JOURNAL
COMPUTER SCIENCES INSTITUTE

JCSI 23 (2022) 145–151
Received: 10 April 2022
Accepted: 20 April 2022

Comparison of LeNet-5, AlexNet and GoogLeNet models in handwriting recognition

Porównanie modeli LeNet-5, AlexNet i GoogLeNet w rozpoznawaniu pisma ręcznego

Bartosz Michalski*, Małgorzata Plechawska-Wójcik

Department of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract
The aim of the study was to compare the accuracy of handwriting recognition and the time needed to classify data from the test sets. The Lenet-5, AlexNet and GoogLeNet architectures were used for the research. All selected architectures are models of convolutional neural networks. The research was carried out with the use of image databases, handwritten digits MNIST and handwritten letters EMNIST. After the tests, it was found that the GoogLeNet model showed the highest accuracy, and the LeNet-5 the lowest. However, the LeNet-5 model needed the least time to complete the task, and GoogLeNet the most. On the basis of the obtained results, it was found that increasing the complexity of the model positively influences the accuracy of object classification, but significantly increases the demand for computer resources.

Keywords: convolutional neural networks; handwriting classification

Streszczenie
Celem badania było porównanie dokładności rozpoznawania pisma odręcznego oraz czasu potrzebnego na klasyfikację danych ze zbiorów testowych. Do badań wykorzystano architektury Lenet-5, AlexNet i GoogLeNet. Wszystkie wybrane architektury są modelami konwolucyjnych sieci neuronowych. Badania przeprowadzono z wykorzystaniem baz obrazów odręczenie pisanych cyfr MNIST i odręcznie pisanych liter EMNIST. Po wykonaniu badań stwierdzono, że największą dokładnością wykazał się model GoogLeNet, a najmniejszą LeNet-5. Natomiast najmniej czasu na wykonanie zadania potrzebował model LeNet-5, a najwięcej GoogLeNet. Na podstawie otrzymanych wyników stwierdzono, że zwiększanie złożoności modelu wpływa pozytywnie na dokładność klasyfikacji obiektów, ale znacznie zwiększa zapotrzebowanie na zasoby komputera.

Słowa kluczowe: Konwolucyjne sieci neuronowe; klasyfikacja pisma odręcznego

*Corresponding author
Email address: **bartosz.michalski@pollub.edu.pl** (B. Michalski)

1. Wstęp

Historia sztucznych sieci neuronowych (SSN) sięga lat 40. XX wieku, kiedy to McCulloch i Pitts (1943) [18] przedstawili model matematyczny aktywności neuronowej mózgu, a Hebb (1949) [1] stworzył mechanizm uczenia się oparty na wzmocnieniu, aby wyjaśnić uczenie się w ludzkim mózgu. Rosenblatt (1958) [2] stworzył następnie model obliczeniowy elementów przetwarzających mózg, zwanych perceptronami.

Jedną z pierwszych udanych prób użycia głębokiego uczenia było stworzenie w 1989 roku architektury LeNet-5 [3]. W 1998 roku architektura ta wykazała się największą dokładnością w klasyfikacji odręcznie pisanych cyfr wśród obecnych wtedy na rynku rozwiązań. To napędziło dalszy rozwój użycia głębokiego uczenia w sztucznych sieciach neuronowych.

Kolejnym przełomowym momentem było użycie mocy obliczeniowej kart graficznych w trenowaniu sieci i późniejszej klasyfikacji danych. Pierwszym wyraźnym sukcesem było stworzenie architektury AlexNet, która wykazała się o ponad 10% mniejszym poziomem błędów w konkursie ImageNet w 2012 roku względem następnego wyniku w rankingu [4].

Dziś popularność SSN stale rośnie. Szerokie zastosowanie SSN jest możliwe dzięki sposobie działania i budowie, które mają odwzorowywać biologiczny system nerwowy człowieka. SSN są obecne w wielu miejscach, wykorzystywane do codziennej pracy wymagającej analizy danych, klasyfikacji czy sterowania. Dobrymi przykładami mogą być kontrola bagażów na lotnisku [5], rozpoznawanie twarzy [6], rozpoznawanie pisma odręcznego [7], prognozy giełdowe [8], a nawet pogody [9].

Ciągle powstają kolejne architektury cechujące się coraz większą dokładnością wykonywanej pracy. Od roku 2010 do 2017 odbywały się coroczne zawody klasyfikacji obiektów 2D z bazy ImageNet [10]. Aby wygrać SSN musiała osiągnąć możliwe najmniejszy poziom błędu podczas klasyfikacji danych testowych. Omawiane w pracy architektury AlexNet i GoogLeNet zwyciężały odpowiednio w roku 2012 i 2014 [4].

Istnieje wiele narzędzi ułatwiających tworzenie modeli sieci neuronowych takich jak biblioteka *Keras* zintegrowana z *TensorFlow* czy biblioteka *PyTorch*, dzięki którym nawet niedoświadczone osoby mogą szybko zacząć swoją przygodę z głębokim uczeniem

maszynowym i sieciami neuronowymi. Stworzone modele SSN można w prosty sposób wytrenować i przetestować korzystając z wielu dostępnych baz danych m.in. *MNIST* [11]*, EMNIST* [12] *lub ImageNet* [10], zawierających nie tylko proste obiekty, jak zdjęcia obiektów czy pisma ręcznego, ale nawet próbek biologicznych pobranych od pacjentów.

