


<p>POLITECHNIKA WROCŁAWSKA</p>  <p>Wydział Informatyki i Telekomunikacji</p>	<p>Wydział: Informatyki i Telekomunikacji Kierunek: Informatyczne Systemy Automatyki Stopień: II Rok Akademicki: 2023/2024 Rok studiów: 1 Termin: Środa 15:15 - 16:55</p>
<p align="center">Algorytmy uczenia maszynowego [W04ISA-SM0004G] Projekt 2 – 3D MNIST</p>	
<p>Prowadzący: dr mgr inż. Michał Zmonarski</p>	<p>Wykonujący sprawozdanie: inż. Mateusz Sabuk 259005</p>
<p>Data rozpoczęcia projektu: 2024-04-06</p>	<p>Data oddania sprawozdania: 2024-06-19</p>

Spis treści

1	Wstęp	2
2	Przebieg wykonania projektu	2
2.1	Użyte narzędzia	2
2.2	Ogólne działanie	3
2.3	Poszczególne funkcjonalności	3
2.3.1	Trenowanie	3
2.4	Parametry sieci	6
2.4.1	Wizualizacja	6
2.5	Przeprowadzone testy i ich wyniki	6
3	Perspektywy na przyszłość	9
4	Wnioski	10
5	Kod	10

1 Wstęp

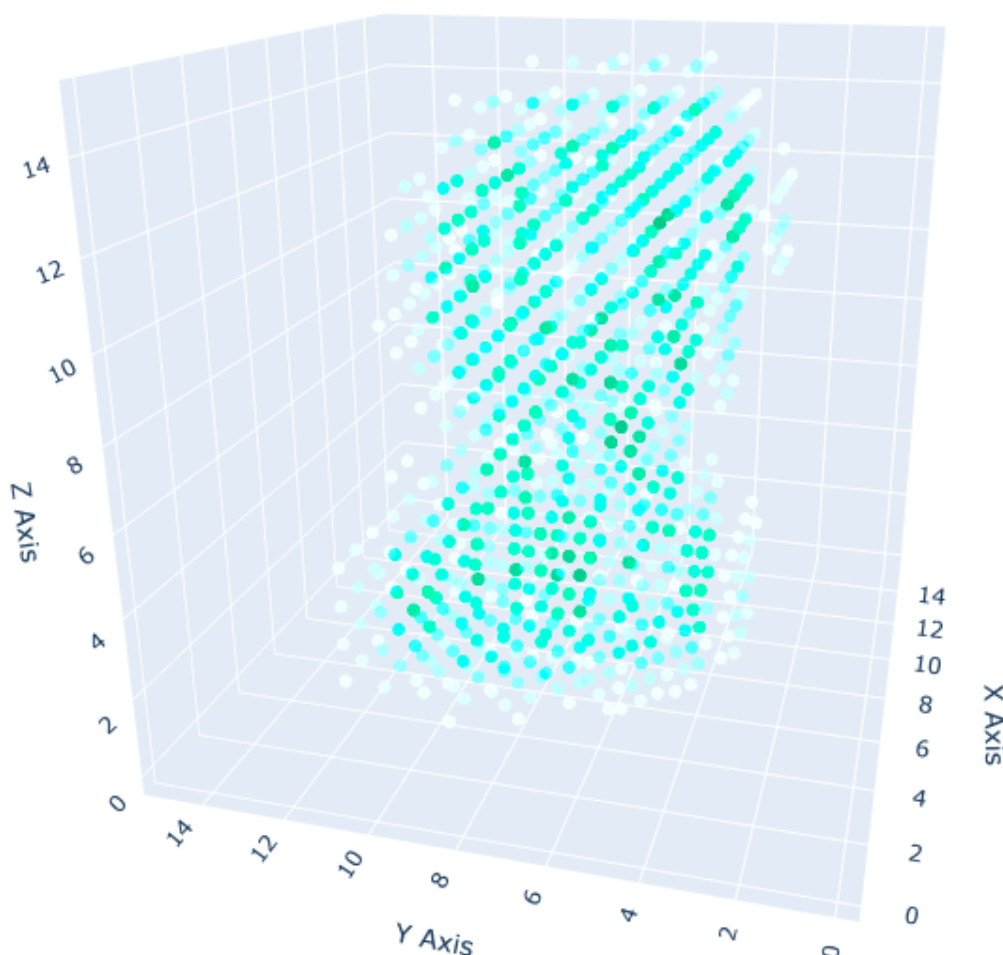
Projekt skupił się na uczeniu maszynowym z wykorzystaniem zbioru danych 3D MNIST. Zbiór pochodził ze strony kaggle. Celem projektu było utworzenie modelu sieci neuronowej, zdolnej do klasyfikacji danych z 3D MNIST. Zostało to osiągnięte z wykorzystaniem trójwymiarowych sieci konwolucyjnych z biblioteki keras.

