

Sprawozdanie
Z przedmiotu: Uczenie maszynowe
Ćwiczenie 2

Wykonały:

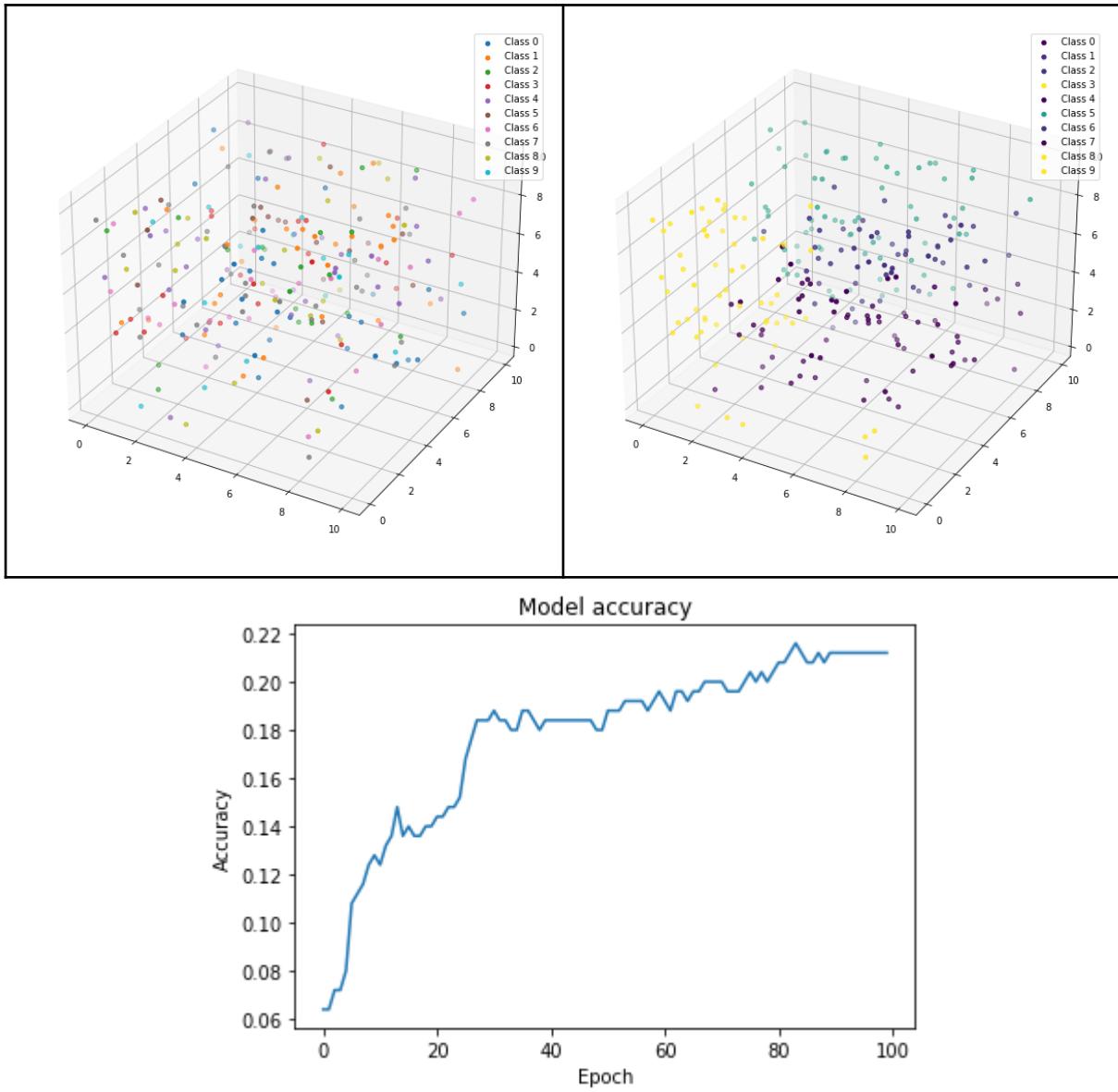
Anna Vezdenetska nr albumu: 107938

Aleksandra Kot nr albumu: 104061

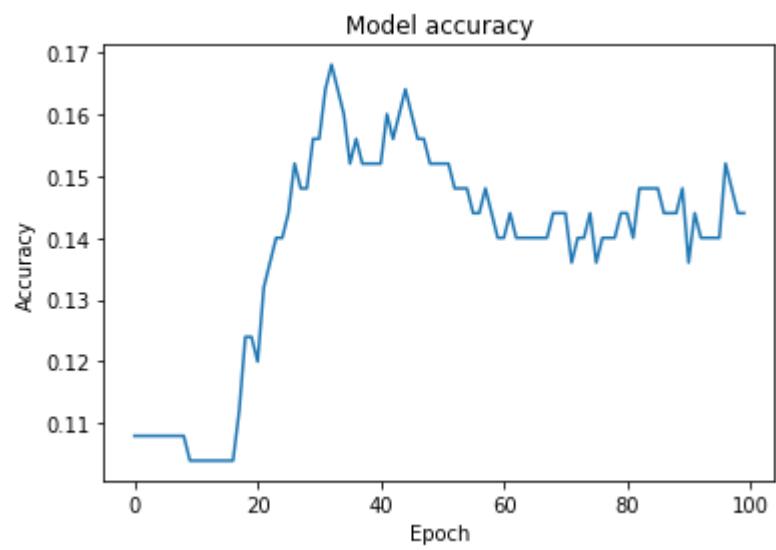
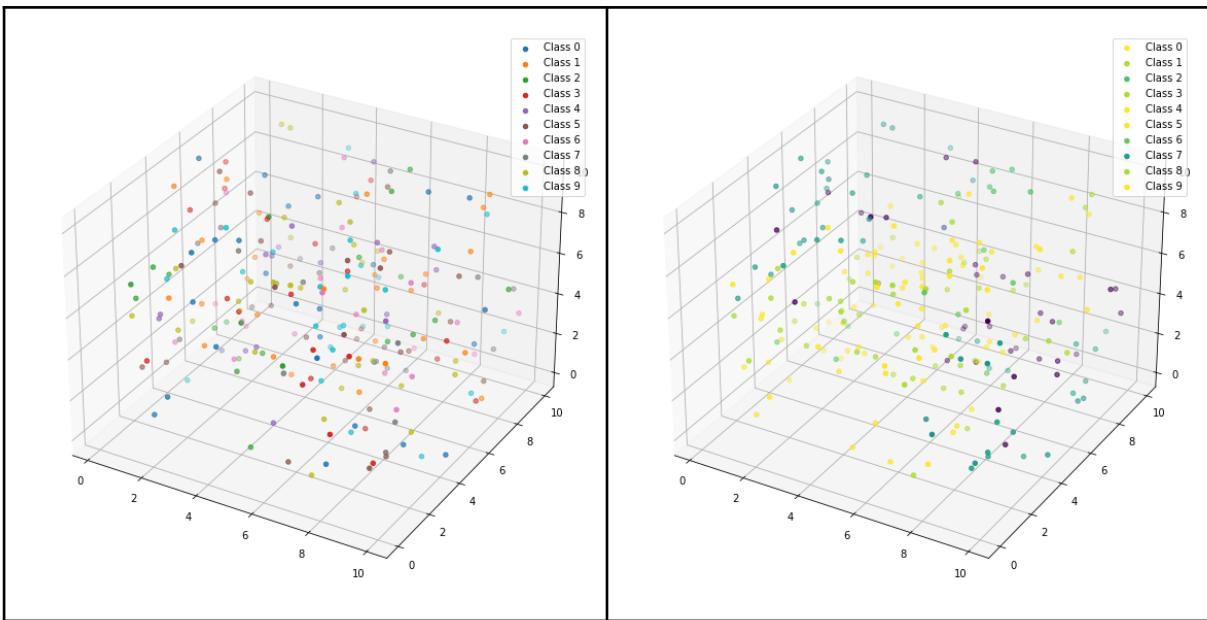
1. Losowy podział punktów

