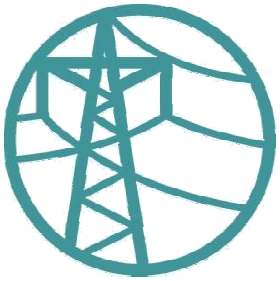


WYDZIAŁ
ELEKTROTECHNIKI
I INFORMATYKI
POLITECHNIKI RZESZOWSKIEJ

Marian Stopyra, 2EF-DI
gr. P3, nr indeksu 164014

Zastosowanie sieci konwolucyjnej do wykrywania obiektów na obrazie, realizacja w języku Python.

Praca projektowa z przedmiotu Sztuczna Inteligencja

Rzeszów, 2022

Spis treści

1. Cel projektu

Celem projektu jest realizacja konwolucyjnej sieci neuronowej w celu wykrywania obiektu (krzesła) na obrazie. Sieć realizuje zadanie na podstawie klasyfikacji binarnej obrazów. Sieć uczona na autorskim zbiorze danych, szacuje czy na zdjęciu widoczne jest krzesło - czy nie. ustalając przynależność obrazu do odpowiedniej klasy.

Sieć zrealizowana została w języku Python, z wykorzystaniem biblioteki Tensorflow. Uczona jest ona metodą wstecznej propagacji błędu, z wykorzystaniem metody momentum w celu przyspieszenia uczenia.

Dodatkowo zbadany został wpływ poszczególnych parametrów na proces uczenia sieci.

2. Opis danych

Zestaw danych został sporządzony ze zdjęć na wolnej licencji wybranych z rozpowszechniających je serwisów internetowych, i wykadrowanych w sposób skupiający się na klasie którą przedstawiają.
Zdjęcia zostały też obrócone lub odbite dla powiększenia liczebności zbioru.

Zestaw zawiera 100 zdjęć, podzielonych na 2 klasy.
Klasa „jest krzesło” – 50 obrazów
Klasa „nie ma krzesła” – 50 obrazów

2.1. Przygotowanie danych i normalizacja

Obrazy po wczytaniu do programu są zmniejszane do rozmiarów 64 na 64 pikseli, oraz sprowadzane do skali szarości, przy wykorzystaniu funkcji obróbki zestawów danych biblioteki Tensorflow.

Następnie dane są normalizowane przez sprowadzenie wartości poszczególnych pikseli z przedziału <0, 255> do przedziału <0, 1>.

Każdy obraz sparowany jest z identyfikatorem o wartości ze zbioru {0, 1} określającym przynależność do odpowiedniej klasy. Identyfikator 0 oznacza przynależność obrazu do klasy „nie ma krzesła”, a identyfikator 1 – do klasy „jest krzesło”.

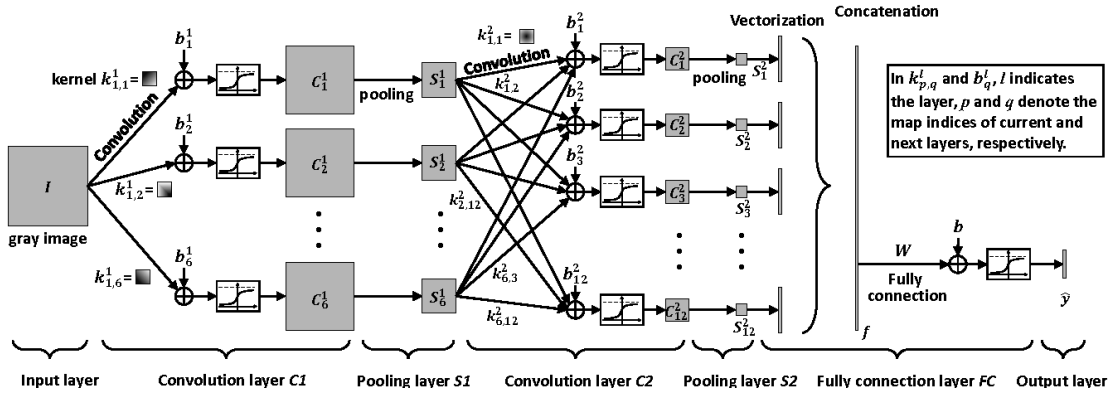
Zestaw danych jest losowo mieszany, z zachowaniem integralności między obrazami a przypisanymi do nich identyfikatorami klas.

Dodatkowo zbiór danych dzielony jest na zestaw danych uczących, i zestaw danych służących do walidacji, z proporcjami 80/20. Rys. 1.1. przedstawia przykładowe 25 elementów wybranych z zestawu danych uczących.

Rys. 1.1. Próbka 25 obrazów ze zbioru uczącego wraz z klasami do których przynależą

3. Opis teoretyczny

Tematyką projektu jest zastosowanie konwolucyjnej sieci neuronowej do rozpoznawania obrazów. Zastosowanie sieci w pełni połączonych w tym celu wymagałoby ogromnej ilości zasobów, gdyż połączenie każdego piksela z każdym neuronem skutkowałoby ogromną liczbą połączeń, a więc uczonych parametrów. Sieć konwolucyjna pozwala zmniejszyć ilość wymaganych zasobów i zwiększyć jej wydajność, poprzez ograniczenie liczby parametrów które trzeba nauczyć, jednocześnie zachowując sąsiedztwo pikseli. Rys. 3.1. przedstawia przykładowy schemat sieci konwolucyjnej.



Rys. 3.1. Typowy schemat sieci konwolucyjnej

Sieć konwolucyjna zbudowana jest z warstw konwolucji ułożonych naprzemiennie z warstwami poolingu, warstwy spłaszczającej, i warstw w pełni połączonych. Warstwa spłaszczająca umożliwia przejście z warstw konwolucji i poolingu na warstwy w pełni połączone.