Wszystkie pliki projektu znajdują się w repozytorium github.

2 Przebieg wykonania projektu

2.1 Użyte narzędzia

Zbiór danych użytych w projekcie to 3D MNIST, czyli zbiór trójwymiarowych wektorów o wymiarach 16x16x16. Każdy przedstawia pisaną cyfrę w trójwymiarowej postaci. Wizualizacja przykładowego elementu zbioru widoczna jest na rysunku ??.



Rysunek 1: Wizualizacja wektora odpowiadającego cyfrze 8

Kod został napisany przy pomocy środowiska jupyter notebook w języku python3. Użyte moduły języka python:

- h5py - pozyskanie danych z pliku,
- pickle - zapis i odczyt zmiennych do pliku,
- tensorflow keras - uczenie maszynowe,
- numpy - praca z macierzami,
- matplotlib - wizualizacja w postaci wykresów,
- plotly - interaktywna wizualizacja danych,
- sklearn - tworzenie macierzy pomyłek.

Do symulacji została użyta platforma google colab, która pozwoliła na wykorzystanie lepszego sprzętu, co przełożyło się na znacznie szybsze uczenie sieci. W trakcie przeprowadzania projektu zaobserwowano aż 30-krotne przyspieszenie uczenia względem użytego laptopa.

2.2 Ogólne działanie

Cały kod zawarty w jupyter notebook znajduje się w sekcji 5, a każdy fragment ma za zadanie kolejno:

import - zaimportowanie modułów python,

funkcje - funkcje użyte dalej w kodzie,

dane - pobranie danych z pliku, sformatowanie i wyświetlenie przykładu,

trening - utworzenie schematu sieci i trenowanie jej,

wizualizacja - przedstawienie wyników uczenia.

2.3 Poszczególne funkcjonalności

2.3.1 Trenowanie

Część kodu zawierająca trenowanie modeli jest główną częścią programu. Następuje w niej iteracja przez wszystkie zestawy parametrów podane w tablicy. Wewnątrz pętli tworzony jest model według tych parametrów, a następnie trenowany. Jeśli w którymkolwiek momencie wystąpi błąd, zostaje on zapisany i program przechodzi do następnego zestawu parametrów.

Dla każdego modelu zapisane zostają celności i straty, treningowe i walidacyjne, w postaci wykresów. Dodatkowo tworzone są macierze pomyłek. Trzy przykłady dla jednego modelu widoczne są na rysunkach 2, 3 i 4.

Podczas trenowania do pliku .pkl zapisywane są dane na temat każdego z modeli. Są to:

zmienne - parametry użyte do utworzenia modelu,

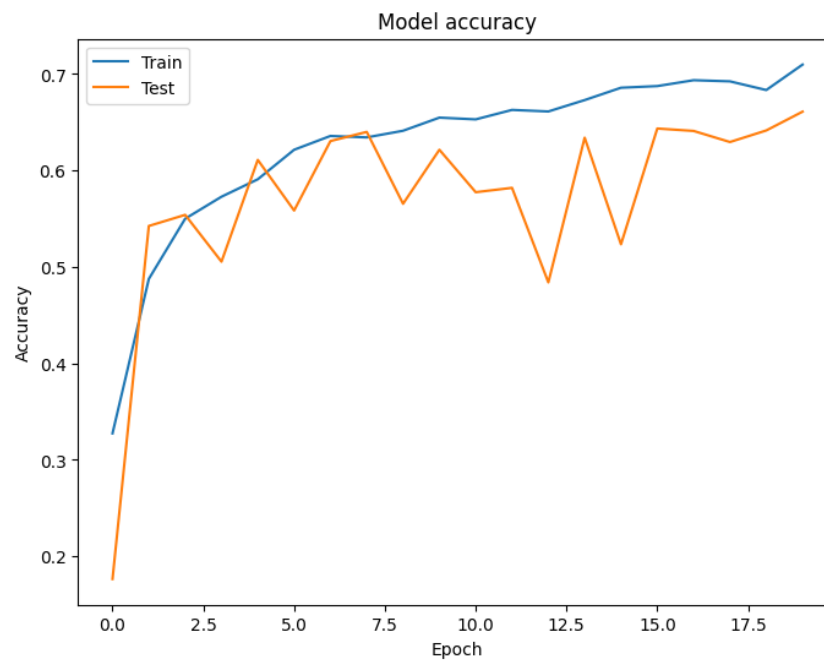
czas - czas uczenia sieci,

accuracy - celność treningowa,

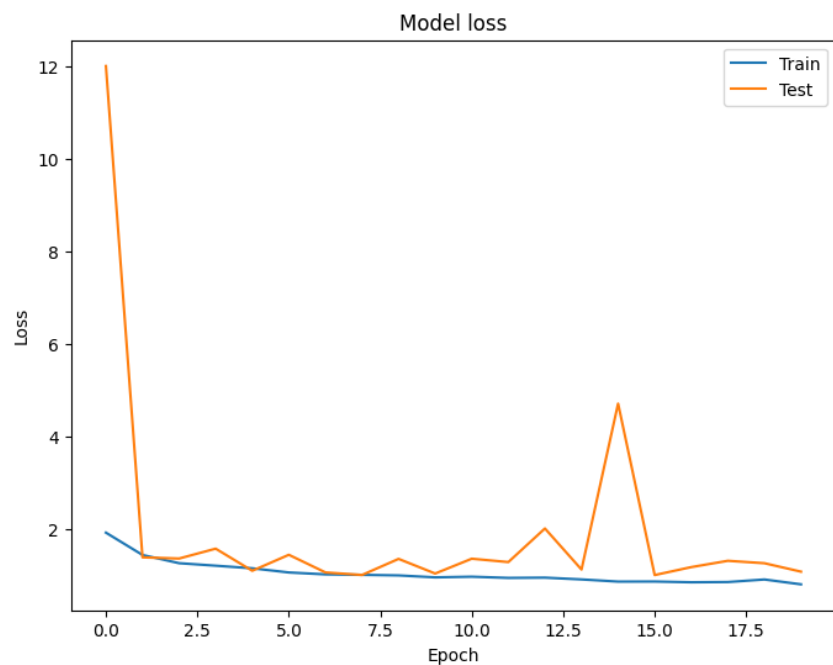
val_accuracy - celność walidacyjna,

loss - strata treningowa,

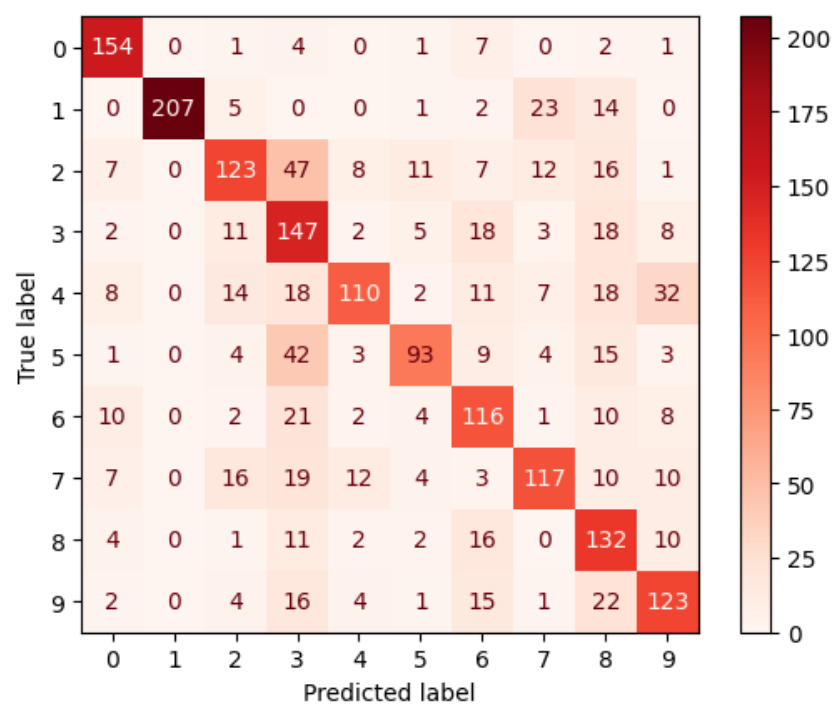
val_loss - strata walidacyjna.



Rysunek 2: Celność przykładowego modelu



Rysunek 3: Strata przykładowego modelu



Rysunek 4: Macierz pomyłek przykładowego modelu

2.4 Parametry sieci

Wspomniane wyżej parametry, według których tworzone są sieci, są w postaci słowników wartości:

filters - tablica wymiarów przestrzeni wyjściowych poszczególnych warstw,

kernel_sizes - tablica rozmiarów jąder warstw konwolucyjnych,

pool_sizes - tablica rozmiarów jąder warstw poolingowych,

dense - rozmiar gęstej warstwy końcowej,

dropout - współczynnik dropoutu,

learning_rate - współczynnik learning rate,

epochs - liczba epoch,

batch_size - rozmiar partii.

Warto zauważyć, że wszystkie tablice, czyli **filters**, **kernel_sizes** i **pool_sizes** muszą być tej samej długości. W innym przypadku utworzenie modelu się nie powiedzie.