1. Przetestowano różne układy sieci:

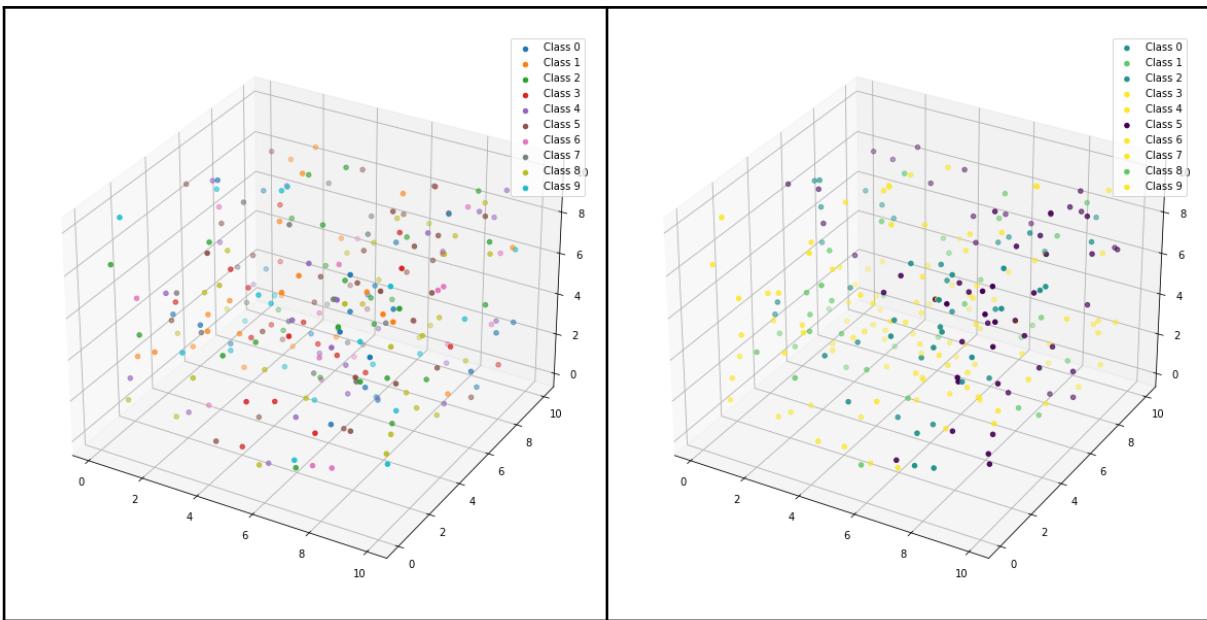
a) 2 warstwy



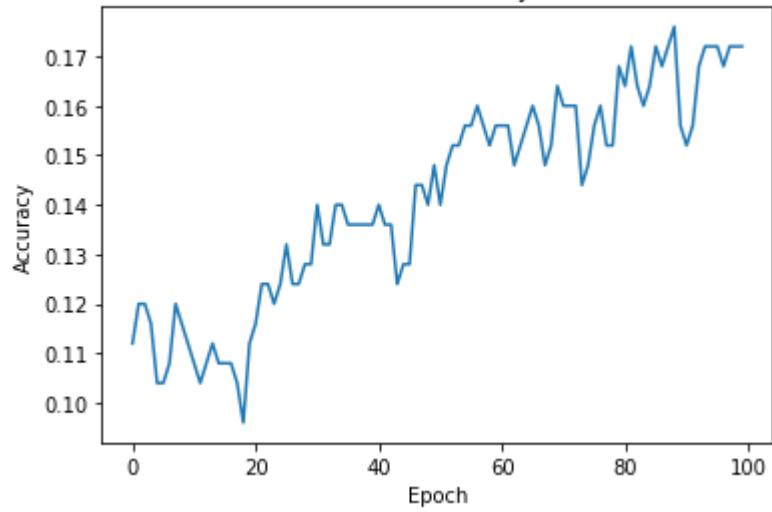
b) 3 warstwy



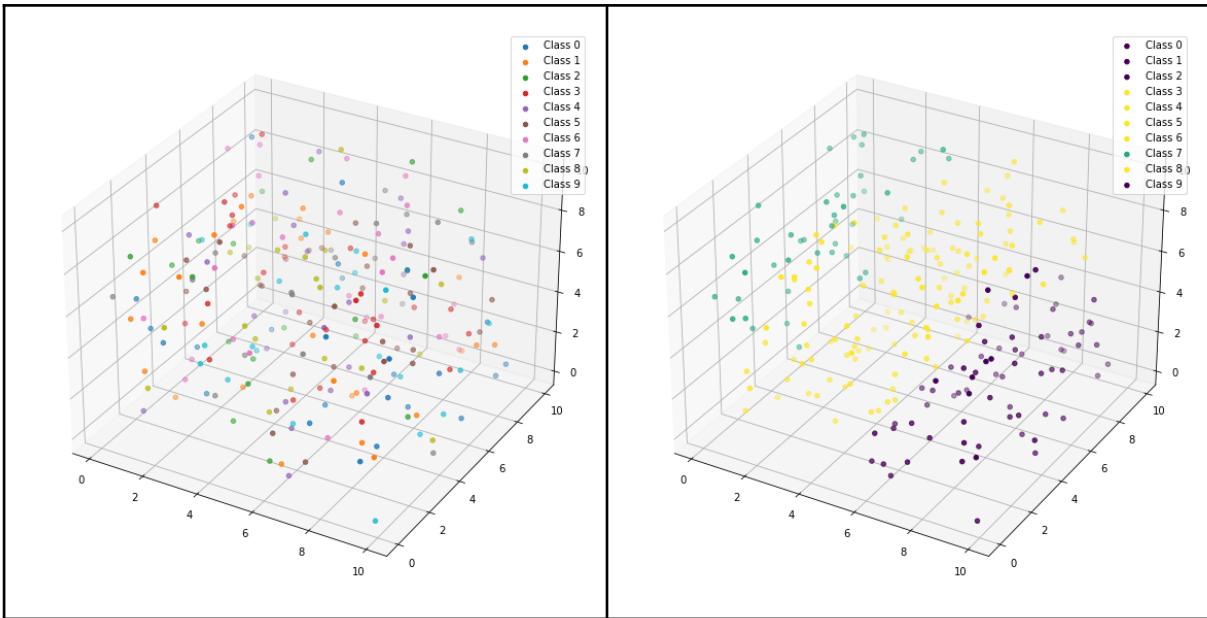
