WPROWADZENIE DO SIECI NEURONOWYCH

Sprawozdanie z laboratorium

Angelika Nadolska, Anna Wąsik, Kacper Wichowicz

1 Podstawowe informacje

1.1 Wprowadzenie

Głównym celem projektu jest zbudowanie modelu konwolucyjnej sieci neuronowej (CNN), klasyfikującego obrazy MRI na podstawie widoczności guza mózgu. Problem ten został już rozwiązany na platformie Kaggle. Dane są dostępne publicznie, wiele instytucji naukowych i badawczych udostępnia takie dane.
Do wytrenowania modelu użyliśmy architektury VGG-16. Miarą trafności modelu jest *accuracy*, którą definiuje się następująco:

$$accuracy = \frac{C}{T} \cdot 100\%,$$

gdzie C oznacza liczbę poprawnie przewidzianych obrazów, a T całkowitą liczbę testowanych obrazów.

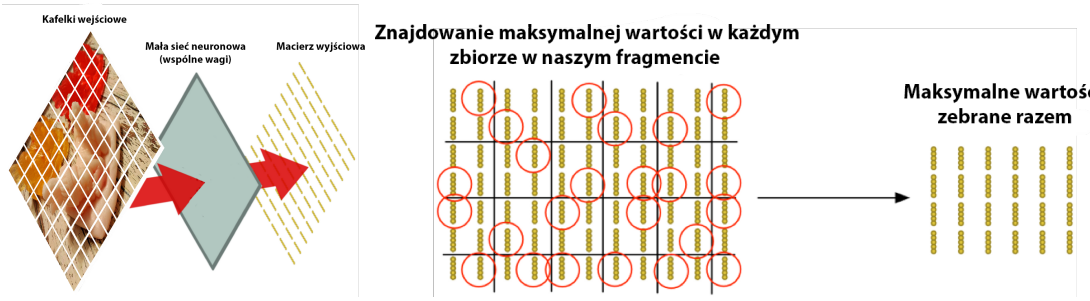
W związku z tym, że nasz zbiór danych był bardzo mały, skorzystaliśmy z tzw. transfer learningu. Także wygenerowaliśmy więcej danych robiąc różne transformacje na obrazkach.

1.2 Konwolucyjne sieci neuronowe (CNN)

Niestety, gdy posiadamy zbyt dużo danych wejściowych istnieje możliwość, że nasza głęboka sieć się "przeuczy". Chodzi tutaj, o to, żeby sieć faktycznie nauczyła się rozpoznawać wilki, a nie sytuacje, że występują one najczęściej na jasnym, zimnym tle (ponieważ w takiej sytuacji sieć rozpoznaje zimę, a nie wilki). Rozwiązaniem tego problemu wydają się być konwolucyjne sieci neuronowe.

Aby uniknąć przeuczania głębokich sieci neuronowych konwolucyjne sieci neuronowe badają wydzieloną część obrazu tak samo jak sieć głęboka, jednak skupiają się na konkretnej charakterystycznej cesze, odrzucając rzeczy nie interesujące.
Istotą sieci konwolucyjnych jest praca z danymi w postaci obrazów. Swoją nazwę zawdzięczają tym, że wybierając fragmenty obrazu korzystają z działania splotu, który definiujemy następująco:

$$s(t) = \int x(a)w(t-a)da$$

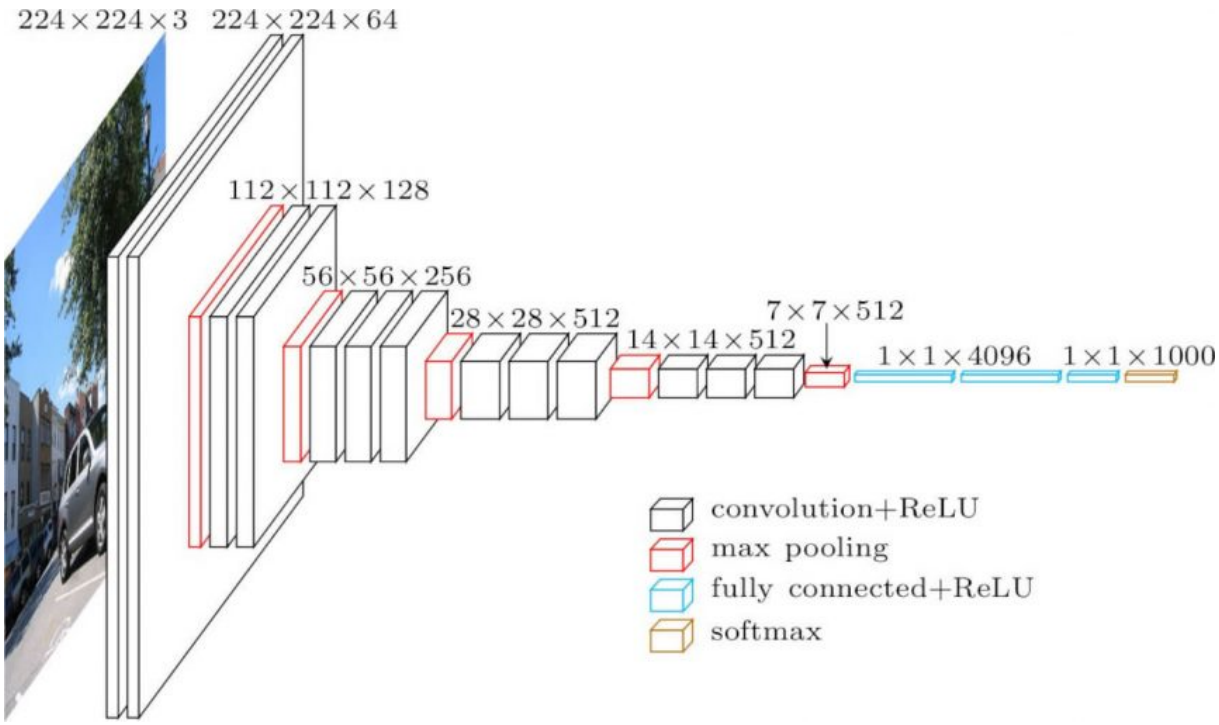


(a) (b)

Rysunek 1: Tworzenie macierzy i wybieranie interesujących wartości.

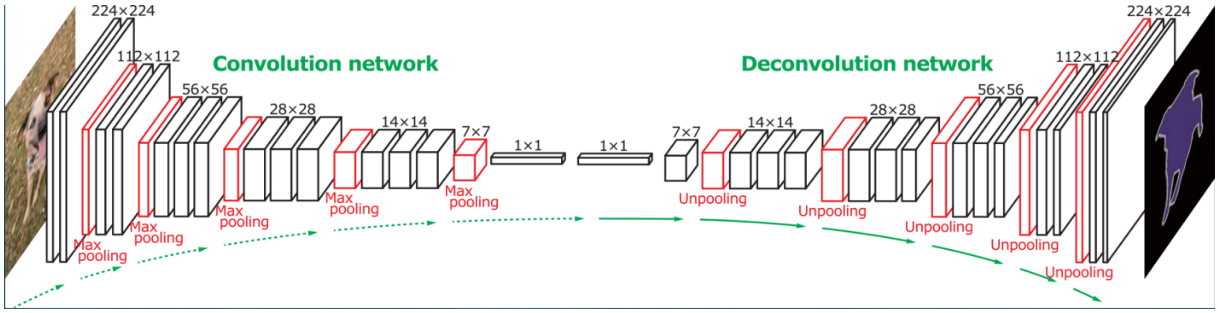
1.3 Architektura VGG-16

Używaliśmy architektury sieci VGG-16 głównie z tego powodu, że cieszy się dobrymi wynikami w zagadnieniach klasyfikacji obrazów. Jest nawet laureatem konkursu. Jej schematyczna architektura pokazana jest na poniższych obrazkach.