Dla warstw konwolucji na jeden obraz nakładane jest jeden lub więcej filtrów, zwanych również kernelami. Filtr mnoży piksele obrazu na wejściu przez swoje wagi, sumuje je dodając bias, i przepuszcza przez funkcję aktywacji, tworząc piksel nowej konwolucji. Proces powtarzany jest dla każdej dozwolonej pozycji filtra na obrazie na wejściu, tworząc tym samym nową konwolucję, mniejszą od obrazu źródłowego.

W projekcie wykorzystane zostały filtry o rozmiarach 3px na 3px, więc powstała w ten sposób konwolucja jest o 2px w każdym wymiarze mniejsza od obrazu z którego powstała.

Konwolucję pierwszej warstwy można opisać wzorem (3.1)

$$C_p^1(m,n) = f\left(\sum_i \sum_j k_{i,j} I_{m-i,n-j}\right) \tag{3.1}$$

Gdzie jako funkcja aktywacji wykorzystana została funkcja ReLU (Rectified Linear Unit) określona wzorem (3.2)

$$f(\mathrm{x}) = \begin{cases} 0 & gdy\ x < 0 \\ x & gdy\ x \geq 0 \end{cases} \tag{3.2}$$

Po każdej warstwie konwolucyjnej użyta jest warstwa poolingu, która redukuje wymiary obrazu poprzez podzielenie obrazu na części 2px na 2px, i przypisując koresponduącemu pikselowi obrazu wyjściowego wartość obliczoną z wybranej funkcji. W warstwach poolingu wykorzystana została funkcja maksimum, wybierająca najwyższą wartość spośród czterech pikseli wydzielonej części.

Warstwa spłaszczająca dokonuje na każdym obrazie na jej wejściach operacji wektoryzacji, przetwarzając go na wektor jednowymiarowy. Następnie dokonuje na uzyskanych wektorach jednowymiarowych operacji łączenia, łącząc je w jeden długi wektor. Wektor ten przekazywany jest na wejście warstwy w pełni połączonej.

4. Algorytm podstawowy

Program napisany został w języku Python, korzystając z bibliotek Tensorflow i Keras. Program obsługuje wczytanie i przygotowanie zestawów danych, budowę i uczenie modelu sieci konwolucyjnej, oraz trzy eksperymenty.

Model złożony jest z trzech warstw konwolucyjnych, trzech następujących po każdej z nich warstw poolingu, warstwy spłaszczającej, a następnie trzech warstw w pełni połączonych, z których ostatnia złożona jest z pojedynczego neuronu, i jest warstwą wyjściową.

Wykorzystane warstwy konwolucyjne używają filtrów o rozmiarach 3 na 3 piksele, a wpływ liczby filtrów przypadających na warstwę zbadany został podczas eksperymentu trzeciego.

Zarówno filtry warstw konwolucyjnych jak i neurony warstw w pełni połączonych, wykorzystują funkcję aktywacji ReLU.

Efektywność nauczonej sieci oceniana jest według klasyfikatora trafności (ACC). Przeprowadzone eksperymenty badają wpływ na trafność nauczonej sieci kolejno współczynnika uczenia, stałej momentum, i liczby filtrów w warstwach konwolucyjnych.