W pracy porównano wspomniane wyżej architektury, czyli Lenet-5, AlexNet i GoogLeNet pod kątem dokładności rozpoznawania odręczenie pisanych liter i cyfr oraz czas potrzebny na klasyfikacje danych testowych. W kolejnych rozdziałach opisane zostały konwolucyjne sieci neuronowe, którymi te architektury są oraz same architektury. Następnie przedstawiono sposób realizacji badań, postawione hipotezy badawcze, implementacje modeli oraz opisano wybrane do badań bazy danych. Na koniec zamieszczone zostały uzyskane wyniki, ich opis oraz wnioski.

2. Konwolucyjne sieci neuronowe

Głównym obszarem, w jakim wykorzystywane są konwolucyjne sieci neuronowe (*CNN*) jest rozpoznawanie wzorców na obrazach. To pozwala zakodować w architekturze cechy charakterystyczne dla obrazu, dzięki czemu sieć lepiej nadaje się do zadań związanych z obrazem - przy jednoczesnym dalszym zmniejszeniu parametrów wymaganych do skonfigurowania modelu [16]. Różne wersje i konfiguracje CNN, potrafią osiągać dokładność ponad 99% podczas klasyfikacji odręczenie pisanych cyfr oraz ponad 95% podczas klasyfikacji odręczenie pisanych liter łacińskich [19].

Na CNN składają się trzy rodzaje warstw. Są to warstwy splotowe (konwolucyjne), warstwy łączące (ang. *Pooling Layers*) i warstwy w pełni połączone (ang. *Fully-Connected Layers*).

Warstwy splotowe, jak nazwa wskazuje, są podstawą CNN. Zawierają w sobie filtry (ang. *kernels*), których zadaniem jest wyodrębnienie cech odróżniających od siebie obrazy. Filtr to dwuwymiarowa tablica wag, która reprezentuje część obrazu. Jest nakładany na obszar obrazu, a następnie obliczany jest iloczyn skalarny między pikselami wejściowymi, a filtrem. Wynik jest podawany do macierzy wyjściowej. Filtr przesuwa się krok po kroku, powtarzając proces, aż przesunie się przez cały obraz. Ostateczne dane wyjściowe są znane, jako mapa funkcji lub mapa aktywacji.

Warstwy łączące mają za zadanie zmniejszenie obrazu. To oznacza mniejszą liczbę parametrów do wyuczenia przez sieć. Dzięki temu model CNN jest prostszy i wydajniejszy. Warstwa łącząca działa na każdej mapie aktywacji na wejściu i skaluje jej wymiarowość za pomocą funkcji „MAX". W większości CNN warstwy łączące mają postać warstw z maksymalnym łączeniem, z filtrami o wymiarowości 2×2, nakładanymi z krokiem 2 wzdłuż przestrzennych wymiarów danych wejściowych. To skaluje mapę aktywacji do 25% oryginalnego rozmiaru [16].

W warstwie w pełni połączonej, połączenia między neuronami wyglądają, że każdy neuron jest połączony z neuronem warstwy poprzedniej. Warstwa ta próbuje wygenerować wyniki klas z mapy aktywacji, które zostaną użyte do klasyfikacji.

3. Wykorzystane Architektury

3.1. LeNet-5

LeNet-5 to architektura, którą Yann LeCun i in. zaprezentowali w 1989 roku [3]. Była jedną z pierwszych CNN. Twórcy wykorzystywali stworzony program do rozpoznawania odręcznie zapisanych kodów pocztowych na listach. Dokładność klasyfikacji wynosiła 90%, co pokazywało, że SSN mogą mieć praktyczne zastosowania. Model składa się z siedmiu warstw, nie licząc warstwy wejścia. Wszystkie warstwy zawierają parametry, które można trenować (wagi). Wejście to obraz o wymiarach 32 x 32 piksele [13]. LeNet-5 definiuje podstawy CNN, jednak w momencie pojawienia się na rynku nie była popularna z powodu braku odpowiedniego sprzętu, a zwłaszcza kart graficznych.

3.2. AlexNet

AlexNet to model zaprezentowany przez Alexa Krizhevskiego i in. w 2012 roku [14]. Architektura ta konkurowała w zawodach ImageNet Large Scale Visual Recognition Challenge (ILSVRC) w tym samym roku, wygrywając je [4]. Współczynnik błędów top-5 modelu wynosił 15,3% i był niższy o 10,8% niż kolejny wynik w konkursie. AlexNet składa się z pięciu warstw splotowych, z czego po niektórych znajdują się warstwy łączące, oraz z trzech w pełni połączonych warstw [14]. Zastosowanie większej liczby warstw było kluczowe dla wydajność, ale potrzebowało wiele zasobów. Było to jednak możliwe dzięki wykorzystaniu kart graficznych w procesie uczenia sieci. Praca naukowa o AlexNet [14] ma duży wpływ na rozwój CNN. Według strony Google Scholar praca była cytowana ponad 90000 razy (styczeń 2022).