Rysunek 2: Schemat architektury VGG-16

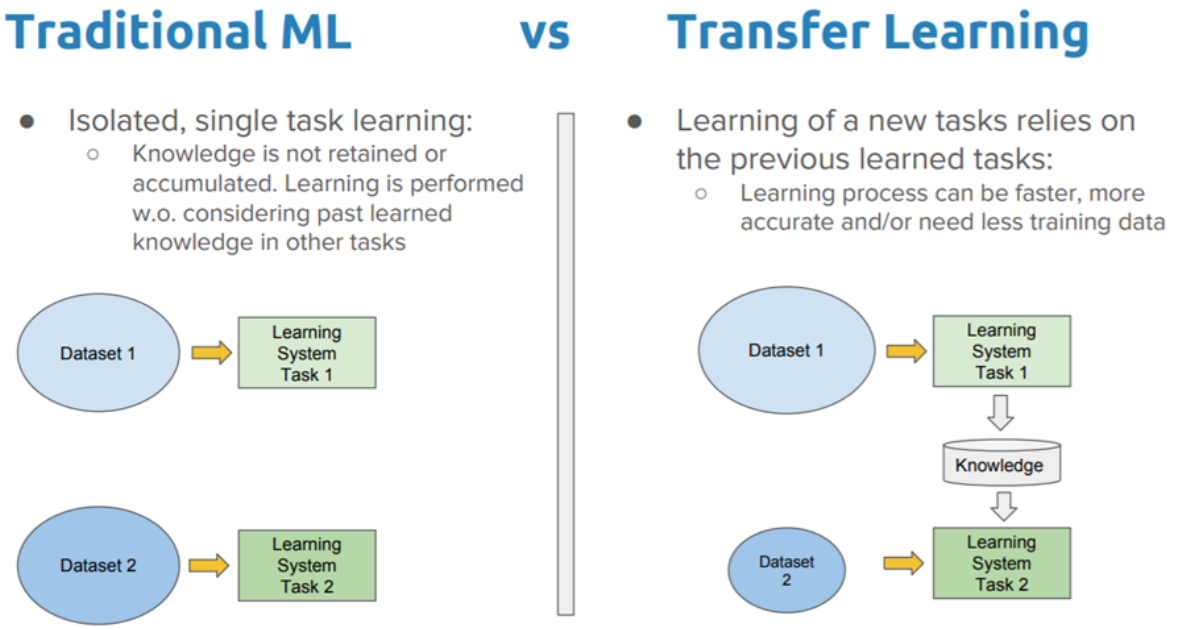
Dla przykładu można podać, że pierwszy etap konwolucji może rozpoznawać ostre krawędzie, w drugim etapie możemy dodać rozpoznawanie dziobów na podstawie wiedzy o ostrych krawędziach, trzecia warstwa konwolucyjna może już rozpoznać całego ptaka, opierając się na wiedzy o dziobie.



Rysunek 3: Przykładowy obraz przed konwolucją wyłapującą krawędzie, a następnie odtworzony

1.4 Transfer learning

Transfer learning polega na użyciu modelu, który został wytrenowany w jednej dziedzinie, do klasyfikacji w innej. Np. jeżeli umiemy jeździć na rolkach łatwiej będzie nam się nauczyć jeździć na łyżwach.

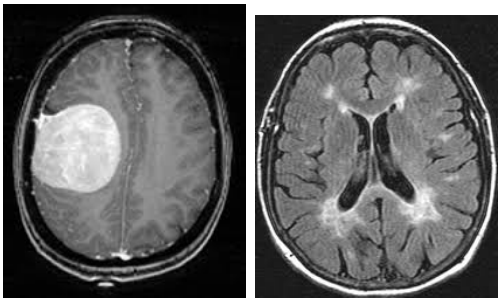


Rysunek 4: Porównanie machine learnignu z transfer Learningiem.

Naszym 'większym' zbiorem danych w transfer learningu jest baza danych Imagenet, w której znajdują się zdjęcia, które są zaprojektowane do użycia w oprogramowaniach do rozpoznawania obiektów na zdjęciach. Możemy wykorzystać wiele przykładowych zdjęć z tej bazy do uczenia naszej sieci neuronowej.

2 Przygotowywanie danych

Nasze dane są już sklasyfikowane na obrazy z nowotworem i bez.



(a) Obraz z nowotworem.

(b) Obraz bez nowotworu.

Rysunek 5: Przykładowe dane.