c) 4 warstwy



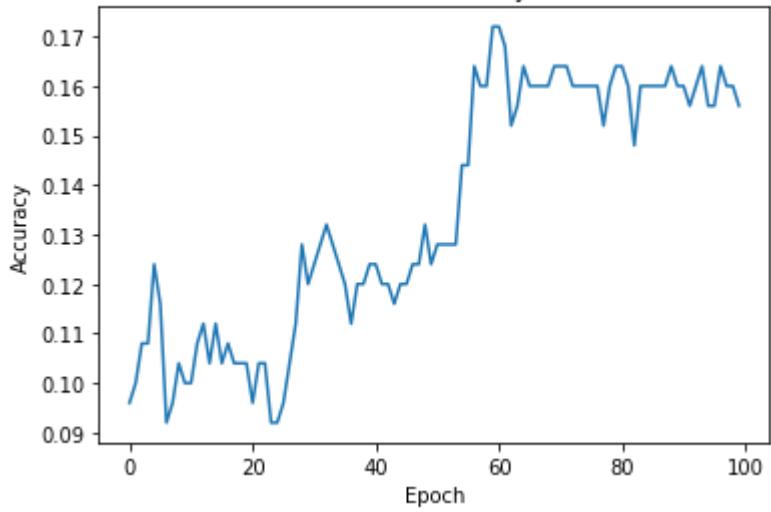
Model accuracy



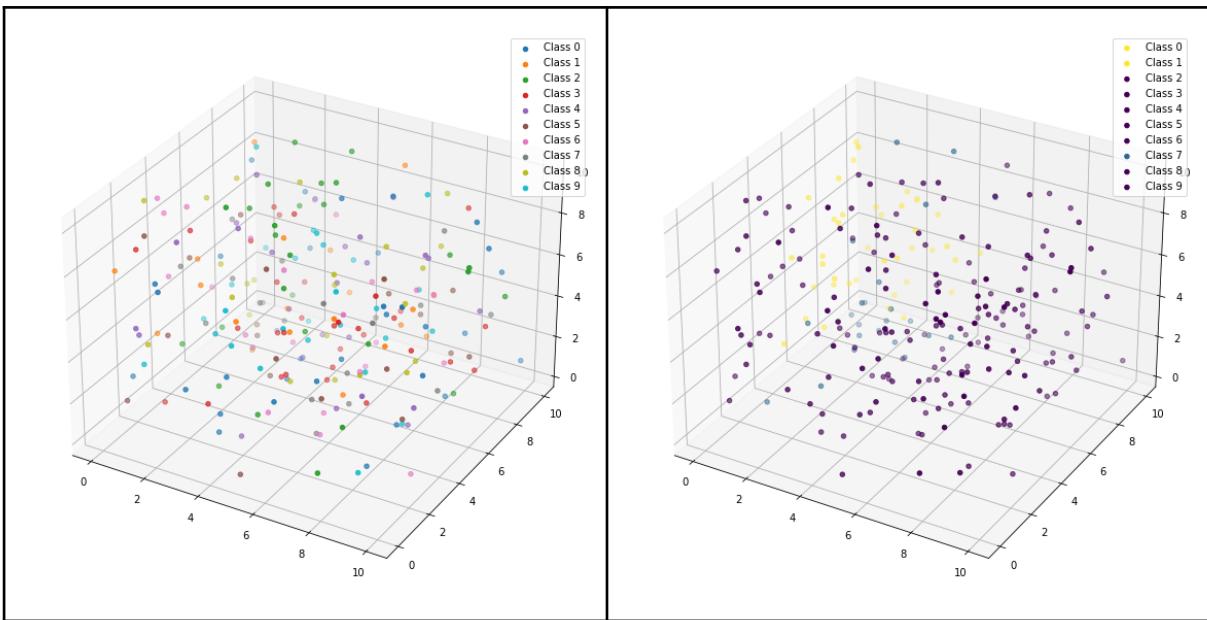
d) 5 warstw



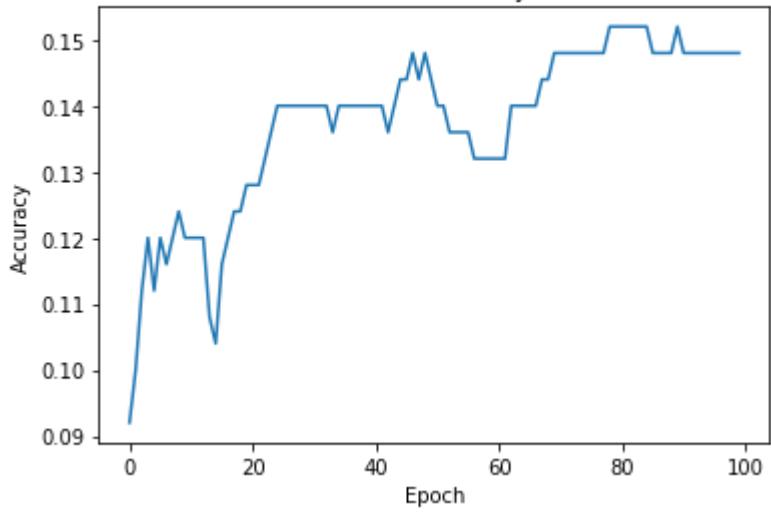
Model accuracy