3.3. GoogLeNet

GoogLeNet to 22-warstwowa architektura, znana także pod nazwą Inception v1. GoogLeNet zostało zaprezentowana w 2014 roku przez firmę Google [15]. W tym samym roku zwyciężyła konkurs ImageNet [4] z poziomem błędów wynoszącym 6,7%. Największą nowością zaprezentowaną wraz z tą architekturą były moduły incepcji. Moduły te polegają na równoległym działaniu warstw konwolucyjnych, każda z różną dokładnością działania (różne wielkości filtrów). Takie rozłożenie warstw ma pomagać w klasyfikacji obiektów o różnych skalach. Wersja naiwna takiego modułu wykonuje splot na wejściu za pomocą 3 różnych rozmiarów filtrów (1x1, 3x3, 5x5). Dodatkowo wykonywany jest proces maksymalnego łączenia (ang. *max pooling*). Następnie wyjścia są łączone i wysyłane dalej. W celu ograniczenia zapotrzebowania na zasoby twórcy zaproponowali także wersję modułu incepcji z redukcją wymiarowości [15]. Wersja ta polega na ograniczeniu kanałów wejściowych przez dodanie warstw splotowych o rozmiarze 1x1 przed warstwami 3x3 i 5x5. Ar-

chitektura GoogLeNet wykorzystuje dziewięć modułów incepcji z redukcją wymiarowości.

4. Realizacja badań

4.1. Scenariusze Badawcze

Celem badania było porównanie modeli LeNet-5, AlexNet i GoogLeNet pod względem dokładności klasyfikacji odręczenie pisanych cyfr oraz liter alfabetu łacińskiego. Dodatkowo porównane zostały czasy potrzebne na klasyfikacje danych ze zbiorów testowych. W tym celu sformułowano przedstawione poniżej scenariusze badawcze.

Scenariusz 1:

1. Wytrenowanie modeli LeNet-5, AlexNet i GoogLeNet pod kątem klasyfikacji odręcznie pisanych cyfr.
2. Porównanie dokładności (metryka *accuracy*) każdego modelu.
3. Porównanie czasu potrzebnego na klasyfikacje danych ze zbioru testowego dla każdego modelu.

Każdy z modeli przeszedł 20 razy proces uczenia od zera. Do każdego procesu uczenia były wykorzystywane te same parametry dla poszczególnych warstw. Każdy taki proces trwał 20 epok, tak, aby upewnić się, że każdy z modeli zbliżył się do maksimum swojej dokładności. Do badania wykorzystany został zbiór danych MNIST [11], czyli 60000 obrazów uczących i 10000 obrazów testowych.

Scenariusz 2:

1. Wytrenowanie modeli LeNet-5, AlexNet i GoogLeNet pod kątem klasyfikacji odręcznie pisanych liter.
2. Ewaluacja skuteczności przez porównanie dokładności (ang. *accuracy*) każdego modelu.
3. Porównanie czasu potrzebnego na klasyfikacje danych ze zbioru testowego dla każdego modelu.

Badanie zostało przeprowadzone tak jak w scenariuszu pierwszym. Różnicą był zbiór danych. Wykorzystany został zbiór odręczenie pisanych liter ze zbioru EMNIST [12]. Zbiór składa się z 88800 obrazów uczących i 14800 obrazów testowych.

Głównym celem pracy jest sprawdzenie poprawności następujących hipotez:

1. Model oparty o architekturę GoogLeNet wykaże się większą dokładnością w klasyfikacji odręcznie pisanych cyfr.
2. Model oparty o architekturę GoogLeNet wykaże się większą dokładnością w klasyfikacji odręcznie pisanych liter.
3. Model oparty o architekturę LeNet-5 będzie potrzebował mniej czasu na klasyfikację zbioru odręcznie pisanych cyfr.
4. Model oparty o architekturę LeNet-5 będzie potrzebował mniej czasu na klasyfikację odręczenie pisanych liter.

4.2. Implementacja modeli

Modele zostały zaimplementowane w języku Python z użyciem bibliotek Tensorflow i Keras. Implementacje modeli zostały przedstawione kolejno na listingach 1, 2 oraz 3. Dodatkowo dla modelu GoogLeNet zaimplementowana została warstwa incepcji przedstawiona na listingu 4.