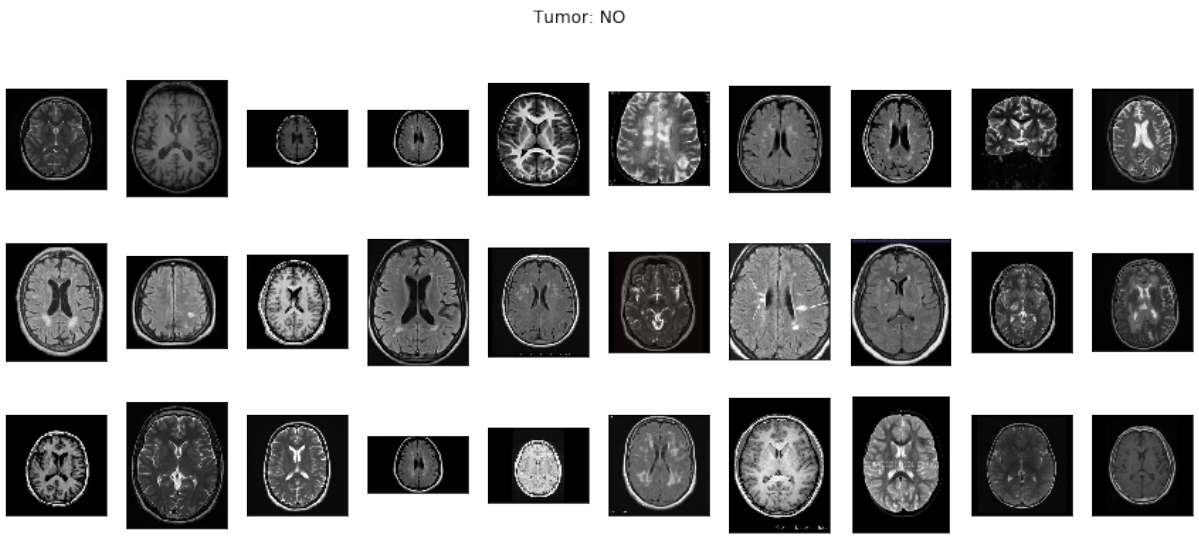
2.1 Podział na zbiory

Pozostaje nam podzielić je na zbiór testowy, treningowy i walidacyjny. Do tego, napisaliśmy program, który losowo wybiera po 5 obrazów do zbioru testowego, następnie 80% reszty losowo do zbioru treningowego, a to co zostało, przydziela do zbioru walidacyjnego.

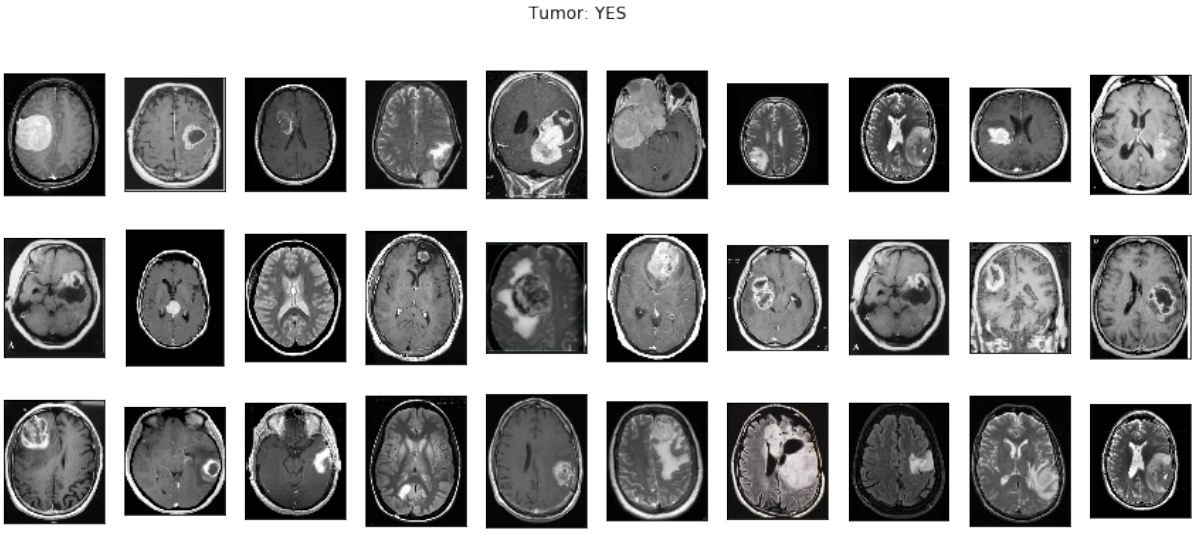
```
import glob
yes_dir = glob.glob('C:/git/obrazyMRI/data/brain_tumor_dataset/yes/*')
no_dir =glob.glob('C:/git/obrazyMRI/data/brain_tumor_dataset/no/*')

import os

# define the name of the directory to be created
path = ["C:/git/obrazyMRI/data/TEST/YES","C:/git/obrazyMRI/data/TEST/NO"
    , "C:/git/obrazyMRI/data/TRAIN/YES",
        "C:/git/obrazyMRI/data/TRAIN/NO", "C:/git/obrazyMRI/data/VAL/YES"
    , "C:/git/obrazyMRI/data/VAL/NO"]
for i in path:
    try:
        os.makedirs(i)
    except OSError:
        print ("Creation of the directory %s failed" % i)
    else:
        print ("Successfully created the directory %s" % i)