Pełen kod programu zamieszczony jest w listingu 4.1.

```
print("uruchomiono program")
#1 - importowanie bibliotek
import tensorflow

from tensorflow.keras import datasets, layers, models, utils,
preprocessing
import matplotlib.pyplot as plt
import tensorflow.keras.preprocessing.image as kerasImg
import numpy
print("zaimportowano biblioteki")
```

```
#2 - przygotowanie danych

rozmiarObrazow = (64, 64)

zbiorObrazow = []
zbiorLabels = []
zbiorValidationObrazow = []
zbiorValidationLabels = []
klasy = ["nie ma krzesla", "jest krzeslo"]
lr=0.0003 #współczynnik uczenia
mc=0.9 #momentum constant
liczbaEpok = 200

sciezka = "obrazy\\jest krzeslo\\"
for i in range(40):
    sciezkaObrazu = sciezka + format(i+1, "02d") + ".jpg"
    #format:
https://docs.python.org/2/library/string.html#format-
specification-mini-language
    #02d - leading zeros, length 2, decimal
    obraz = kerasImg.load_img(sciezkaObrazu,
        color_mode="rgb", #kolor obrazu zrodlowego
        target_size=rozmiarObrazow
    )
    dane = tensorflow.image.rgb_to_grayscale(obraz)
    #rgb_to_grayscale zastepuje dodatkowo
kerasImg.img_to_array()
    zbiorObrazow.append(dane) #tf.Tensor danych
    zbiorLabels.append([1]) # [1] bo labelsy w Keras sa
arrayami 1-elementowymi

sciezka = "obrazy\\nie ma krzesla\\"
for i in range(40):
    sciezkaObrazu = sciezka + format(i+1, "02d") + ".jpg"
    obraz = kerasImg.load_img(sciezkaObrazu,
        color_mode="rgb",
        target_size=rozmiarObrazow
    )
    dane = tensorflow.image.rgb_to_grayscale(obraz)
    zbiorObrazow.append(dane)
    zbiorLabels.append([0])
```

```
sciezka = "obrazy\\validation jest krzeslo\\"
for i in range(10):
    sciezkaObrazu = sciezka + format(i+1, "02d") + ".jpg"
    obraz = kerasImg.load_img(sciezkaObrazu,
        color_mode="rgb",
        target_size=rozmiarObrazow
    )
    dane = tensorflow.image.rgb_to_grayscale(obraz)
    zbiorValidationObrazow.append(dane)
    zbiorValidationLabels.append([1])

sciezka = "obrazy\\validation nie ma krzesla\\"
for i in range(10):
    sciezkaObrazu = sciezka + format(i+1, "02d") + ".jpg"
    obraz = kerasImg.load_img(sciezkaObrazu,
        color_mode="rgb",
        target_size=rozmiarObrazow
    )
    dane = tensorflow.image.rgb_to_grayscale(obraz)
    zbiorValidationObrazow.append(dane)
    zbiorValidationLabels.append([0])

print("wczytano dane")

#3.2 pomieszanie danych uczacych
zbiorShuffle = []
for i in range(len(zbiorObrazow)):
     zbiorShuffle += [[zbiorObrazow[i], zbiorLabels[i]]]

numpy.random.shuffle(zbiorShuffle)

for i in range(len(zbiorShuffle)):
     zbiorObrazow[i] = zbiorShuffle[i][0]
     zbiorLabels[i] = zbiorShuffle[i][1]


#3.3 - weryfikacja danych
plt.figure(figsize=(8,8))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(zbiorObrazow[i], cmap=plt.cm.gray) #wyswietla
obraz w skali szarosci
    plt.xlabel(klasy[zbiorLabels[i][0]]) #labelsy sa arrayami
1-elem
plt.show()
```

```
#3.4 normalizacja danych
for obr in range(len(zbiorObrazow)):
    zbiorObrazow[obr] = zbiorObrazow[obr].numpy().tolist()
    #konwersja tensor -> numpy array -> array
    #normalizacja:
    for i in range(len(zbiorObrazow[obr])):
        for j in range(len(zbiorObrazow[obr][i])):
            zbiorObrazow[obr][i][j][0] =
zbiorObrazow[obr][i][j][0] / 255

for obr in range(len(zbiorValidationObrazow)):
    zbiorValidationObrazow[obr] =
zbiorValidationObrazow[obr].numpy().tolist()
    for i in range(len(zbiorValidationObrazow[obr])):
        for j in range(len(zbiorValidationObrazow[obr][i])):
            zbiorValidationObrazow[obr][i][j][0] =
zbiorValidationObrazow[obr][i][j][0] / 255


#4 - budowa modelu
print("#4 - budowa modelu")
model = models.Sequential()
model.add(layers.Conv2D(14, (3, 3), activation='relu',
input_shape=(64, 64, 1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(11, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(11, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())
model.add(layers.Dense(24, activation='relu'))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(1))

#model.summary()

#5 - kompilacja i uczenie modelu
print("#5 - kompilacja i uczenie modelu")
optimizerMomentum = tensorflow.keras.optimizers.SGD(
    learning_rate=lr, momentum=mc)
model.compile(optimizer=optimizerMomentum,

loss=tensorflow.keras.losses.BinaryCrossentropy(),

metrics=[tensorflow.keras.metrics.BinaryAccuracy()])
```

```
history = model.fit(zbiorObrazow,
                    zbiorLabels,
                    epochs=liczbaEpok,
                    validation_data=(zbiorValidationObrazow,
                                     zbiorValidationLabels))

#6 - wykresy
acc = history.history['binary_accuracy']
val_acc = history.history['val_binary_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs_range = range(liczbaEpok)

#   6.2 wygładzenie
smooth_size = 10
acc_smooth = []
val_acc_smooth = []

for i in range(smooth_size):
    acc_smooth += [acc[i]]
    val_acc_smooth += [val_acc[i]]

for i in range(smooth_size, len(acc)):
    smoothen = 0
    for s in range(smooth_size):
        smoothen += acc[i-s]
    acc_smooth += [smoothen/smooth_size]
    smoothen = 0
    for s in range(smooth_size):
        smoothen += val_acc[i-s]
    val_acc_smooth += [smoothen/smooth_size]

#   6.3 rysowanie wykresów trafności i błędu
plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc_smooth, label='Trafność na zbiorze
uczącym')
plt.plot(epochs_range, val_acc_smooth, label='Trafność na
zbiorze walidacyjnym')
plt.legend(loc='lower right')
plt.xlabel("Epoka")
plt.ylabel("Trafność")
plt.title('Wykres trafności')
```

```
plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Błąd na zbiorze uczącym')
plt.plot(epochs_range, val_loss, label='Błąd na zbiorze
walidacyjnym')
plt.legend(loc='upper right')
plt.xlabel("Epoka")
plt.ylabel("Błąd")
plt.title('Wykres funkcji błędu')
plt.show()

print("końcowe:")
print("[niewygł]Trafność na zbiorze uczącym", acc[len(acc)-1])
print("[wygł]Trafność na zbiorze uczącym",
acc_smooth[len(acc_smooth)-1])
print("[niewygł]Trafność na zbiorze walidacyjnym",
val_acc[len(val_acc)-1])
print("[wygł]Trafność na zbiorze walidacyjnym",
val_acc_smooth[len(val_acc_smooth)-1])
print("Loss na zbiorze uczącym", loss[len(loss)-1])
print("Loss na zbiorze walidacyjnym", val_loss[len(val_loss)-
1])


#7. eksperymenty
#   7.1 eksperyment learning rate
print("7.1 eksperyment learning rate")
liczbaEpok = 100
lr_range = []
for i in range(9):
    lr_range += [(i+1)/100000]
for i in range(9):
    lr_range += [(i+1)/10000]
for i in range(9):
    lr_range += [(i+1)/1000]
for i in range(9):
    lr_range += [(i+1)/100]
for i in range(10):
    lr_range += [(i+1)/10]

eksperymentlr_acc = []
eksperymentlr_val_acc = []
for lr in lr_range:
    optimizerMomentum = tensorflow.keras.optimizers.SGD(
    learning_rate=lr, momentum=mc)
    model.compile(optimizer=optimizerMomentum,

loss=tensorflow.keras.losses.BinaryCrossentropy(),

metrics=[tensorflow.keras.metrics.BinaryAccuracy()])
```