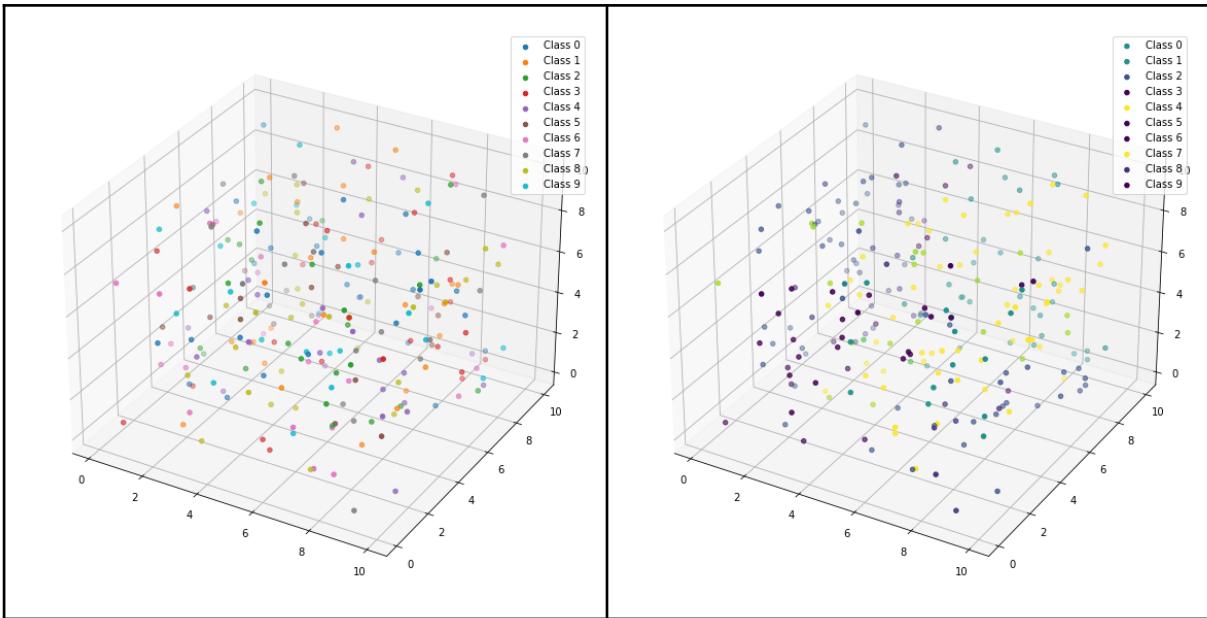
e) 6 warstw



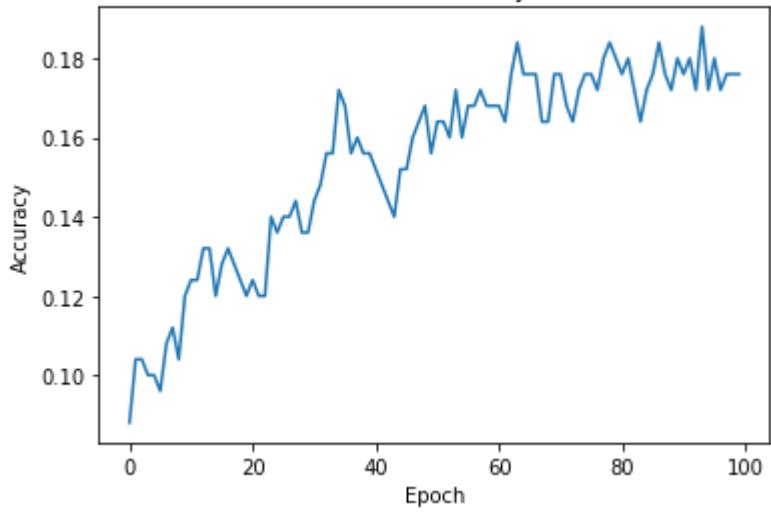
Model accuracy



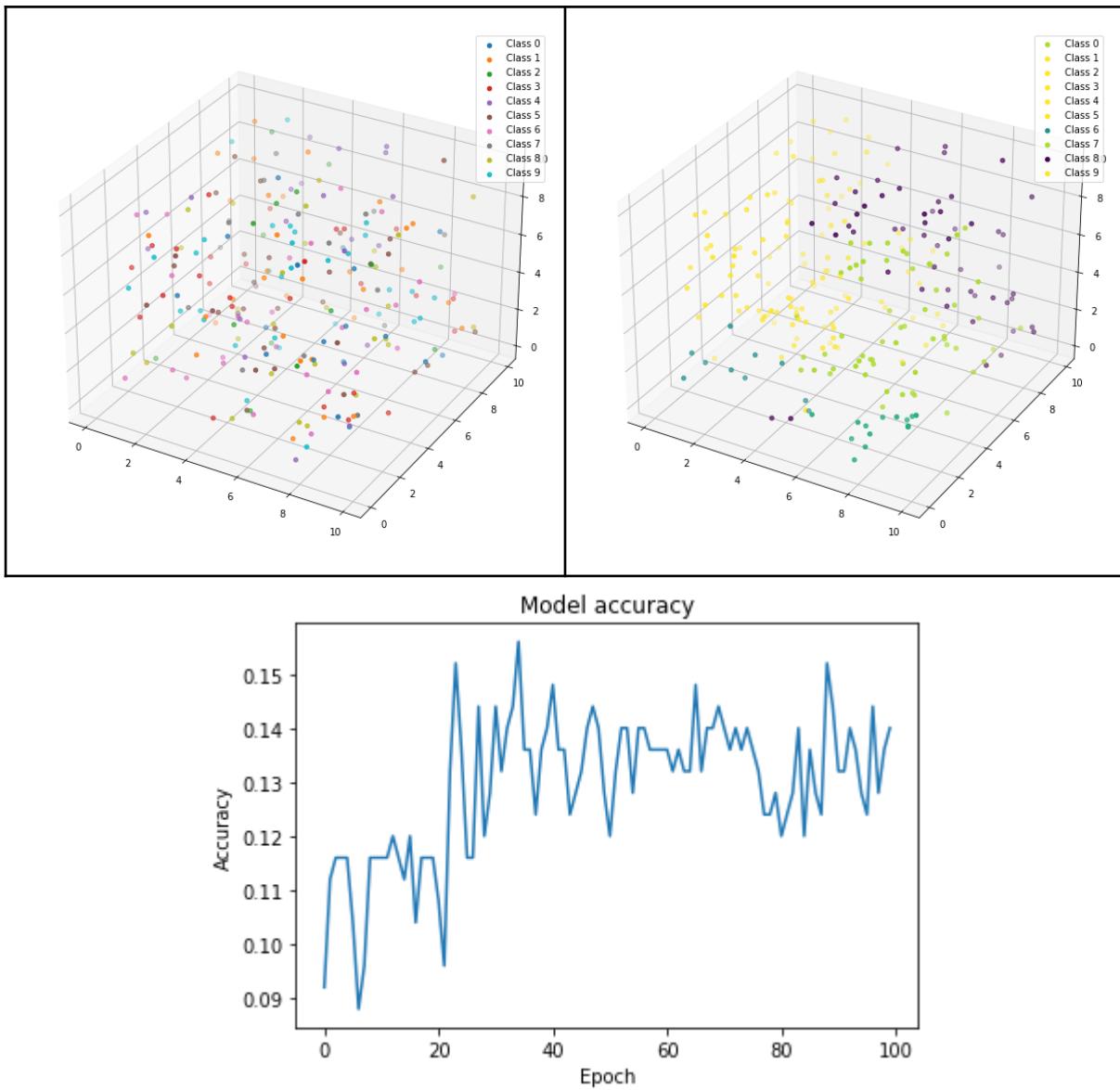
f) 7 warstw