from math import *
import numpy as np
import shutil

yes_test=np.random.choice(yes_dir, size=5, replace=False)
```

```
23 for i in yes_test:
24     shutil.copy(i, "C:/git/obrazyMRI/data/TEST/YES" )
25 yes_dir=list(set(yes_dir)-set(yes_test))
26 yes_train=np.random.choice(yes_dir, size=floor(0.8*len(yes_dir)),
       replace=False)
27 for i in yes_train:
28     shutil.copy(i, "C:/git/obrazyMRI/data/TRAIN/YES" )
29 yes_val=list(set(yes_dir)-set(yes_train))
30 for i in yes_val:
31     shutil.copy(i, "C:/git/obrazyMRI/data/VAL/YES" )
32
33
34 no_test=np.random.choice(no_dir, size=5, replace=False)
35 for i in no_test:
36     shutil.copy(i, "C:/git/obrazyMRI/data/TEST/NO" )
37 no_dir=list(set(no_dir)-set(no_test))
38 no_train=np.random.choice(no_dir, size=floor(0.8*len(no_dir)), replace=
       False)
39 for i in no_train:
40     shutil.copy(i, "C:/git/obrazyMRI/data/TRAIN/NO" )
41 no_val=list(set(no_dir)-set(no_train))
42 for i in no_val:
43     shutil.copy(i, "C:/git/obrazyMRI/data/VAL/NO" )
```