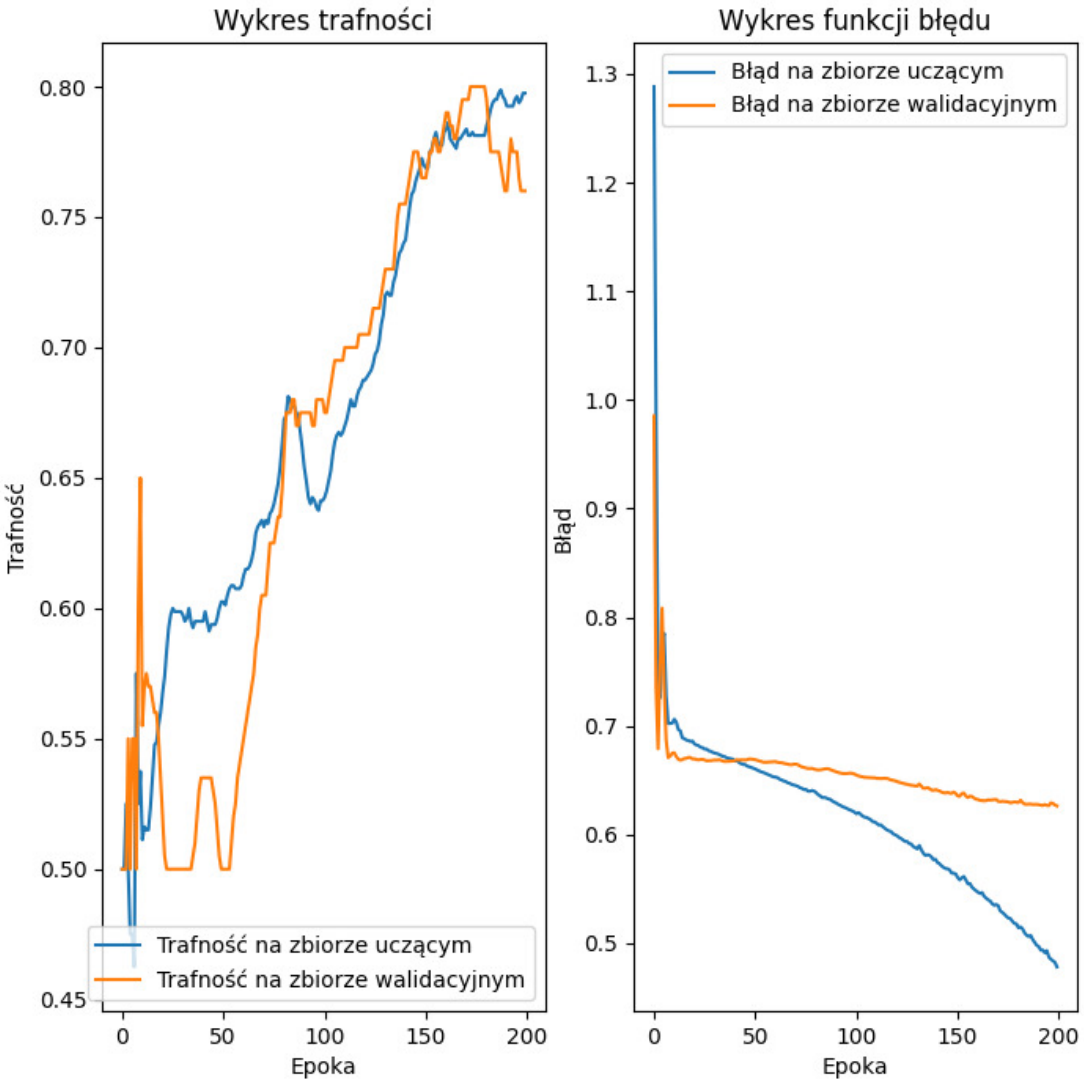
```
history = model.fit(zbiorObrazow,
                zbiorLabels,
                epochs=liczbaEpok,
                validation_data=(zbiorValidationObrazow,
                zbiorValidationLabels),
                verbose=0) #nie wypisuje wynikow co epokę

    acc = history.history['binary_accuracy']
    val_acc = history.history['val_binary_accuracy']

    eksperymentlr_acc += [acc[len(acc)-1]]
    eksperymentlr_val_acc += [val_acc[len(val_acc)-1]]
    print("dla lr =", lr, "osiagnieta trafność =",
[acc[len(acc)-1]])


plt.figure(figsize=(8, 8))
plt.xscale("log")
plt.plot(lr_range, eksperymentlr_acc, label='Trafność na
zbiorze uczącym')
plt.plot(lr_range, eksperymentlr_val_acc, label='Trafność na
zbiorze walidacyjnym')
plt.legend(loc='lower right')
plt.xlabel("Współczynnik uczenia")
plt.ylabel("Trafność")
plt.title('Zależność trafności od współczynnika uczenia')
plt.show()


#   7.2 eksperyment momentum constant
print("7.2 eksperyment momentum constant")
lr=0.0003
liczbaEpok = 100
mc_range = []
for i in range(20):
    mc_range += [(i+1)/20]


eksperymentmc_acc = []
eksperymentmc_val_acc = []
for mc in mc_range:
    optimizerMomentum = tensorflow.keras.optimizers.SGD(
    learning_rate=lr, momentum=mc)
    model.compile(optimizer=optimizerMomentum,

loss=tensorflow.keras.losses.BinaryCrossentropy(),

metrics=[tensorflow.keras.metrics.BinaryAccuracy()])
```

```
    history = model.fit(zbiorObrazow,
                        zbiorLabels,
                        epochs=liczbaEpok,

validation_data=(zbiorValidationObrazow,

zbiorValidationLabels),
                        verbose=0) #nie wypisuje wynikow co
epokę

    acc = history.history['binary_accuracy']
    val_acc = history.history['val_binary_accuracy']

    eksperymentmc_acc += [acc[len(acc)-1]]
    eksperymentmc_val_acc += [val_acc[len(val_acc)-1]]
    print("dla mc =", mc, "osiagnieta trafność =",
[acc[len(acc)-1]])


plt.figure(figsize=(8, 8))
plt.plot(mc_range, eksperymentmc_acc, label='Trafność na
zbiorze uczącym')
plt.plot(mc_range, eksperymentmc_val_acc, label='Trafność na
zbiorze walidacyjnym')
plt.legend(loc='lower right')
plt.xlabel("Stała momentum")
plt.ylabel("Trafność")
plt.title('Zależność trafności od stałej momentum')
plt.show()



#   7.3 eksperyment liczby filtrów w warstwach 1, 2 i 3
print("7.3 eksperyment liczby filtrów w warstwach")
first_layer_filters_range = [4, 6, 8, 10, 12, 14, 16, 18, 20]
second_and_3rd_layers_filters_range = [4, 6, 8, 10, 12, 14,
16, 18, 20]
liczbaEpok = 100
lr = 0.0003
mc = 0.9

experiment3_results = []
experiment3_val_results = []
```

```
for S1 in first_layer_filters_range:
    res = []
    val_res = []
    #budowa modelu z nową liczbą filtrów
