecznie uzyskiwane wyniki są zazwyczaj nieco gorsze niż SGDM
- Warto w sieciach stosować warstwy regularyzacyjne takie jak DropOut, szczególnie przy warstwach w pełni połączonych, gdyż zapobiega to przeuczaniu się sieci.

7. Źródła

- https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53
- https://towardsdatascience.com/vgg-neural-networks-the-next-step-after-alexnet-3f91fa9ffe2c
- https://towardsdatascience.com/batch-normalization-in-neural-networks-1ac91516821c
- https://towardsdatascience.com/deep-learning-googlenet-explained-de8861c82765
- https://towardsdatascience.com/deep-learning-understand-the-inception-module-56146866e652
- https://towardsdatascience.com/intuitions-on-l1-and-l2-regularisation-235f2db4c261
- https://medium.com/@sonish.sivarajkumar/relu-most-popular-activation-function-for-deep-neural-networks-10160af37dda
- https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/
- https://towardsdatascience.com/adam-latest-trends-in-deep-learning-optimization-6be9a291375c#:~:text=Adam%20%5B1%5D%20is%20an%20adaptive,for%20training%20deep%20neural%20networks.&text=The%20algorithms%20leverages%20the,learning%20rates%20for%20each%20parameter.