Listing 1: Implementacja modelu GoogLeNet

```
input_layer = layers.Input(shape=(32, 32, 3))
input_tensor = layers.experimental.preprocessing.Resizing(224, 224,
    interpolation="bilinear", input_shape=x_train.shape[1:])(input_layer)
temp = layers.Conv2D(64, 7, strides=2, padding='same',
    activation='relu')(input_tensor)
temp = layers.MaxPooling2D(3, strides=2)(temp)
temp = layers.Conv2D(64, 1, strides=1, padding='same',
    activation='relu')(temp)
temp = layers.Conv2D(192, 3, strides=1, padding='same',
    activation='relu')(temp)
temp = layers.MaxPooling2D(3, strides=2)(temp)
temp = inception_layer(temp, filters_1x1=64, filters_3x3_reduce=96,
    filters_3x3=128, filters_5x5_reduce=16, filters_5x5=32, filters_pool=32)
temp = inception_layer(temp, filters_1x1=128, filters_3x3_reduce=128,
    filters_3x3=192, filters_5x5_reduce=32, filters_5x5=96, filters_pool=64)
temp = layers.MaxPooling2D(3, strides=2)(temp)
temp = inception_layer(temp, filters_1x1=192, filters_3x3_reduce=96,
    filters_3x3=208, filters_5x5_reduce=16, filters_5x5=48, filters_pool=64)
auxiliaryOutput1 = layers.AveragePooling2D((5, 5), strides=3)(temp)
auxiliaryOutput1 = layers.Conv2D(128, 1, padding='same',
    activation='relu')(auxiliaryOutput1)
auxiliaryOutput1 = layers.Flatten()(auxiliaryOutput1)
auxiliaryOutput1 = layers.Dense(1024, activation='relu')(auxiliaryOutput1)
auxiliaryOutput1 = layers.Dropout(0.7)(auxiliaryOutput1)
auxiliaryOutput1 = layers.Dense(10, activation='softmax')(auxiliaryOutput1)
temp = inception_layer(temp, filters_1x1=160, filters_3x3_reduce=112,
    filters_3x3=224, filters_5x5_reduce=24, filters_5x5=64, filters_pool=64)
temp = inception_layer(temp, filters_1x1=128, filters_3x3_reduce=128,
    filters_3x3=256, filters_5x5_reduce=24, filters_5x5=64, filters_pool=64)
temp = inception_layer(temp, filters_1x1=112, filters_3x3_reduce=144,
    filters_3x3=288, filters_5x5_reduce=32, filters_5x5=64, filters_pool=64)
auxiliaryOutput2 = layers.AveragePooling2D((5, 5), strides=3)(temp)
auxiliaryOutput2 = layers.Conv2D(128, 1, padding='same',
    activation='relu')(auxiliaryOutput2)
auxiliaryOutput2 = layers.Flatten()(auxiliaryOutput2)
auxiliaryOutput2 = layers.Dense(1024, activation='relu')(auxiliaryOutput2)
auxiliaryOutput2 = layers.Dropout(0.7)(auxiliaryOutput2)
auxiliaryOutput2 = layers.Dense(10, activation='softmax')(auxiliaryOutput2)
temp = inception_layer(temp, filters_1x1=256, filters_3x3_reduce=160,
    filters_3x3=320, filters_5x5_reduce=32, filters_5x5=128, filters_pool=128)
temp = layers.MaxPooling2D(3, strides=2)(temp)
temp = inception_layer(temp, filters_1x1=256, filters_3x3_reduce=160,
    filters_3x3=320, filters_5x5_reduce=32, filters_5x5=128, filters_pool=128)
temp = inception_layer(temp, filters_1x1=384, filters_3x3_reduce=192,
    filters_3x3=384, filters_5x5_reduce=48, filters_5x5=128, filters_pool=128)
temp = layers.GlobalAveragePooling2D()(temp)
temp = layers.Dropout(0.4)(temp)
output = layers.Dense(10, activation='softmax')(temp)
model = Model(inputs=input_layer, outputs=[output,
    auxiliaryOutput1, auxiliaryOutput2])
```

Listing 2: Implementacja modelu LeNet-5

```
from tensorflow import keras
model = keras.models.Sequential()
model.add(keras.layers.Conv2D(6, kernel_size=5, strides=1,
    activation='tanh', input_shape=train_x[0].shape, padding='same'))
model.add(keras.layers.AveragePooling2D())
model.add(keras.layers.Conv2D(16, kernel_size=5, strides=1,
    activation='tanh', padding='valid'))
model.add(keras.layers.AveragePooling2D())
model.add(keras.layers.Flatten())
model.add(keras.layers.Dense(120, activation='tanh'))
model.add(keras.layers.Dense(84, activation='tanh'))
model.add(keras.layers.Dense(10, activation='softmax'))
```

Listing 3: Implementacja modelu AlexNet

```
model = keras.models.Sequential()
model.add(keras.layers.experimental.preprocessing.Resizing(224, 224,
    interpolation="bilinear", input_shape=x_train.shape[1:]))
model.add(keras.layers.Conv2D(96, 11, strides=4, padding='same'))
model.add(keras.layers.Activation('relu'))
model.add(keras.layers.MaxPooling2D(3, strides=2))
model.add(keras.layers.Conv2D(256, 5, strides=4, padding='same'))
model.add(keras.layers.Activation('relu'))
model.add(keras.layers.MaxPooling2D(3, strides=2))
model.add(keras.layers.Conv2D(384, 3, strides=4, padding='same'))
model.add(keras.layers.Activation('relu'))
model.add(keras.layers.Conv2D(384, 3, strides=4, padding='same'))
model.add(keras.layers.Activation('relu'))
model.add(keras.layers.Conv2D(256, 3, strides=4, padding='same'))
model.add(keras.layers.Activation('relu'))
model.add(keras.layers.Flatten())
model.add(keras.layers.Dense(4096, activation='relu'))
model.add(keras.layers.Dropout(0.5))
model.add(keras.layers.Dense(4096, activation='relu'))
model.add(keras.layers.Dropout(0.5))
model.add(keras.layers.Dense(10, activation='softmax'))
```