    for S2S3 in second_and_3rd_layers_filters_range:
        model = models.Sequential()
        model.add(layers.Conv2D(S1, (3, 3), activation='relu',
                                input_shape=(64, 64, 1)))
        model.add(layers.MaxPooling2D((2, 2)))
        model.add(layers.Conv2D(S2S3, (3, 3),
activation='relu'))
        model.add(layers.MaxPooling2D((2, 2)))
        model.add(layers.Conv2D(S2S3, (3, 3),
activation='relu'))
        model.add(layers.MaxPooling2D((2, 2)))
        model.add(layers.Flatten())
        model.add(layers.Dense(24, activation='relu'))
        model.add(layers.Dense(16, activation='relu'))
        model.add(layers.Dense(1))

        #kompilacja i uczenie
        optimizerMomentum = tensorflow.keras.optimizers.SGD(
            learning_rate=lr, momentum=mc)
        model.compile(optimizer=optimizerMomentum,

loss=tensorflow.keras.losses.BinaryCrossentropy(),

metrics=[tensorflow.keras.metrics.BinaryAccuracy()])

        history = model.fit(zbiorObrazow,
                            zbiorLabels,
                            epochs=liczbaEpok,

validation_data=(zbiorValidationObrazow,

zbiorValidationLabels),
                            verbose=0)

        #zapis wyników
        acc = history.history['binary_accuracy']
        val_acc = history.history['val_binary_accuracy']

        res += [acc[len(acc)-1]]
        val_res += [val_acc[len(val_acc)-1]]
        print("dla S1 =", S1, "i S2S3 = ", S2S3,
              "osiagnieta trafność =", [acc[len(acc)-1]])
    experiment3_results += [res]
    experiment3_val_results += [val_res]
```

```
#przygotowanie danych do wykresów
experiment3_results = numpy.array(experiment3_results)
experiment3_val_results = numpy.array(experiment3_val_results)
xaxis, yaxis = numpy.meshgrid(first_layer_filters_range,
second_and_3rd_layers_filters_range)


#wykres dla zbioru uczącego
plt.figure(figsize=(8, 8))
trojwym = plt.axes(projection='3d')
trojwym.set_xlabel("Liczba filtrów pierwszej warstwy")
trojwym.set_ylabel("Liczba filtrów drugiej i trzeciej
warstwy")
trojwym.set_zlabel("Trafność")
trojwym.plot_surface(xaxis, yaxis,
                     experiment3_results, cmap='jet')
plt.title('Zależność trafności od liczby filtrów na warstwę
dla zbioru uczącego')
plt.show()

#wykres dla zbioru walidacyjnego
plt.figure(figsize=(8, 8))
trojwym2 = plt.axes(projection='3d')
trojwym2.set_xlabel("Liczba filtrów pierwszej warstwy")
trojwym2.set_ylabel("Liczba filtrów drugiej i trzeciej
warstwy")
trojwym2.set_zlabel("Trafność")
trojwym2.plot_surface(xaxis, yaxis,
                     experiment3_val_results, cmap='plasma')
plt.title('Zależność trafności od liczby filtrów na warstwę
dla zbioru walidacyjnego')
plt.show()
```