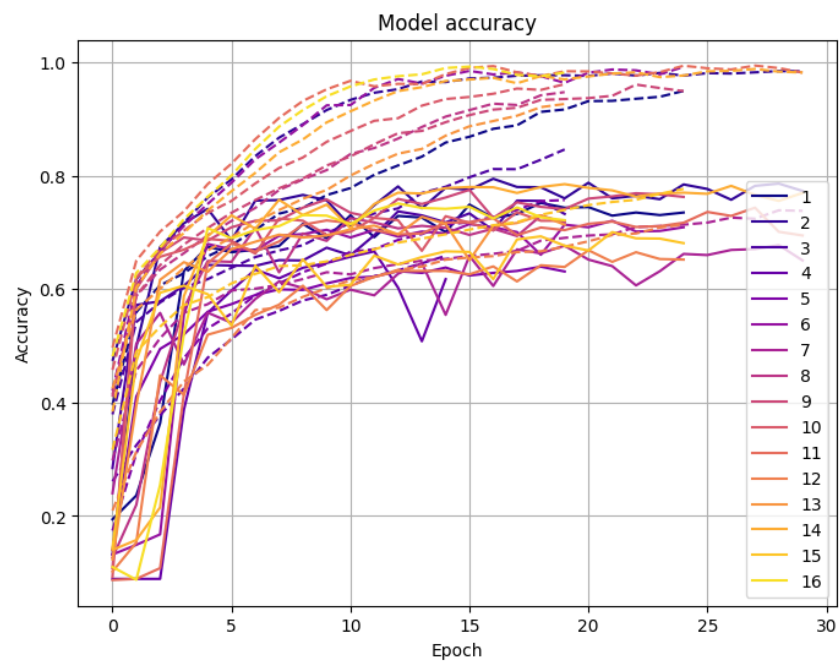
2.4.1 Wizualizacja

Poza wizualizacją przy pomocy funkcji "plot_model" podczas uczenia, możliwa jest także wizualizacja i porównanie wszystkich dostępnych danych. Dane mogą pochodzić z aktualnej sesji, lub z pliku .pkl co pozwala na pominięcie ponownego uczenia w celu analizy danych.

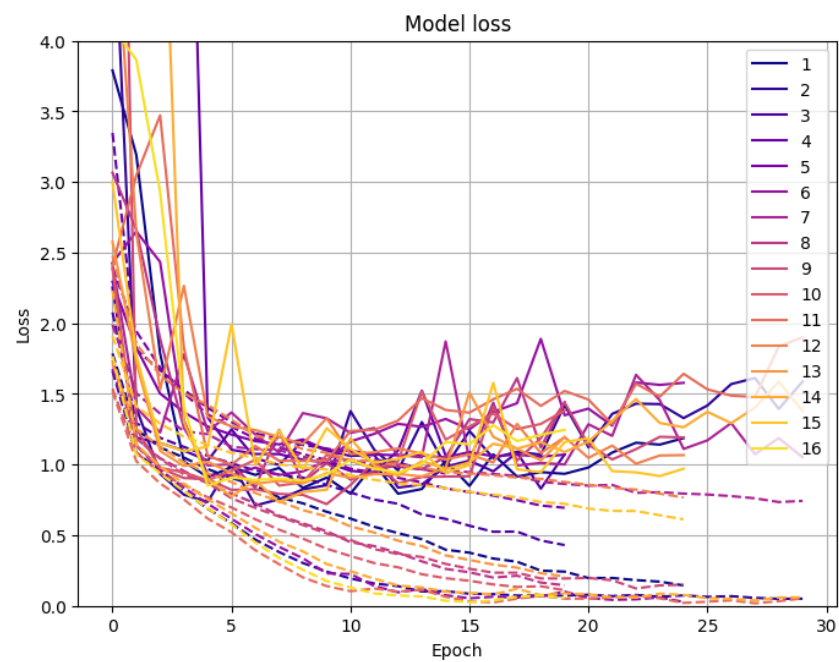
Tworzone wykresy są wykresami słupkowymi zawierającymi czas uczenia, średnią celność walidacyjną z domyślnym pominięciem pierwszych 5 wartości oraz średnią stratę także z pomijaniem 5 pierwszych wartości.