Listing 4: Implementacja warstwy incepcji z redukcją wymiarowości

```
def inception_layer(temp,
                    filters_1x1,
                    filters_3x3_reduce,
                    filters_3x3,
                    filters_5x5_reduce,
                    filters_5x5,
                    filters_pool):
path1 = layers.Conv2D(filters_1x1, (1, 1), padding='same',
    activation='relu')(temp)
path2 = layers.Conv2D(filters_3x3_reduce, (1, 1), padding='same',
    activation='relu')(temp)
path2 = layers.Conv2D(filters_3x3, (1, 1), padding='same',
    activation='relu')(path2)
path3 = layers.Conv2D(filters_5x5_reduce, (1, 1), padding='same',
    activation='relu')(temp)
path3 = layers.Conv2D(filters_5x5, (1, 1), padding='same',
    activation='relu')(path3)
path4 = layers.MaxPool2D((3, 3), strides=(1, 1), padding='same')(temp)
path4 = layers.Conv2D(filters_pool, (1, 1), padding='same',
    activation='relu')(path4)
return tf.concat([path1, path2, path3, path4], axis=3)
```

4.3. Bazy danych

4.3.1. MNIST

Baza obrazów odręcznie pisanych cyfr MNIST [11] zawiera zestaw uczący 60000 obiektów oraz zestaw testowy 10000 obiektów. Baza ta jest często wykorzystywana do testowania stworzonych modeli sztucznych sieci neuronowych. MNIST została zbudowana na podstawie Specjalnej Bazy Danych 3 NIST i Specjalnej Bazy Danych 1, które zawierają binarne obrazy odręcznie pisanych cyfr [2]. Oryginalne czarno-białe (dwupoziomowe) obrazy z NIST zostały znormalizowane pod względem rozmiaru, aby zmieściły się w polu 20x20 pikseli, zachowując ich proporcje. Uzyskane obrazy zawierają poziomy szarości uzyskane w wyniku zastosowania techniki antyaliasingu, używanej przez algorytm normalizacji. Obrazy zostały wyśrodkowane na obrazie 28x28 przez obliczenie środka masy pikseli i przemieszczenie obrazu tak, aby umieścić ten punkt w środku pola 28x28 [11].

4.3.2. EMNIST

EMNIST [12] (Extended MNIST) to zestaw sześciu baz danych mający na celu zapewnienie trudniejszej do klasyfikacji alternatywy dla zestawu danych MNIST [11]. Znaki Specjalnej Bazy Danych 19 NIST [17] zostały przekonwertowane do formatu, który pasuje do zbioru danych MNIST [11], czyniąc go kompatybilnym z każdą siecią zdolną do pracy z oryginalnym zbiorem danych [12]. Do badań wykorzystano bazę danych liter. Baza ta składa się z 88800 obiektów treningowych oraz 14800 obiektów testowych.

5. Wyniki

Wszystkie testy opisanych wcześniej modeli zostały przeprowadzone na komputerze osobistym wyposażonym w procesor Intel Core i7-6700 oraz 16 GB pamięci RAM. Na urządzeniu zainstalowany jest system operacyjny Windows 10 Pro w wersji 21H1. Do implementacji i wykonania badań na modelach użyty został interpreter Python w wersji 3.9 oraz biblioteka Tensorflow w wersji 2.6.0.

Na podstawie wcześniejszych opisów wybranych modeli można łatwo stwierdzić duże różnice pod względem skomplikowania ich budowy oraz zapotrzebowania na zasoby urządzenia, na którym pracują. Przedstawione poniżej wyniki powinny pokazać, czy używanie nowych i bardziej skomplikowanych rozwiązań może być opłacalne w pracy przy prostszych zadaniach takich jak klasyfikacja odręcznego pisma.

5.1. Dokładność klasyfikacji odręczenie pisanych cyfr

Wszystkie otrzymane wyniki klasyfikacji zbioru testowego MNIST [11] znajdują się w tabeli 1. Najlepszym średnim wynikiem dokładności wykazał się GoogLeNet osiągając 99,04%, a najgorszym, czyli 98,63%, LeNet-5. AlexNet uplasował się pomiędzy nimi z wynikiem 98,78%. Jak widać, różnice w otrzymanych wynikach są małe. Pomiędzy modelem, który wypadł najgorzej, a tym z najlepszym wynikiem, różnica ta, to zaledwie 0,39%. Z drugiej jednak strony pole do poprawy było niewielkie, ponieważ już LeNet-5 osiągnął wysoki poziom dokładności. Wykres pudełkowy przedstawiony na rysunku 1 pokazuje, że wraz ze wzrostem średniej dokładności klasyfikacji, poszerzał się też przedział otrzymanych wyników. Najgorszy wynik dla teoretycznie najlepszego modelu, był gorszy niż ten najniższy modelu LeNet-5.