Listing 4.1. Pełen kod programu



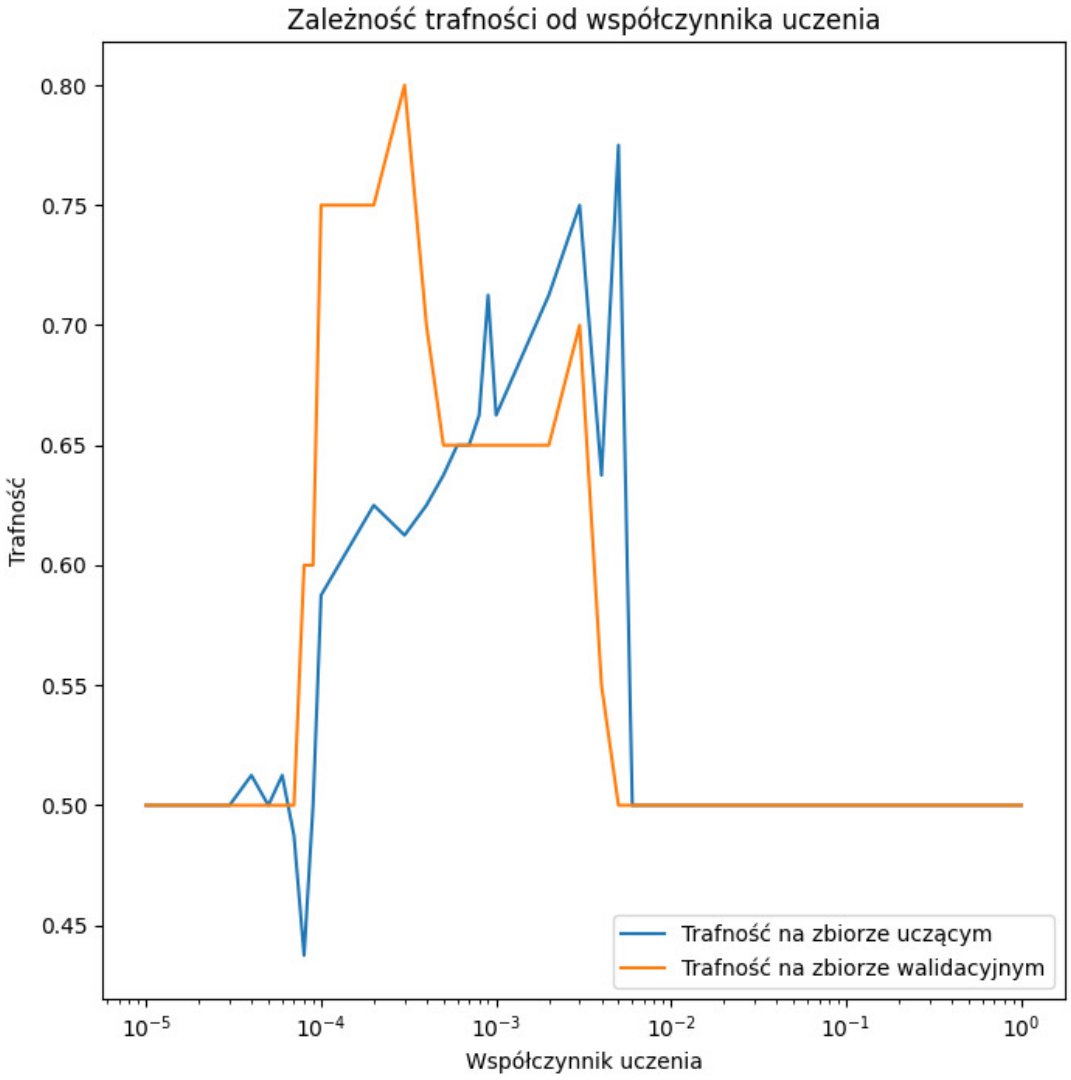
Rys. 4.1. Wyniki pojedynczego uczenia

Na Rys. 4.1. widoczne są wykresy zmian trafności i błędu względem liczby epok uczenia. Wraz z kolejnymi epokami trafność zarówno na zbiorze uczącym, jak i na zbiorze walidacyjnym, wykazują trend wzrostowy, a wartości błędu – spadkowy. Efekty uczenia są bardziej uwydatnione na wynikach osiąganych na zbiorze uczącym, niż na tych osiąganych na zbiorze walidacyjnym. Jest to prawidłowe i oczekiwane zachowanie uczonej sieci.

5. Eksperymenty

W celu zbadania wpływu poszczególnych parametrów na proces uczenia sieci, la znalezienia ich optymalnej konfiguracji, przeprowadzone zostały eksperymenty opisane w sekcjach 5.1, 5.2, i 5.3.

5.1. Eksperyment 1 – badanie wpływu współczynnika uczenia na trafność



Rys. 5.1. Wyniki eksperymentu 1

W ramach eksperymentu pierwszego zbadany został wpływ współczynnika uczenia na trafność nauczonej sieci. Eksperyment został przeprowadzony dla współczynników uczenia z zakresu $<10^{-5}, 1>$, stałej momentum równej 0.0003, i czasu uczenia 100 epok.

Na Rys. 5.1. widoczny jest wykres zależności trafności nauczonej sieci od współczynnika uczenia *lr*. Dla współczynnika uczenia mniejszego od 10^{-4} lub większego od 10^{-2} sieć ma wyraźne problemy z nauczeniem się. Dla współczynnika uczenia w zakresie pomiędzy tymi liczbami, sieć uczy się prawidłowo.

Najlepsze wyniki dla zbioru uczącego skupione są wokół wartości $lr \approx 5*10^{-3}$.
Dla zbioru walidacyjnego najwyższą trafność sieć uzyskuje dla wartości *lr* pomiędzy 10^{-4} a 10^{-3}. Osiągana dla nich trafność przewyższa trafność uzyskaną dla zbioru uczącego, co może być uznane za anomalię. Wytłumaczeniem zajścia tego zjawiska może być relatywna prostota klasyfikacji obrazów znajdujących się w zbiorze danych walidacyjnych, względem tych w zbiorze danych uczących.