2.5 Przeprowadzone testy i ich wyniki

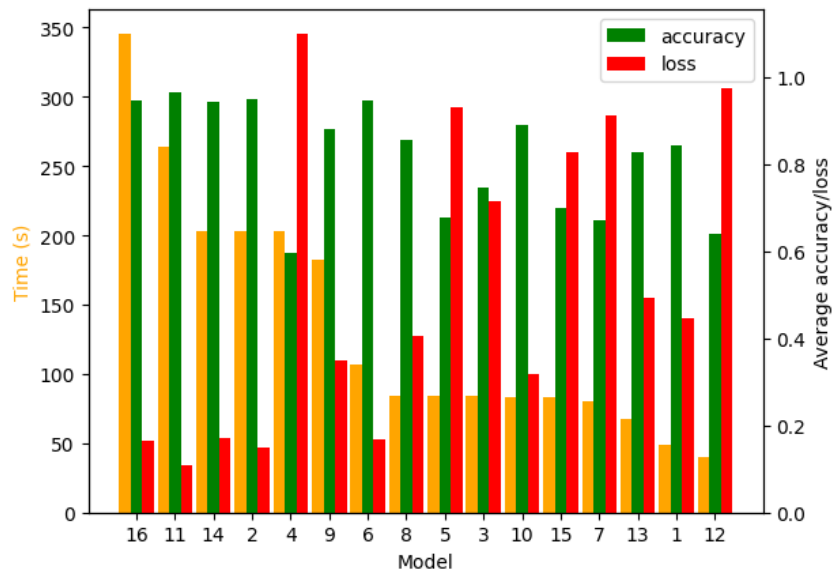
Testy zostały przeprowadzone dla różnych modeli o parametrach wylosowanych przez popularny transformer tekstowy chat-gpt. Wykresy utworzone przy pomocy tych danych widoczne są na rysunkach od 5 do 7. Linie przerywane odpowiadają wartościom testowym, a ciągłe walidacyjnym.