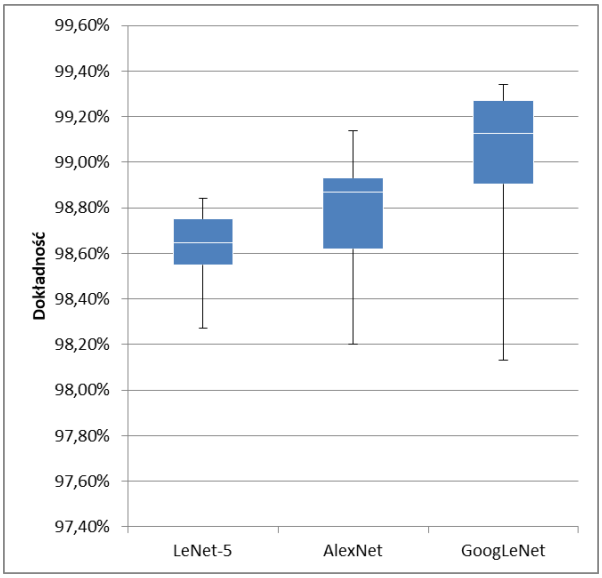
Tabela 1: Wyniki dokładności klasyfikacji odręczenie pisanych cyfr

	Dokładność		
	LeNet-5	AlexNet	GoogLeNet
1	98,84%	98,91%	99,23%
2	98,49%	98,90%	98,73%
3	98,75%	98,70%	98,13%
4	98,60%	99,05%	99,14%
5	98,55%	98,99%	99,14%
6	98,61%	98,86%	99,34%
7	98,34%	98,64%	99,03%
8	98,47%	99,00%	98,93%
9	98,63%	98,56%	98,83%
10	98,73%	98,20%	99,30%
11	98,76%	98,90%	99,27%
12	98,76%	98,56%	99,26%
13	98,75%	98,78%	99,11%
14	98,82%	98,88%	98,83%
15	98,60%	99,02%	98,95%
16	98,77%	99,14%	99,28%
17	98,55%	98,78%	98,57%
18	98,27%	98,56%	99,30%
19	98,66%	98,90%	99,31%
20	98,74%	98,20%	99,11%

5.2. Dokładność klasyfikacji odręczenie pisanych liter

W przypadku klasyfikacji zbioru odręczenie pisanych liter różnice pomiędzy modelami stają się znacznie wyraźniejsze. Średnio GoogLeNet i LeNet-5 osiągnęły kolejno 94,31% i 92,52%, a AlexNet 93,10%. W tym

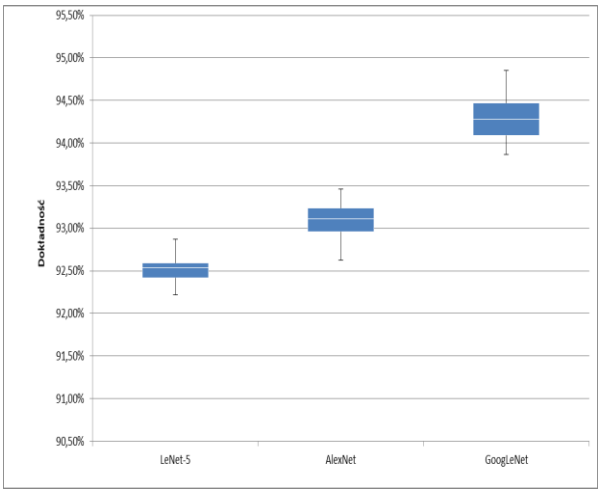
przypadku najlepszy wynik także uzyskał model GoogLeNet, a model LeNet-5 wyprzedził o 1,79%. Przewagę dobrze obrazuje wykres pudełkowy z rysunku 2. Widać na nim, że przedział otrzymanych wyników jest węższy niż w przypadku poprzedniego zbioru danych. Dodatkowo wykres dla modelu GoogLeNet nie pokrywa się z wykresem LeNet-5. Szczegółowe wyniki zostały zamieszone w tabeli 2.



Rysunek 1: Wykres pudełkowy wyników klasyfikacji cyfr.

Tabela 2: Wyniki dokładności klasyfikacji odręczenie pisanych liter

	Dokładność		
	LeNet-5	AlexNet	GoogLeNet
1	92,55%	93,10%	94,85%
2	92,62%	93,12%	94,04%
3	92,58%	93,18%	94,66%
4	92,35%	93,28%	94,04%
5	92,39%	92,98%	94,22%
6	92,52%	92,89%	94,09%
7	92,57%	93,04%	94,13%
8	92,39%	92,88%	94,79%
9	92,60%	92,76%	94,39%
10	92,45%	93,32%	94,57%
11	92,60%	93,24%	94,25%
12	92,43%	93,01%	93,87%
13	92,38%	93,46%	94,31%
14	92,87%	92,62%	94,65%
15	92,57%	93,18%	94,34%
16	92,22%	93,20%	94,43%
17	92,59%	93,43%	94,08%
18	92,68%	93,10%	93,93%
19	92,51%	93,25%	94,17%
20	92,49%	92,85%	94,44%



Rysunek 2: Wykres pudełkowy otrzymanych wyników klasyfikacji liter.