5.2. Eksperyment 2 – badanie wpływu stałej momentum na trafność

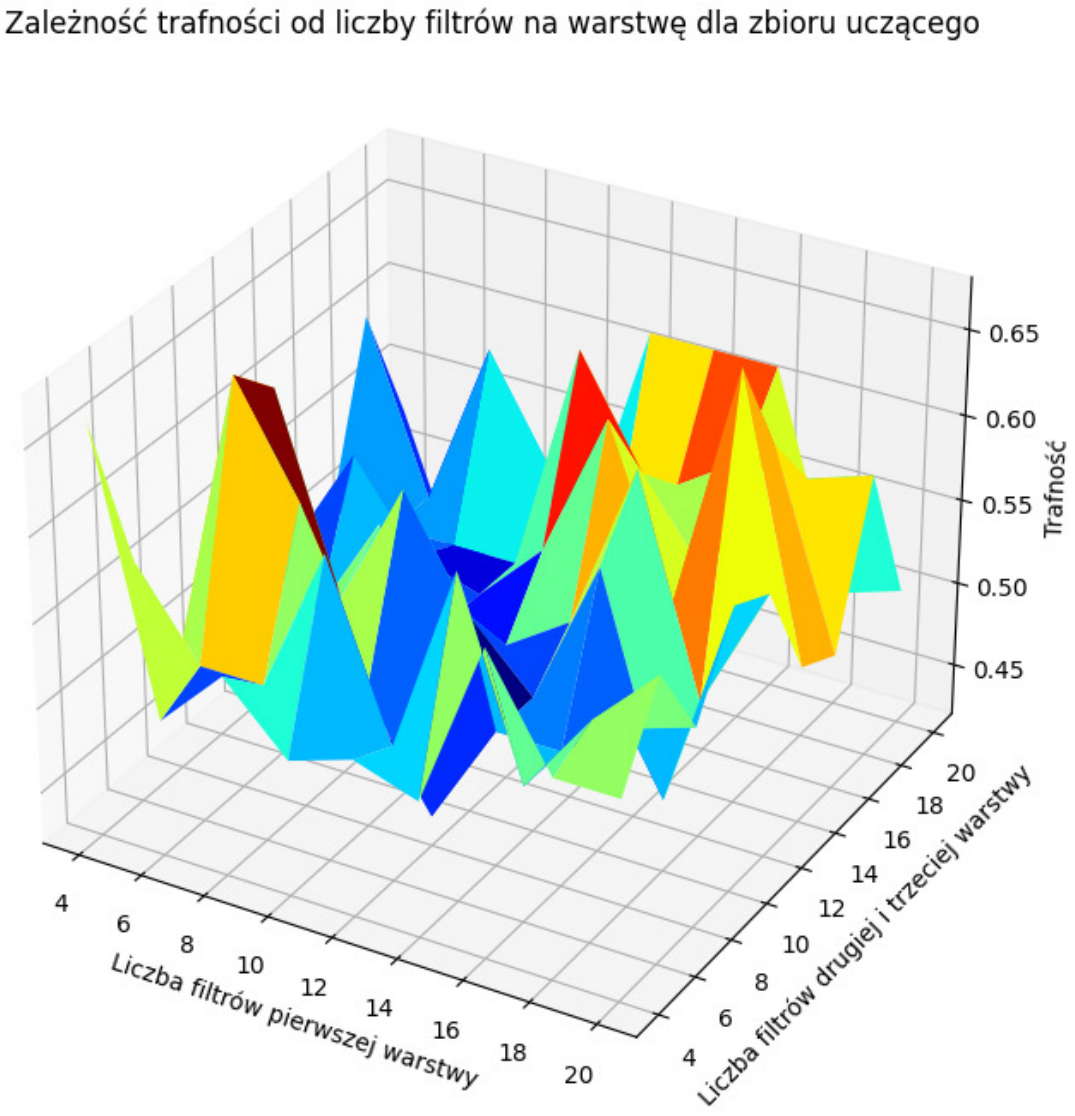
Rys. 5.2. Wyniki eksperymentu 2

W ramach eksperymentu drugiego zbadany został wpływ stałej momentum na trafność nauczonej sieci. Eksperyment został przeprowadzony dla współczynnika uczenia 0.0003, stałych momentum z przedziału <0.05, 1>, i czasu uczenia 100 epok.

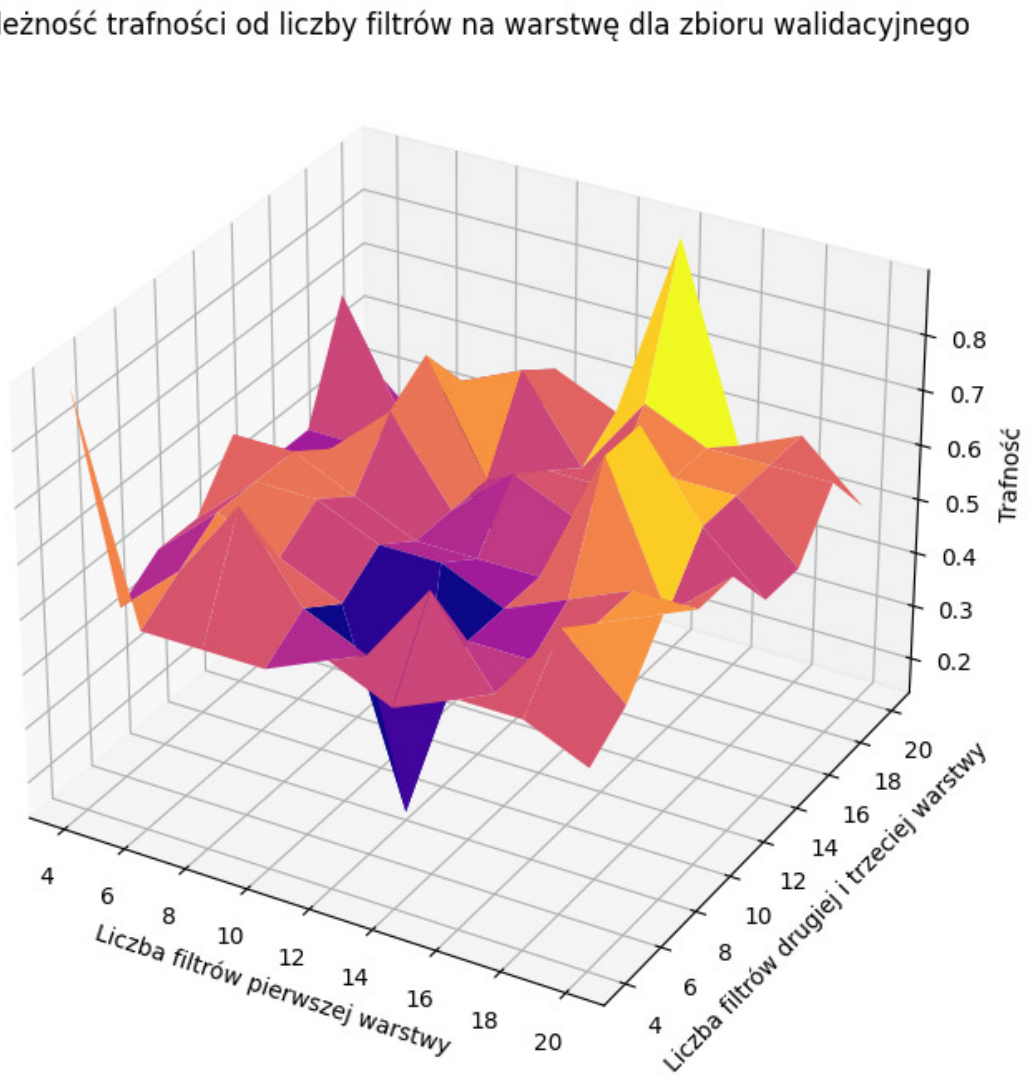
Na Rys. 5.2. widoczny jest wykres zależności trafności nauczonej sieci od stałej momentum *mc*. Sieć uczy się dla wszystkich wartości *mc*, ale tylko dla *mc*∈<0.8, 1.0) uczy się efektywnie. Dla *mc*=1.0 sieć osiąga najniższą trafność. Najwyższą trafność zaś sieć osiąga dla *mc*=0.95. Podobnie jak w eksperymencie pierwszym, również widoczna jest anomalia dla trafności na zbiorze walidacyjnym, ale jest ona mniej uwydatniona niż w eksperymencie pierwszym.