W skrócie - w linijkach 2-3 tworzymy dwie listy napisów, a dokładniej ścieżek do każdego pliku. W zmiennej ‘path‘ znajdują się ścieżki do katalogów, które chcemy stworzyć. Pętla, zaczynająca się od 10-tej linijki tworzy nam foldery przy czym wypisuje, czy udało się je stworzyć, czy nie. W linijce 22-ej losujemy 5 obrazów, następnie pętlą wpisujemy je w folder. W linijce 25-tej odejmujemy obrazki już wylosowane. Reszta kodu jest analogiczna.



Rysunek 6: Bez nowotworu w zbiorze treningowym



Rysunek 7: Z nowotworem w zbiorze treningowym

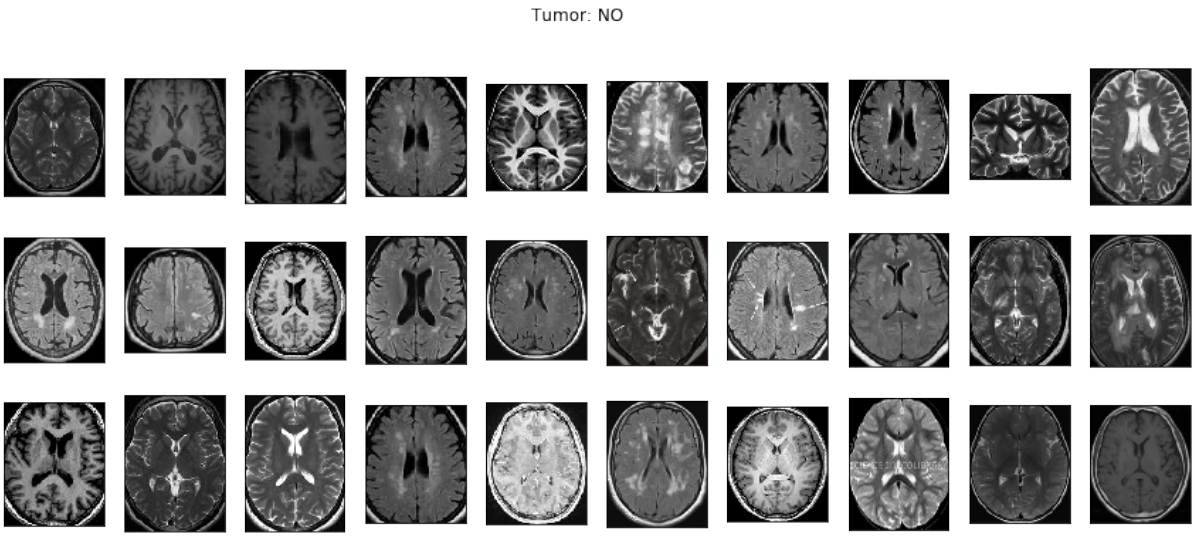
2.2 Normalizacja obrazów

Widzimy, że nasze zdjęcia mózgu są robione z różnej odległości. Spróbujemy je znormalizować tak, aby widoczny był mózg z jak najmniejszą ilością tła. W tym celu znajdziemy minimum i maksimum mózgu w pionie i poziomie.

Oto kod funkcji, która to zrobi:

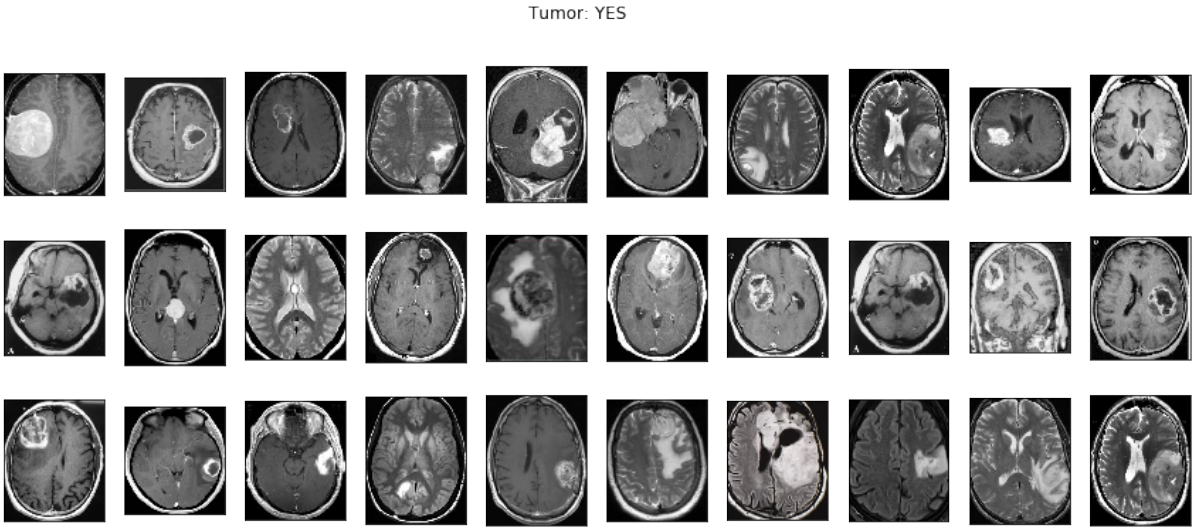
```
def crop_imgs(set_name, add_pixels_value=0):
    """
    Finds the extreme points on the image and crops the rectangular out
    of them
    """
    set_new = []
    for img in set_name:
        gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
        gray = cv2.GaussianBlur(gray, (5, 5), 0)

        # threshold the image, then perform a series of erosions +
        # dilations to remove any small regions of noise
        thresh = cv2.threshold(gray, 45, 255, cv2.THRESH_BINARY)[1]
        thresh = cv2.erode(thresh, None, iterations=2)
        thresh = cv2.dilate(thresh, None, iterations=2)

        # find contours in thresholded image, then grab the largest one
        cnts = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL, cv2.
    CHAIN_APPROX_SIMPLE)
        cnts = imutils.grab_contours(cnts)
        c = max(cnts, key=cv2.contourArea)

        # find the extreme points
```

```
        extLeft = tuple(c[c[:, :, 0].argmin()][0])
        extRight = tuple(c[c[:, :, 0].argmax()][0])
        extTop = tuple(c[c[:, :, 1].argmin()][0])
        extBot = tuple(c[c[:, :, 1].argmax()][0])

        ADD_PIXELS = add_pixels_value
        new_img = img[extTop[1]-ADD_PIXELS:extBot[1]+ADD_PIXELS, extLeft
    [0]-ADD_PIXELS:extRight[0]+ADD_PIXELS].copy()
        set_new.append(new_img)

    return np.array(set_new)
```