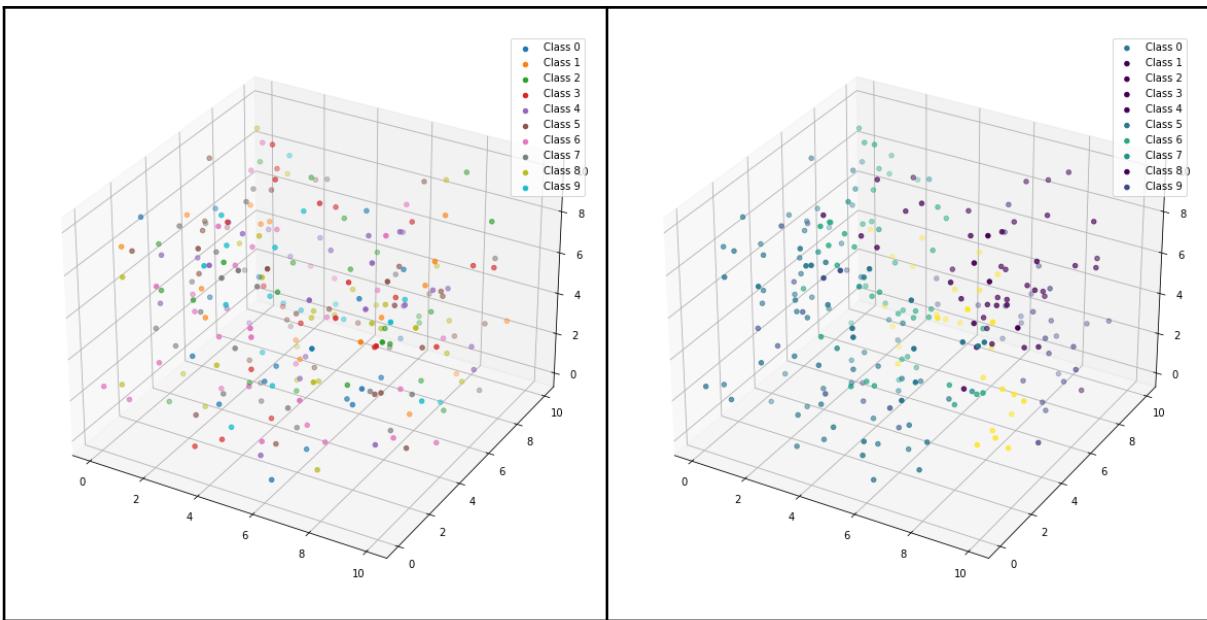
Model accuracy



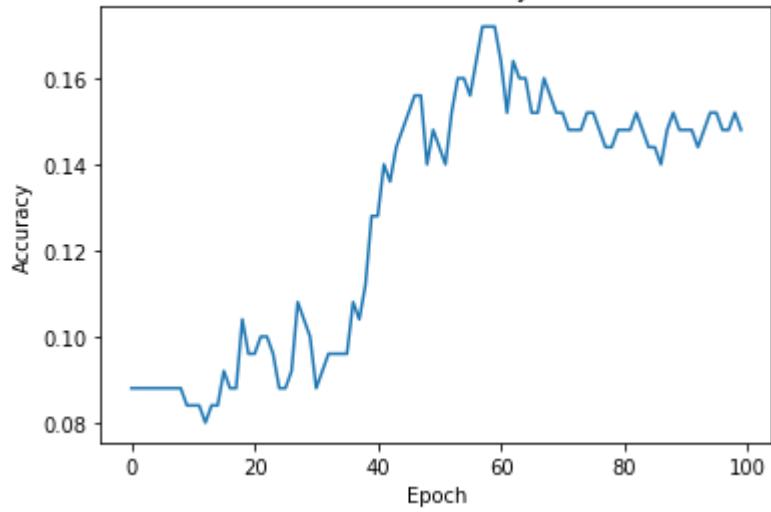
g) 8 warstw



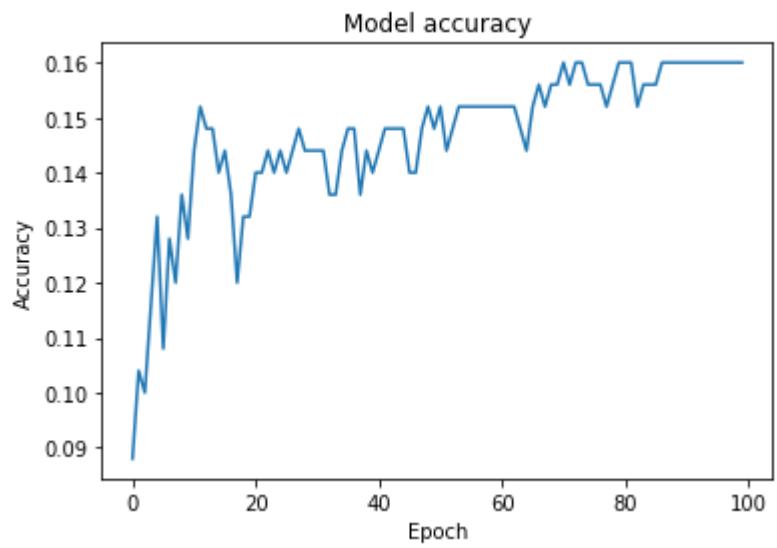
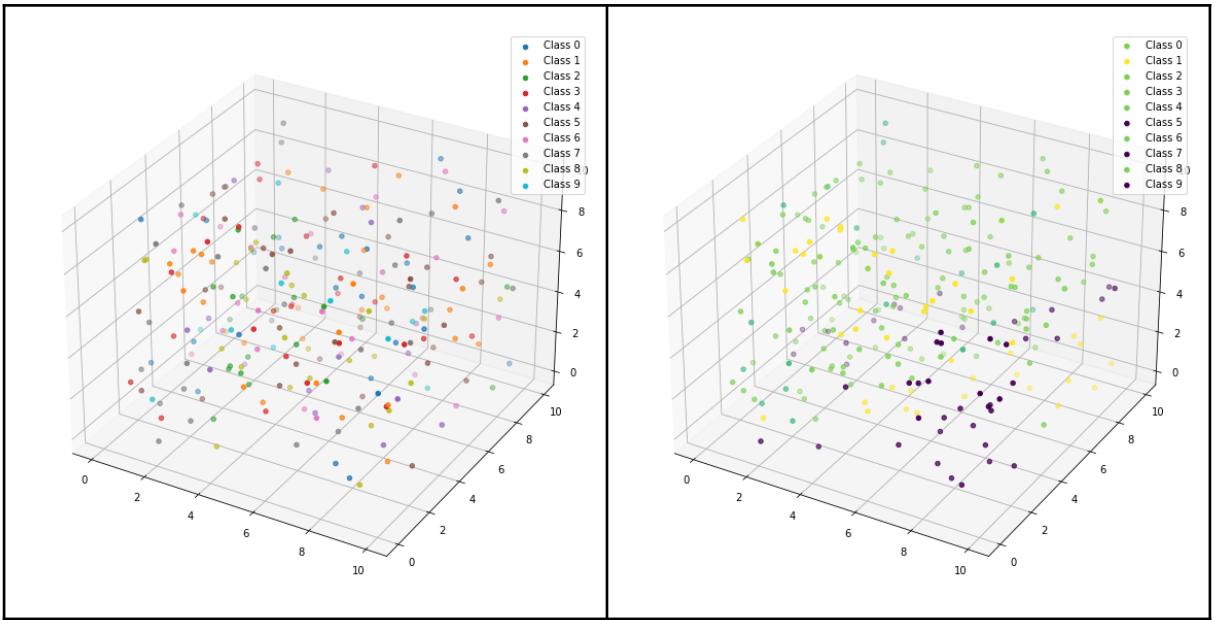
h) 9 warstw