Rysunek 5: Celność dla szesnastu testowanych modeli

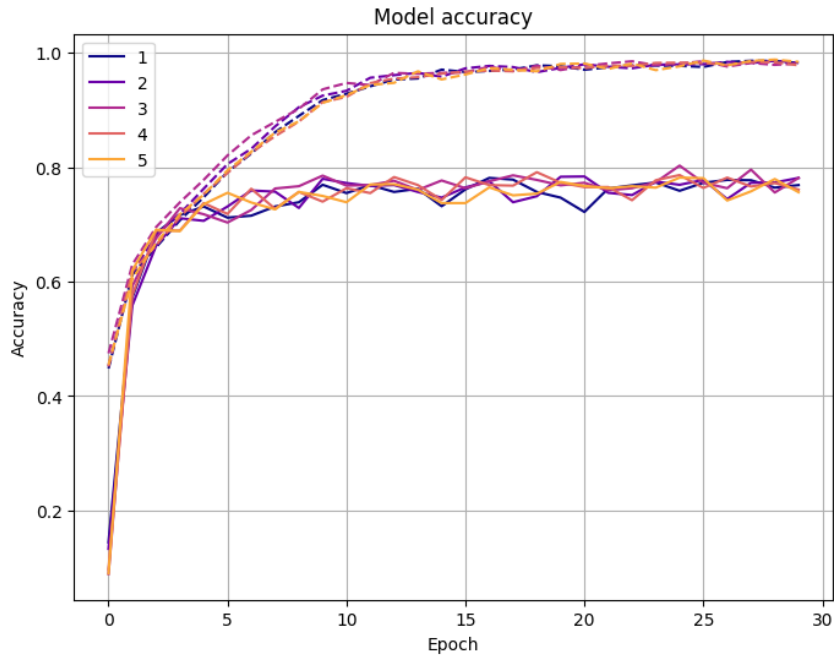


Rysunek 6: Straty dla szesnastu testowanych modeli

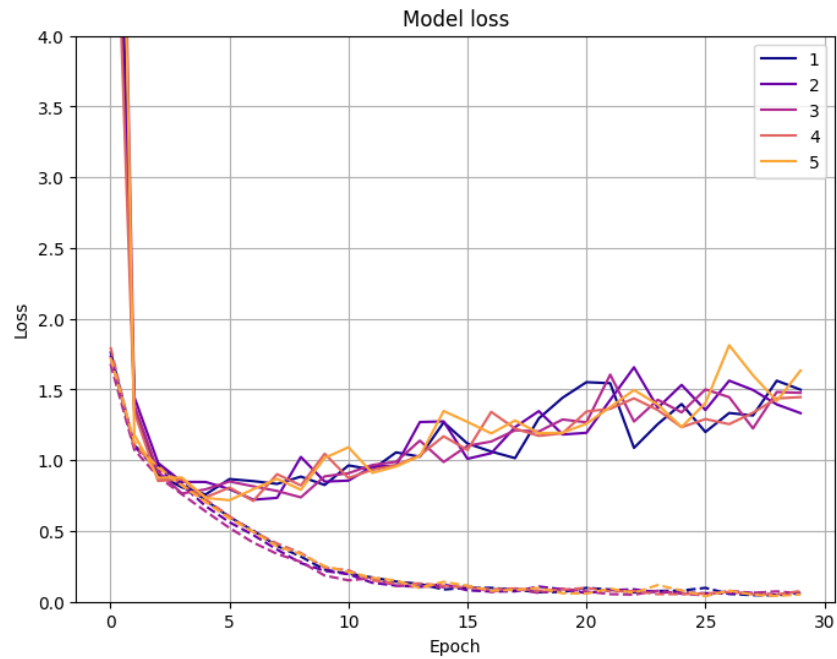


Rysunek 7: Czas i średnie wartości dla szesnastu testowanych modeli

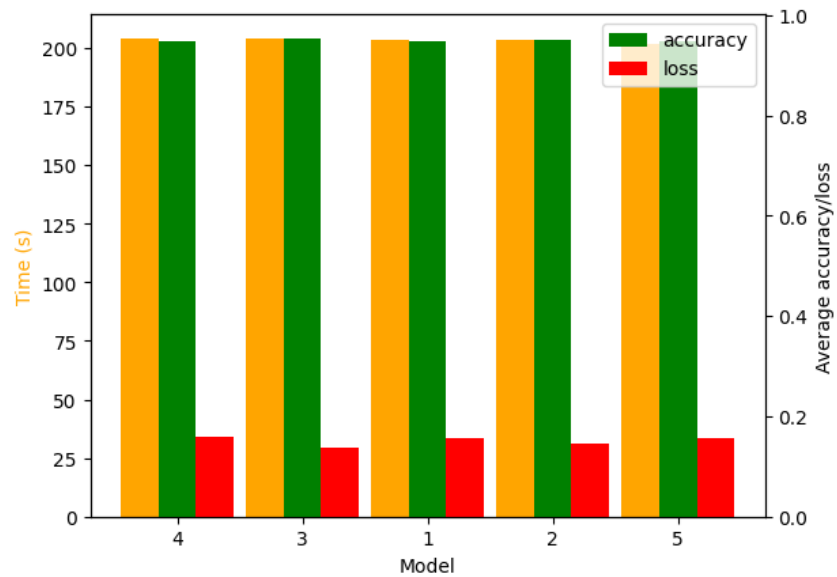
Przeprowadzono także testy dla pięciu modeli o takich samych parametrach. Były to parametry skopiowane z modelu z poprzedniej części, który poradził sobie lepiej niż większość względem wszystkich kryteriów.



Rysunek 8: Celność dla 5 modeli o tych samych parametrach



Rysunek 9: Straty dla 5 modeli o tych samych parametrach



Rysunek 10: Czas i średnie wartości dla 5 modeli o tych samych parametrach

3 Perspektywy na przyszłość

Kolejne kroki w rozwijaniu projektu mogłyby skupiać się na:

- możliwości dokładniejszego sparametryzowania tworzenia modeli, żeby mieć większą kontrolę nad tworzonymi modelami,

- przeprowadzenie dokładniejszych i bardziej powtarzalnych testów w celu możliwości opisanie poszczególnych zmian na działanie sieci,
- przeniesienie programu z jupyter notebook do samodzielnego skryptu python w celu ułatwienia pracy z narzędziem

4 Wnioski

- Nie jest łatwe stwierdzenie dokładnych zależności w działaniu konkretnych parametrów sieci neuronowej.
- Mimo wielu zmian parametrów, prawie żaden z modeli nie uzyskał zdolności klasyfikującej na poziomie większym niż 0.8. Może być to związane z niewłaściwą budową samej sieci.
- Sieci o tej samej budowie nie różniły się ostatecznie od siebie znacznie wynikami mimo losowości w ich wagach i obciążeniach. Znaczący wpływ na działanie sieci ma jej budowa, a nie losowość wewnętrznych wartości.
- Mimo poruszonego tego tematu na dyskusjach na stronie kaggle nie zauważono dużych pomyłek między cyframi 6 i 9.
- Cyfra 1, przez to, że jest po prostu odcinkiem, jest najprostsza do rozpoznania i najczęściej rozpoznawana poprawnie.

5 Kod

Import:

```
1 import h5py
2 import pickle
3
4 import time
5
6 import numpy as np
7
8 from tensorflow.keras.models import Sequential
9 from tensorflow.keras.layers import *
10 from tensorflow.keras.optimizers import Adam
11
12 import matplotlib.pyplot as plt
13 import plotly.graph_objects as go
14 from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
```