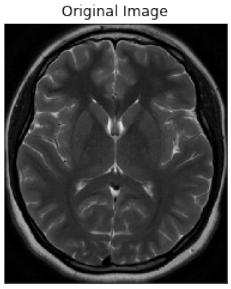


Rysunek 8: Bez nowotworu po normalizacji

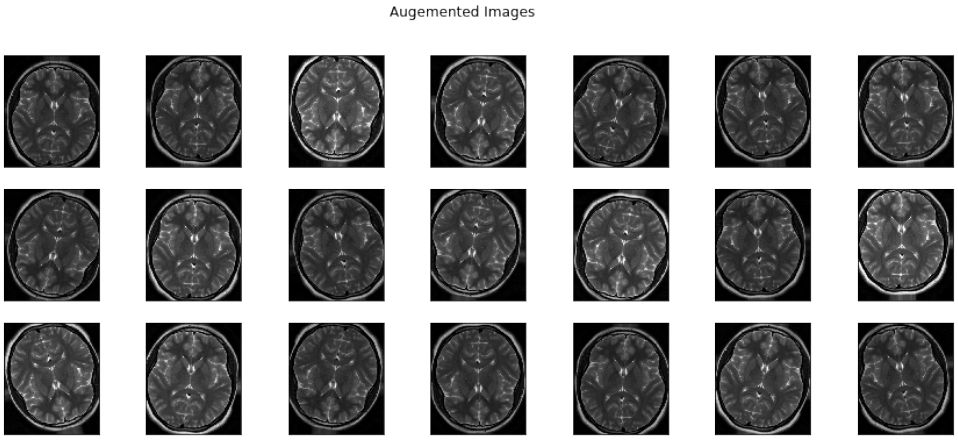
Obróbka danych jest już prawie skończona. Wystarczy jeszcze tylko ujednolicić rozmiar obrazów. Ujednoliciliśmy go do 224 na 224 pikseli.
W związku z tym, że nasza baza danych jest bardzo ograniczona, powiększymy ją przerabiając obrazy, tak aby wciąż było widoczne gdzie jest nowotwór, a gdzie go nie ma.



Rysunek 9: Z nowotworem po normalizacji



(a) Originalny obraz



Rysunek 10: Oraz 21 jego transformacji.

3 Tworzenie modelu

Tworzymy model bazowy architektury VGG-16, gdzie wagi są dostosowane z bazą danych imagenet.

```
base_model = VGG16(weights='imagenet', input_shape=IMG_SIZE + (3,),
    include_top=False)
```

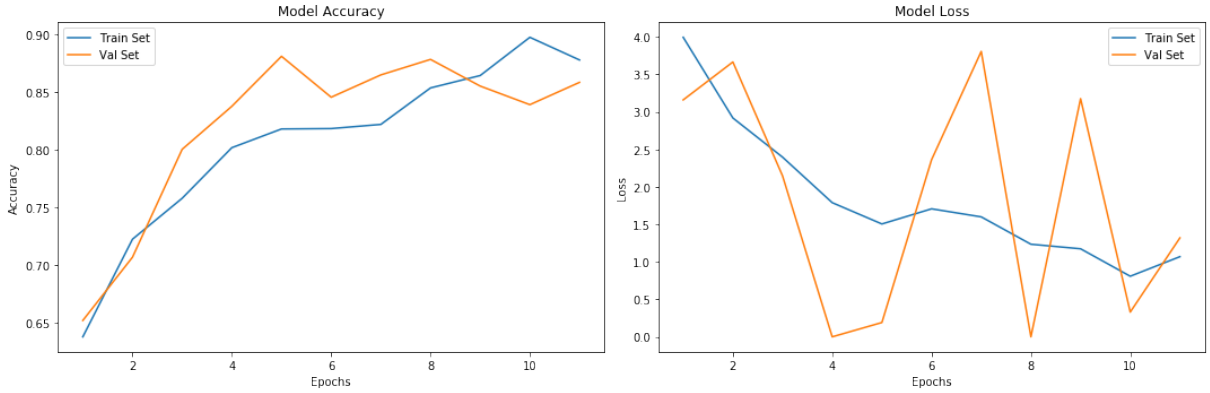
Następnie uzupełniamy model o warstwy flatten, dropout. Funkcją aktywacji jest oczywiście jakaś funkcja sigmoidalna, funkcją kosztu jest binary crossentropy, która jest używana przy problemach typu tak/nie (problemy binarne). Nasz problem oczywiście jest takim problemem, ponieważ musimy każde zdjęcie zaklasyfikować do pewnej kategorii („jest nowotwór" vs „nie ma nowotworu"). Funkcja binary crossentropy mierzy jak daleko od rzeczywistej wartości (która jest albo 0, albo 1), jest nasza predykcja dla każdej z klas, a następnie uśrednia błędy, aby obliczyć końcową stratę.

```
NUM_CLASSES = 1

model = Sequential()
model.add(base_model)
model.add(layers.Flatten())
model.add(layers.Dropout(0.5))
model.add(layers.Dense(NUM_CLASSES, activation='sigmoid'))

model.layers[0].trainable = False

model.compile(
    loss='binary_crossentropy',
    optimizer=RMSprop(lr=1e-4),
    metrics=['accuracy']
)

model.summary()
```