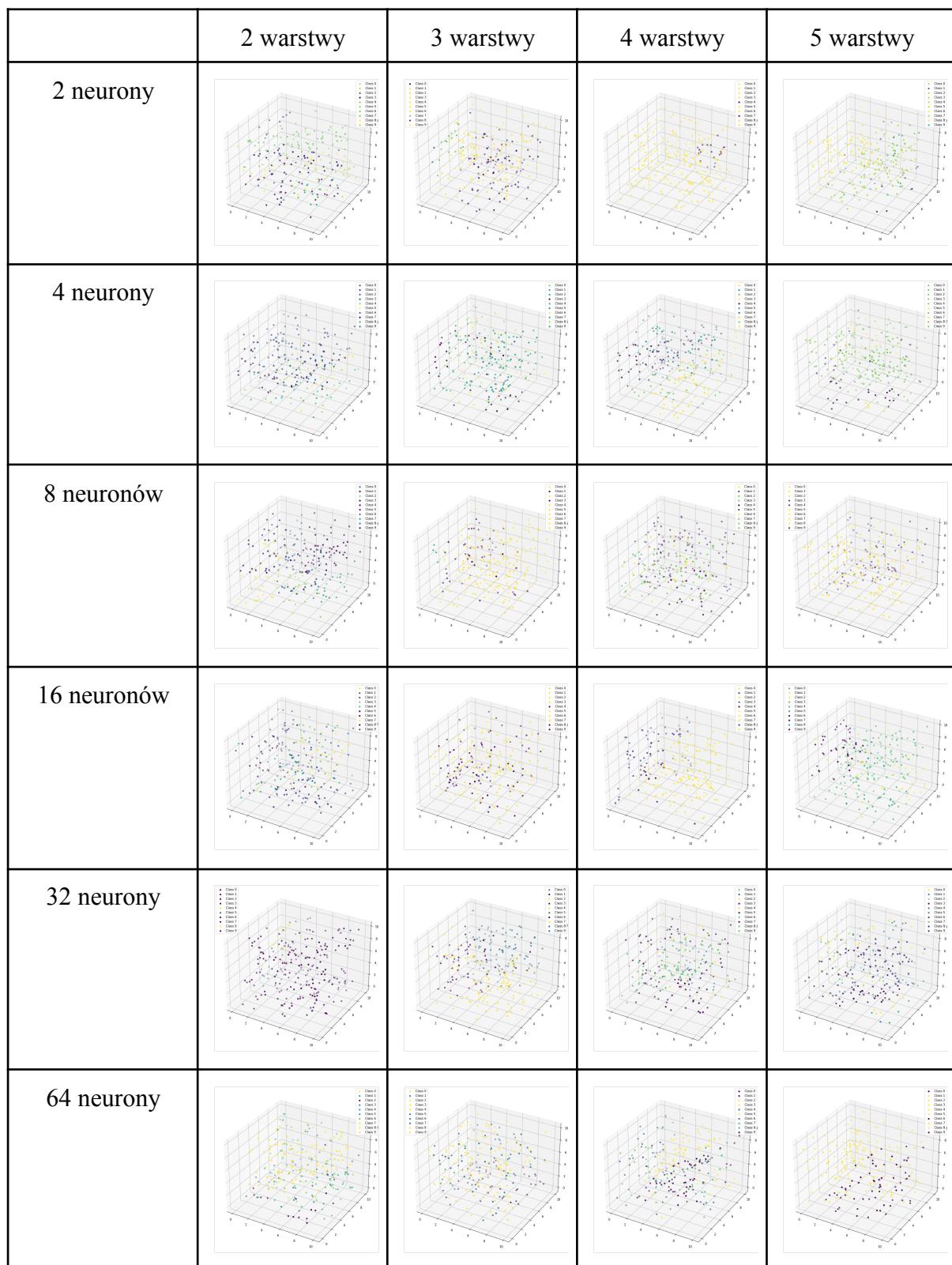
Model accuracy

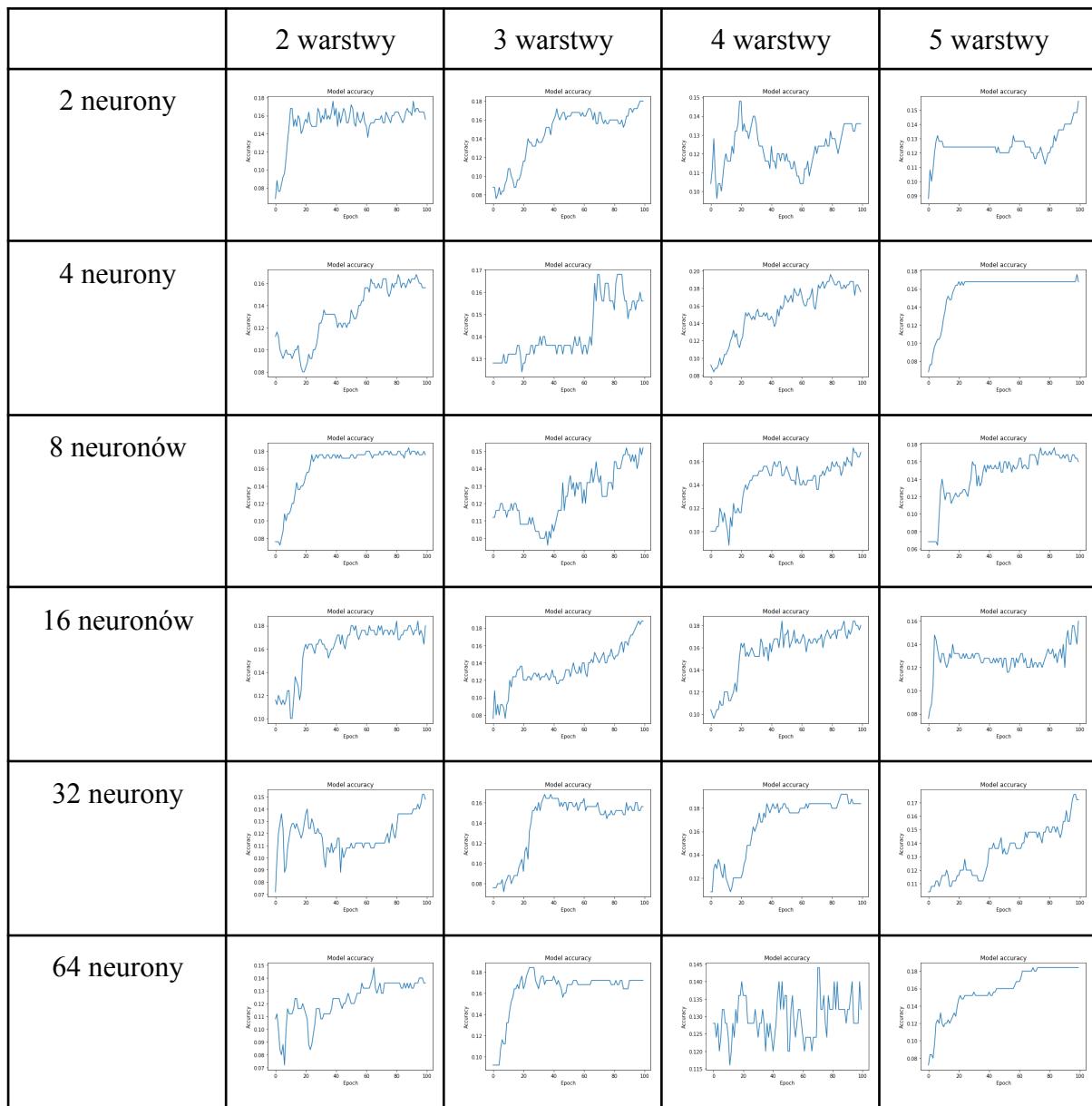


i) 10 warstw

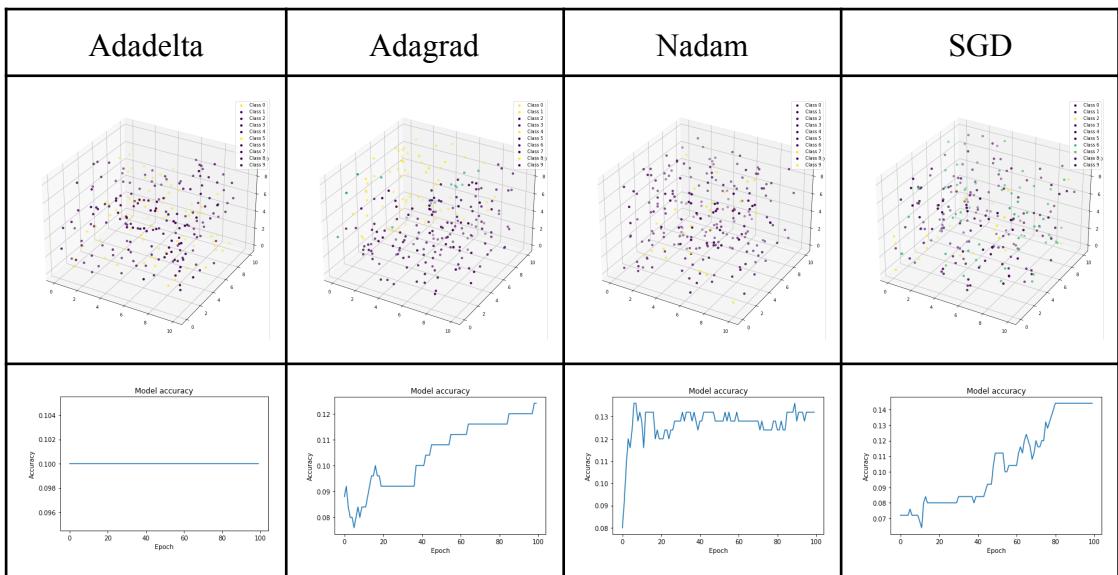


2. Dla każdego układu sieci przetestowano różną ilość neuronów. W poniższej tabeli przedstawione są wyniki



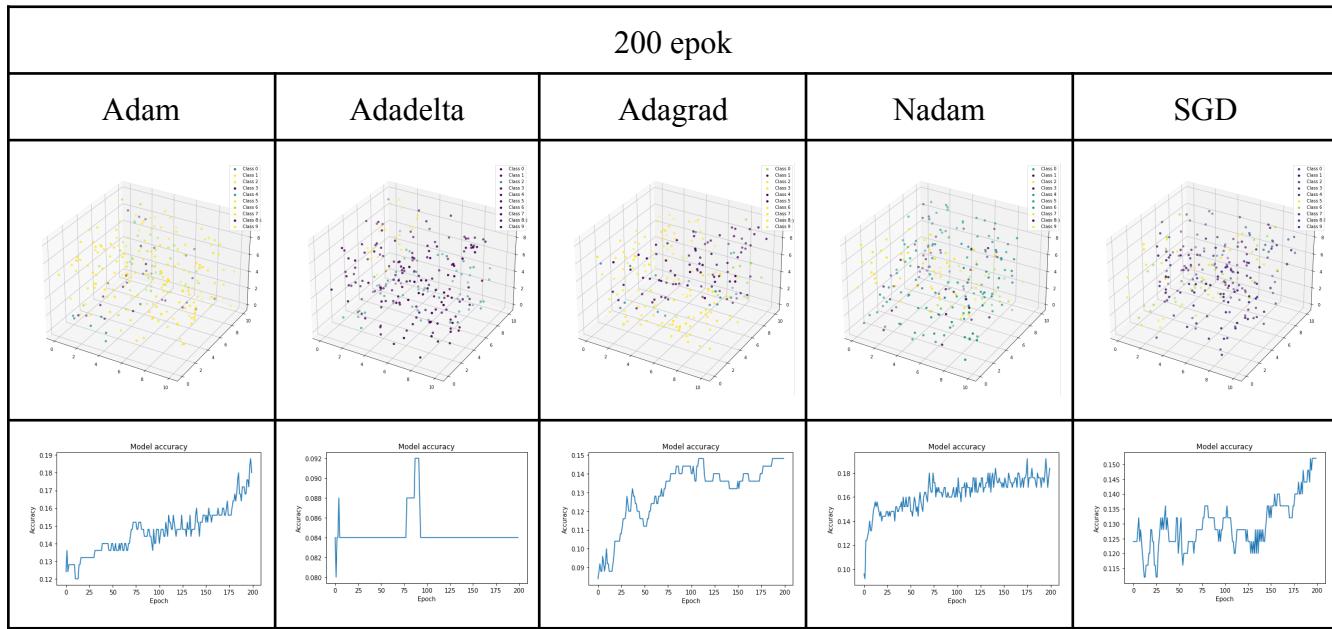


3. Testowanie optymalizatorów przetestowano na najlepszych wartościach, mianowicie ilość warstw 4 ilość neuronów 64

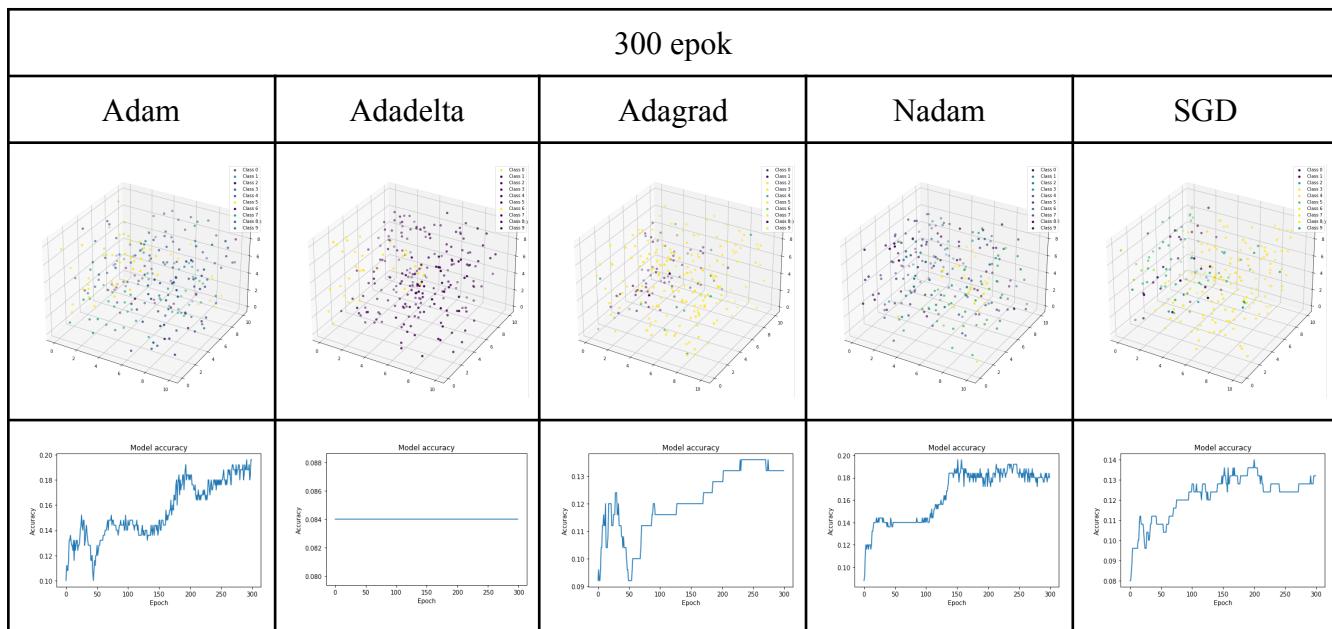


4. Sprawdzić, czy ilością trenowań można poprawić klasyfikację

200 epok

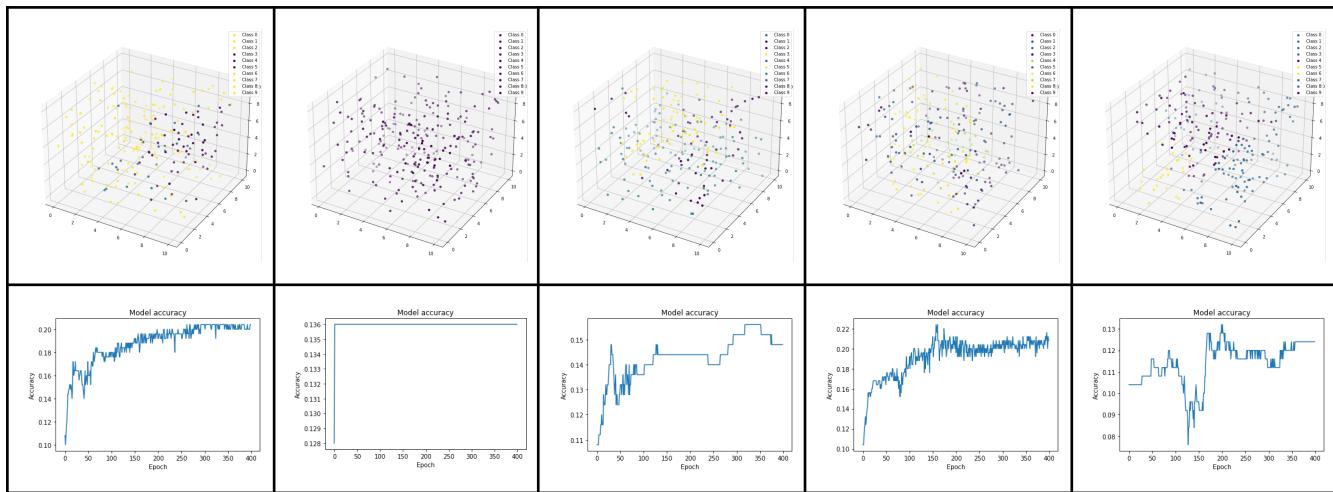


300 epok

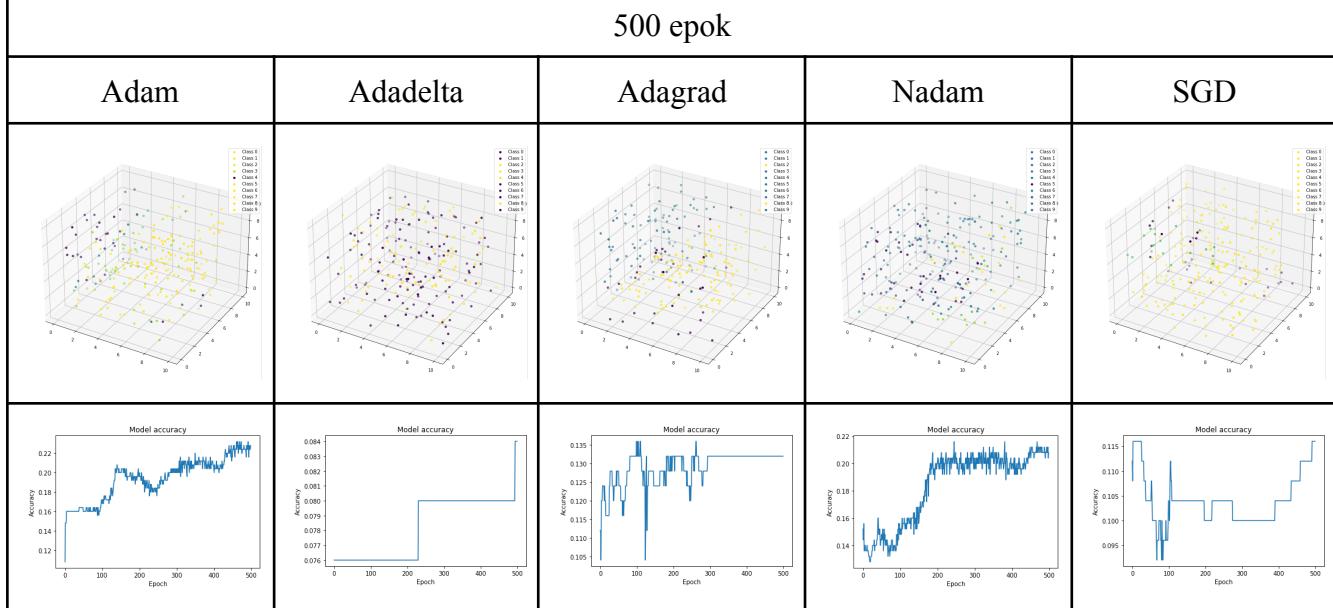


400 epok

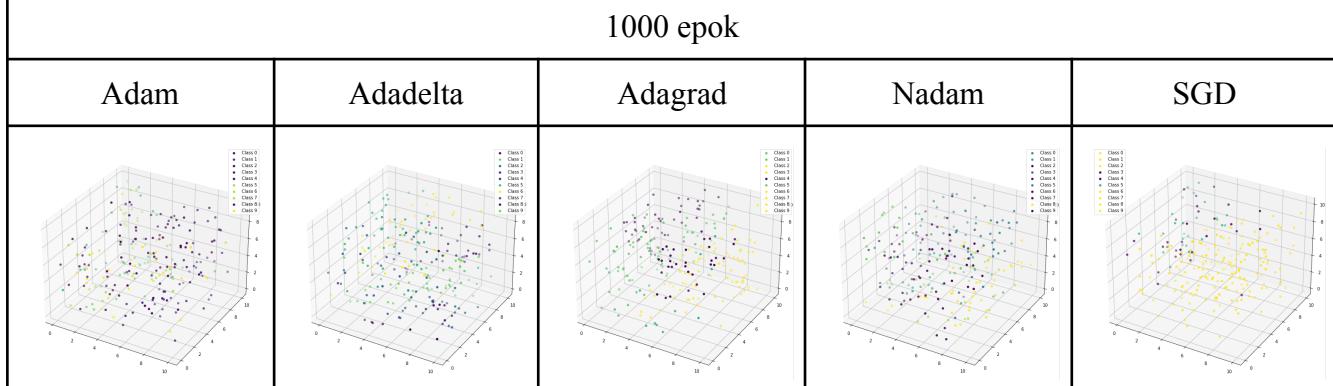