5.3. Eksperyment 3 – badanie wpływu liczby filtrów w warstwach konwolucyjnych na trafność

W ramach eksperymentu trzeciego zbadany został wpływ liczby filtrów w poszczególnych warstwach konwolucyjnych na trafność nauczonej sieci. Eksperyment został przeprowadzony dla współczynnika uczenia 0.0003, stałej momentum 0.9, i czasu uczenia 100 epok. Badanie przeprowadzone zostały z wykorzystaniem dwóch zmiennych. Pierwsza zmienna odpowiadała za liczbę filtrów w warstwie pierwszej (wejściowej). Druga zaś – za liczbę filtrów warstw drugiej i trzeciej jednocześnie. Obydwie zmienne przybierały wartości ze zbioru {4, 6, 8, …, 16, 18, 20}, i zbadane zostały wszystkie ich kombinacje.



Rys. 5.3. Wyniki eksperymentu 3 dla zbioru danych uczących



Rys. 5.4. Wyniki eksperymentu 3 dla zbioru danych walidacyjnych

Na Rys. 5.3. i 5.4. widoczny jest wykres zależności trafności nauczonej sieci od liczb filtrów w poszczególnych warstwach konwolucyjnych. Rys. 5.3. przedstawia tę zależność dla zbioru danych uczących, a Rys. 5.4. – dla danych walidacyjnych.

W obydwu przypadkach, wpływ liczby filtrów w poszczególnych warstwach zdaje się mieć dużo mniejsze znaczenie dla trafności nauczonej sieci niż pozostałe zbadane parametry.

Najwyższa trafność dla obu zbiorów danych osiągnięta została dla 14 filtrów warstwy pierwszej, i po 20 filtrów warstw drugiej i trzeciej. Wysoka trafność została również osiągnięta na drugim krańcu badanego zakresu - dla 4 filtrów we wszystkich trzech warstwach konwolucyjnych.

6. Wnioski

Utworzona konwolucyjna sieć neuronowa uczy się poprawnie odnajdywać obiekt na obrazie z zadowalającą jak na rozmiar zestawu danych i liczbę epok przeznaczoną na uczenie trafnością. Jednak trafność jaką nauczona sieć osiągnie jest silnie zależna od współczynnika uczenia i stałej momentum, a dla niektórych wartości współczynnika uczenia – sieć w ogóle się nie nauczy. Wpływ liczby filtrów przypadających na każdą warstwę sieci był – w porównaniu do wpływu współczynnika uczenia i stałej momentum – znikomy, aczkolwiek liczba filtrów również odbijała się na końcowej trafności sieci.

Znalezione przez przeprowadzone eksperymenty optymalne parametry wskazują, że dla współczynnika uczenia $lr \approx 5 * 10^{-3}$, stałej momentum $mc \approx 0.95$, 14 filtrów w pierwszej warstwie konwolucyjnej, i 20 filtrów w warstwach drugiej i trzeciej, sieć powinna uczyć się najszybciej i osiągać najwyższą trafność.

Podczas eksperymentów dla niektórych parametrów zaobserwowane zostały anomalie, w postaci trafności dla zbioru danych walidacyjnych przewyższającej trafność dla zbioru danych uczących. Możliwe, że znikłyby one przy zastosowaniu innego algorytmu uczącego, lub znacznym zwiększeniu liczby epok przeznaczonych na uczenie.

Literatura

[1] R. Zajdel „Sztuczna inteligencja ćw8 sieć jednokierunkowa jednowarstwowa”

[2] R. Zajdel „Sztuczna inteligencja ćw9 sieć jednokierunkowa wielowarstwowa”

[3] R. Zajdel „Sztuczna inteligencja ćw10 przyspieszanie procesu uczenia”

[4] Z. Zhang „Derivation of Backpropagation in Convolutional Neural Network (CNN)”

[5] P. Skalski „Gentle Dive into Math Behind Convolutional Neural Networks”

[6] Dokumentacja biblioteki Tensorflow

[7] Dokumentacja biblioteki Keras