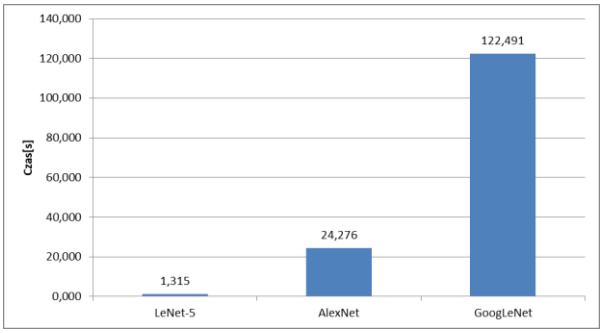
5.3. Czas potrzebny na klasyfikacje zbioru testowego cyfr

Wyniki czasów potrzebnych na klasyfikację zbioru testowego przedstawionych na rysunku 3 pokazują ogromne różnice w wymaganiach, co do urządzenia, na którym pracują poszczególne modele. Model LeNet-5 na klasyfikację zbioru testowego składającego się z 10000 obrazów potrzebował niewiele ponad sekundę, model AlexNet pracował na tym samym zbiorem już około 24 sekund, a GoogLeNet zajęło to ponad dwie minuty. Na podstawie wyników zamieszczonych w tabeli 3 widać, że czas potrzebny dla każdego modelu, w każdej próbie był podobny.

Tabela 3: Wyniki czasu potrzebnego na klasyfikacje zbioru testowego odręczenie pisanych cyfr

	Czas[s]		
	LeNet-5	AlexNet	GoogLeNet
1	1,277	24,932	122,920
2	1,321	24,268	122,400
3	1,347	24,117	122,929
4	1,301	24,528	122,834
5	1,326	24,188	122,795
6	1,300	24,072	122,161
7	1,299	24,231	122,089
8	1,319	24,145	122,082
9	1,328	24,177	122,830
10	1,304	24,077	122,240
11	1,385	24,135	122,586
12	1,309	24,244	122,257
13	1,305	24,180	123,006
14	1,319	24,900	122,167
15	1,315	24,447	122,272
16	1,318	24,119	122,612
17	1,275	24,126	122,410

18	1,348	24,232	122,332
19	1,305	24,200	122,331
20	1,296	24,209	122,559



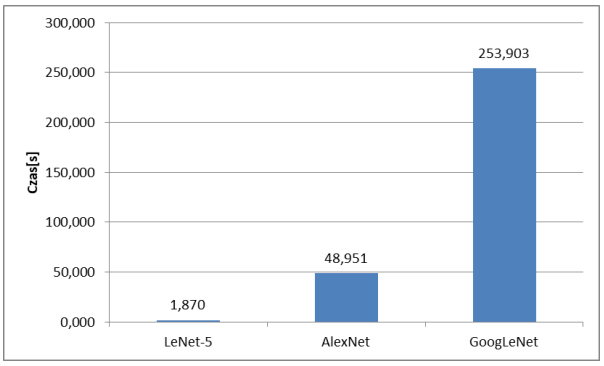
Rysunek 3: Porównanie średnich wartości czasu potrzebnego do klasyfikacji zbioru testowego cyfr.

5.4. Czas potrzebny na klasyfikacje zbioru testowego liter

Zwiększenie możliwych klas do sklasyfikowania z 10 do 26 oraz zbioru testowego z 10000 do 14800 spowodowała podwojenie czasu pracy modeli AlexNet i GoogLeNet, natomiast dla modelu LeNet-5 oznaczało to tylko około pół sekundy dłużej. Na rysunku 4 zamieszczone zostały średnie wyniki czasów potrzebnych na klasyfikację zbioru testowego, a w tabeli 4 zebrane są wszystkie otrzymane czasy.

Tabela 4: Wyniki czasu potrzebnego na klasyfikacje zbioru testowego odręczenie pisanych liter

	Czas[s]		
	LeNet-5	AlexNet	GoogLeNet
1	1,889	49,420	254,971
2	1,843	48,841	253,889
3	1,902	48,851	253,868
4	1,849	49,174	253,994
5	1,874	48,797	254,179
6	1,810	48,719	254,173
7	1,828	48,853	254,017
8	1,872	48,789	253,249
9	1,789	49,536	253,561
10	1,875	48,845	253,135
11	1,889	49,089	254,298
12	1,790	48,888	254,829
13	1,901	48,751	253,322
14	1,916	49,050	254,086
15	1,868	48,861	253,221
16	1,862	48,668	253,878
17	1,856	48,944	253,597
18	1,828	49,173	254,100
19	1,910	48,761	253,839
20	2,050	49,017	253,849



Rysunek 4: Porównanie średnich wartości czasu potrzebnego do klasyfikacji zbioru testowego liter.