Funkcje:

```
1 def create_model(variables):
2     model = Sequential()
3
4     model.add(Input(shape=(16, 16, 16, 1)))
5
6     # Create convolutional layers
7     for filter_num, kernel_size, pool_size in zip(variables["filters"], variables["
    ↪ kernel_sizes"], variables["pool_sizes"]):
```

```

8         model.add(Conv3D(filters=filter_num, kernel_size=kernel_size, activation='relu'
9             ↪ , padding='same'))
10        model.add(BatchNormalization())
11        model.add(MaxPooling3D(pool_size=pool_size, padding='same'))
12
13        # Flatten the output from the convolutional layers
14        model.add(Flatten())
15
16        # Fully connected layer
17        model.add(Dense(units=variables["dense"], activation='relu'))
18        model.add(Dropout(variables["dropout"])) # Adding dropout for regularization
19
20        # Output layer
21        model.add(Dense(units=10, activation='softmax'))
22
23        # Compile the model
24        model.compile(optimizer=Adam(learning_rate=variables["learning_rate"]), loss='
25            ↪ sparse_categorical_crossentropy', metrics=['accuracy'])
26
27        return model
28
29    def train_model(model, variables):
30        return model.fit(X_train, y_train, epochs=variables["epochs"], batch_size=
31            ↪ variables["batch_size"], validation_data=(X_test, y_test))
32
33    def plot_model(model, training_history, model_name):
34        # Plot training & validation accuracy values
35        plt.figure(figsize=(8, 6))
36        plt.plot(training_history.history['accuracy'])
37        plt.plot(training_history.history['val_accuracy'])
38        plt.title('Model_{}_accuracy'.format(model_name))
39        plt.ylabel('Accuracy')
40        plt.xlabel('Epoch')
41        plt.legend(['Train', 'Test'])
42        plt.savefig(f'output/{model_name}_accuracy.png', bbox_inches='tight')
43
44        # Plot training & validation loss values
45        plt.figure(figsize=(8, 6))
46        plt.plot(training_history.history['loss'])
47        plt.plot(training_history.history['val_loss'])
48        plt.title('Model_{}_loss'.format(model_name))
49        plt.ylabel('Loss')
50        plt.xlabel('Epoch')
51        plt.legend(['Train', 'Test'])
52        plt.savefig(f'output/{model_name}_loss.png', bbox_inches='tight')
53
54        # Predict the classes on the test set
55        y_pred = model.predict(X_test)
56        y_pred_classes = y_pred.argmax(axis=1)
57        # Compute the confusion matrix
58        cm = confusion_matrix(y_test, y_pred_classes)
59        # Display the confusion matrix
60        disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=range(10))

```

```

58     disp.plot(cmap=plt.cm.Reds)
59     plt.savefig(f'output/{model_name}_cm.png', bbox_inches='tight')
60
61 def display_3d_vector(data):
62     data = data.reshape(16,16,16)
63
64     x, y, z = np.indices(data.shape)
65     values = data.flatten()
66     x = x.flatten()
67     y = y.flatten()
68     z = z.flatten()
69     values = [f"rgba(20,{x},129,{x})" for x in values]
70
71     fig = go.Figure(data=go.Scatter3d(
72         x=x,
73         y=y,
74         z=z,
75         mode='markers',
76         marker=dict(
77             size=4,
78             color=values,
79             colorbar=dict(title='Value')
80         )
81     ))
82
83     fig.update_layout(
84         autosize=False,
85         width=800,
86         height=800,
87         scene=dict(
88             xaxis_title='X_Axis',
89             yaxis_title='Y_Axis',
90             zaxis_title='Z_Axis'
91         )
92     )
93     fig['layout']['showlegend'] = False
94     fig.show()
95
96
97 def save_data_to_file(data, data_name="data"):
98     with open(f'output/{data_name}.pkl', 'wb') as f:
99         pickle.dump(data, f)
100
101 def get_data_from_file(data_name="data"):
102     with (open(f'output/{data_name}.pkl', "rb")) as f:
103         data = pickle.load(f)
104     return data

```

Dane:

```

1 with h5py.File("data/full_dataset_vectors.h5", "r") as hf:
2     X_train_vec = hf["X_train"][:]
3     y_train = hf["y_train"][:]

```

```

4     X_test_vec = hf["X_test"][:]
5     y_test = hf["y_test"][:]
6
7     X_train_vec.shape, y_train.shape, X_test_vec.shape, y_test.shape
8
9     # Reshape the vectors
10    X_train = X_train_vec.reshape(-1, 16, 16, 16)
11    X_test = X_test_vec.reshape(-1, 16, 16, 16)
12
13    X_train.shape, X_test.shape
14
15    # Display a 3d vector and print its real value
16    display_n = 299
17    print(y_test[display_n])
18    display_3d_vector(X_test[display_n])

```

Trening:

```

1  variables_array = [
2      # Example
3      {
4          "filters": [16, 32, 64],
5          "kernel_sizes": [3, 3, 3],
6          "pool_sizes": [2, 2, 2],
7          "dense": 256,
8          "dropout": 0.5,
9          "learning_rate": 0.001,
10         "epochs": 25,
11         "batch_size": 64,
12     },
13 ]
14
15  data = []
16  for i, variables in enumerate(variables_array):
17      try:
18          model = create_model(variables)
19          start = time.time()
20          training_history = train_model(model, variables)
21          end = time.time()
22          plot_model(model, training_history, str(i).zfill(2))
23
24          data.append({
25              "variables": variables,
26              "time": end-start,
27              "accuracy": training_history.history['accuracy'],
28              "val_accuracy": training_history.history['val_accuracy'],
29              "loss": training_history.history['loss'],
30              "val_loss": training_history.history['val_loss']})
31      except Exception as e:
32          print(e)
33          data.append(f"Model_{nr}_{str(i).zfill(2)}_error:_{e}")
34  save_data_to_file(data)