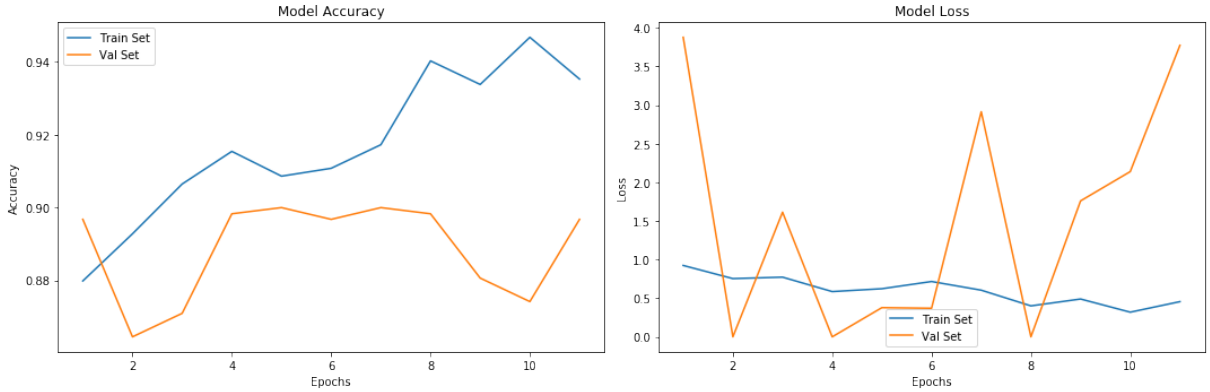
Po około 7 godzinach nasza sieć zakończyła proces uczenia się.

4 Wyniki

Trafność w zbiorze walidacyjnym początkowo wynosiła 82% oraz 90% dla zbioru testowego. Jest to całkiem dobry wynik, natomiast dziwnym wydaje się być fakt, że w zbiorze testowym nie sklasyfikowała jednego obrazu.
Łudząc się, że być może douczanie sieci pomoże w klasyfikacji tego obrazka, który nie został sklasyfikowany, postanowiliśmy podjąć to wyzwanie. Proces douczania okazał się być nic wznoszący do poprawności modelu (ewentualnie trafność na zbiorze walidacyjnym wzrosła do 90%), sieć wciąż nie klasyfikowała tego obrazka.



Rysunek 11: Wzrost trafności dla zbioru treningowego oraz walidacyjnego



Rysunek 12: Wartość trafności oraz funkcji kosztu dla zbioru treningowego oraz walidacyjnego po douczeniu sieci

5 Wnioski

Douczenie sieci za bardzo nic nie dało. Ewentualnie jako sukces można uznać wzrost trafności o 4% dla zbioru walidacyjnego, jednak jak widać z powyższych wykresów zbiór walidacyjny miał niemalże przez cały czas mniejszą trafność niż zbiór treningowy. Ogólnie możemy uznać, że model ma zadawalające wyniki, jednak dziwne jest dlaczego nie sklasyfikował jednego obrazka w zbiorze testowym.

W ćwiczeniu najbardziej pracochłonne, oczywiście, było przygotowanie danych. Okazało się też, że sama instalacja pakietów była lekkim wyzwaweniem.