6. Wnioski

Przed rozpoczęciem badań postawione zostały tezy mówiące o tym, że model GoogLeNet będzie dokładniejszy w klasyfikacji odręczenie pisanych liter jak i cyfr od modeli AlexNet i LeNet-5. Dodatkowo sformułowano tezy o tym, że LeNet-5 będzie szybszy od pozostałych dwóch modeli podczas klasyfikacji zarówno zbioru odręczeni pisanych cyfr jak i zbioru odręczenie pisanych liter. Na podstawie uzyskanych wyników można stwierdzić, że wszystkie postawione tezy są prawdziwe, ponieważ model GoogLeNet w obu przypadkach wykazał się większą dokładnością od pozostałych modeli, a LeNet-5 był ze wszystkich zdecydowanie najszybszy. Można też stwierdzić stosunkowo małe korzyści pod względem dokładności klasyfikacji wybranych zbiorów. W najlepszym przypadku uzyskano około 1,8% dla zbioru liter. Przychodzi to jednak wielkim kosztem użycia zasobów komputera. Na klasyfikacje zbioru testowego cyfr, model GoogLeNet potrzebował sto razy więcej czasu niż dużo prostszy model LeNet-5, a w przypadku zbioru liter jest nawet gorzej. Trzeba jednak wziąć pod uwagę fakt, że dwa nowsze z tego zestawienia modele (AlexNet i GoogLeNet), były projektowane z myślą o klasyfikacji dużo bardziej skomplikowanych obiektów znajdujących się na zdjęciach, z którymi model LeNet-5 mógłby mieć duży problem. Można bezpiecznie powiedzieć, że w przypadku klasyfikacji odręcznego pisma, korzyści uzyskane z użycia bardziej skomplikowanych rozwiązań są za małe względem kosztów jakie trzeba ponieść, aby ich użycie było opłacalne.

Literatura

[1] D. O. Hebb, The organisation of behaviour: a neuropsychological theory. New York: Science Editions (*1949*).

[2] F. Rosenblatt, The perceptron: a probabilistic model for information storage and organization in the brain. Psychological review 65(6) (1958) 386.

[3] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard., W. Hubbard, L. D. Jackel, Backpropagation applied to handwritten zip code recognition. Neural computation 1(4) (1989) 541-551.

[4] O. Russakovsky, J. Deng, H. Su, et al. ImageNet Large Scale Visual Recognition Challenge. Int J Comput

Vis 115 (2015) 211–252. https://doi.org/10.1007/s11263-015-0816-y

[5] Ü. Budak, A. Şengür, U. Halici, Deep convolutional neural networks for airport detection in remote sensing images. 26th Signal Processing and Communications Applications Conference (SIU) (2018) 1-4, doi: 10.1109/SIU.2018.8404195.

[6] M. J. Aitkenhead, A. J. S. McDonald. A neural network face recognition system. Engineering Applications of Artificial Intelligence 16(3) (2003) 167-176.

[7] D. S. Maitra, U. Bhattacharya, S. K. Parui, CNN based common approach to handwritten character recognition of multiple scripts. 13th International Conference on Document Analysis and Recognition (ICDAR) (2015) 1021-1025, doi: 10.1109/ICDAR.2015.7333916.

[8] K. Nygren, Stock prediction–a neural network approach. Royal Instiute of Technology (2004) 1-34.

[9] S. S. Baboo, I. K. Shereef, An efficient weather forecasting system using artificial neural network. International journal of environmental science and development 1(4) (2010) 321.

[10] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, L. Fei-Fei, Imagenet: A large-scale hierarchical image database. In 2009 IEEE conference on computer vision and pattern recognition (2009) 248–255.

[11] Y. LeCun, C. Cortes, The MNIST database of handwritten digits (2005).

[12] G. Cohen, S. Afshar, J. Tapson, A. van Schaik, EMNIST: an extension of MNIST to handwritten letters (2017) arXiv:1702.05373.

[13] Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, Gradient-based learning applied to document recognition in Proceedings of the IEEE 86(11) (1998) 2278-2324, doi: 10.1109/5.726791.

[14] A. Krizhevsky, I. Sutskever, G. E. Hinton, Imagenet classification with deep convolutional neural networks. In F. Pereira, C. Burges, L. Bottou, K. Weinberger, eds., Advances in Neural Information Processing Systems 25. Curran Associates (2012) 1097–1105. arXiv:1803.01164

[15] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, Going Deeper with Convolutions (2014) arXiv:1409.4842.

[16] K. O'Shea, R. Nash, An introduction to convolutional neural networks (2015) arXiv preprint arXiv:1511.08458.

[17] Grother, P. J, NIST special database 19. Handprinted forms and characters database, National Institute of Standards and Technology (1995).

[18] W. S. McCulloch, W. Pitts, A logical calculus of the ideas immanent in nervous activity. The bulletin of mathematical biophysics *5*(4) (1943) 115-133.

[19] E. Lukasik, M. Charytanowicz, M. Milosz, M. Tokovarov, M. Kaczorowska, D. Czerwinski, T. Zientarski, Recognition of handwritten Latin characters with diacritics using CNN. Bulletin of the Polish Academy of Sciences. Technical Sciences *69*(1) (2021).