```

Wizualizacja:

```
1  # import data from file
2  data = get_data_from_file("same")
3  len(data)
4
5
6  # Plot all accuracies and losses
7  plots_name = "batch_same"
8
9  colormap = plt.get_cmap('plasma')
10 num_colors = len(data)
11
12 plt.figure(figsize=(8, 6))
13 for i, model in enumerate(data):
14     color = colormap(i / num_colors)
15     plt.plot(model["accuracy"], "--", color=color)
16     plt.plot(model["val_accuracy"], color=color, label=f"{i+1}")
17 plt.grid()
18 plt.title('Model_accuracy')
19 plt.ylabel('Accuracy')
20 plt.xlabel('Epoch')
21 plt.legend()
22 plt.savefig(f'output/{plots_name}_accuracy.png', bbox_inches='tight')
23
24 plt.figure(figsize=(8, 6))
25 for i, model in enumerate(data):
26     color = colormap(i / num_colors)
27     plt.plot(model["loss"], "--", color=color)
28     plt.plot(model["val_loss"], color=color, label=f"{i+1}")
29 plt.grid()
30 ax = plt.gca()
31 ax.set_ylim([0, 4])
32 plt.title('Model_loss')
33 plt.ylabel('Loss')
34 plt.xlabel('Epoch')
35 plt.legend()
36 plt.savefig(f'output/{plots_name}_loss.png', bbox_inches='tight')
37
38
39 # Display Bar Plots
40 rescale = lambda y: (y - np.min(y)) / (np.max(y) - np.min(y))
41 avg = lambda x: sum(x) / len(x)
42
43 x = [x+1 for x in [*range(len(data))]]
44 accuracy_y = [avg(x["accuracy"][5:]) for x in data]
45
46 my_cmap = plt.get_cmap("plasma")
47 plt.figure(figsize=(8, 6))
48 plt.bar(x, accuracy_y, color=my_cmap(rescale(x)))
49 plt.title('Average_accuracy')
50 plt.ylabel('Accuracy')
51 plt.xlabel('Model')
```

```

52 plt.xticks(np.arange(1,len(data)+1), np.arange(1,len(data)+1))
53 plt.savefig(f'output/{plots_name}_avg_accuracy.png', bbox_inches='tight')
54
55 loss_y = [avg(x["loss"][5:]) for x in data]
56
57 plt.figure(figsize=(8, 6))
58 plt.bar(x, loss_y, color=my_cmap(rescale(x)))
59 plt.title('Average_loss')
60 plt.ylabel('Loss')
61 plt.xlabel('Model')
62 plt.xticks(np.arange(1,len(data)+1), np.arange(1,len(data)+1))
63 plt.savefig(f'output/{plots_name}_avg_loss.png', bbox_inches='tight')
64
65 time_y = [x["time"] for x in data]
66
67 plt.figure(figsize=(8, 6))
68 plt.bar(x, time_y, color=my_cmap(rescale(x)))
69 plt.title('Training_time')
70 plt.ylabel('Time(s)')
71 plt.xlabel('Model')
72 plt.xticks(np.arange(1,len(data)+1), np.arange(1,len(data)+1))
73 plt.savefig(f'output/{plots_name}_training_time.png', bbox_inches='tight')
74
75 zipped_model = list(zip(x, accuracy_y, loss_y, time_y))
76 sorted_model = reversed(sorted(zipped_model, key = lambda t: t[3]))
77 sorted_x, sorted_accuracy_y, sorted_loss_y, sorted_time_y = zip(*sorted_model)
78
79 plt.figure(figsize=(8, 6))
80 width = 0.3
81 fig, ax1 = plt.subplots()
82 ax2 = ax1.twinx()
83 ax1.bar([x+0.7 for x in [*range(len(data))]], sorted_time_y, width, color='orange',
84         ↪ label='time')
85 ax2.bar([x+1 for x in [*range(len(data))]], sorted_accuracy_y, width, color='green',
86         ↪ label='accuracy')
87 ax2.bar([x+1.3 for x in [*range(len(data))]], sorted_loss_y, width, color='red', label
88         ↪ ='loss')
89
90
91 ax1.set_xlabel('Model')
92 ax1.set_ylabel('Time(s)', color="orange")
93 ax2.set_ylabel('Average_accuracy/loss')
94
95 plt.xticks(np.arange(1,len(data)+1), sorted_x)
96 plt.legend()
97 plt.savefig(f'output/{plots_name}_bars_grouped.png', bbox_inches='tight')

```