| Adam | Adadelta | Adagrad | Nadam | SGD |
|------|----------|---------|-------|-----|
|------|----------|---------|-------|-----|

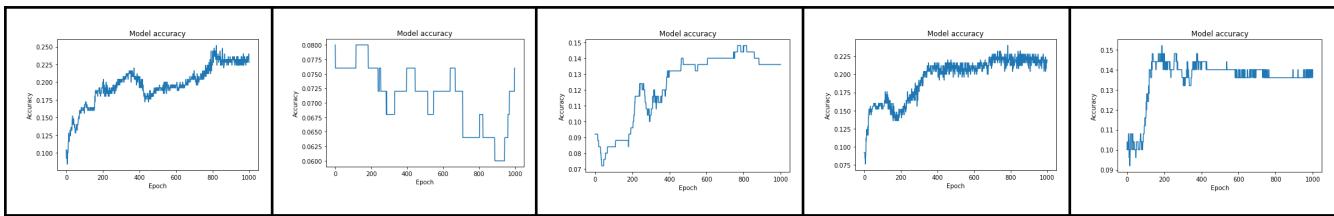


500 epok



1000 epok





Odpowiedzieć na pytania:

- Dlaczego sieć osiąga tak niską skuteczność trenowania na wskazanym zbiorze.

Powodem niskiej skuteczności wyników otrzymanych z powyższego zbioru, może wynikać ze złożoności wykorzystanego zbioru danych, a także z powodu architektury sieci. W kodzie użyto prostych, jednowarstwowych sieci neuronowych z ograniczoną liczbą neuronów na warstwę. Taka architektura może być niewystarczająca do nauczenia się złożonych wzorców w danych. Zbiór danych wygenerowany w tym kodzie zawiera różne klasy i kształty. Niektóre klasy są bardziej skomplikowane niż inne. Niektóre klasy są nieliniowe, podczas gdy inne są liniowe. Tego rodzaju złożoność może stanowić wyzwanie dla prostych modeli sieci neuronowych i może wymagać bardziej zaawansowanych technik lub bardziej złożonych architektur sieci.

- Czy jest coś co można zrobić aby klasyfikator poprawił jakość trenowania?

W celu poprawienia sugeruję zmianę optymalizatora, w kodzie użyto popularnego optymalizatora Adam. Jednak różne optymalizatory mogą działać lepiej lub gorzej w zależności od danych i problemu. Jak widać w powyższych wynikach, wykorzystując optymalizatory SGD oraz Nadam, udało się uzyskać, bardzo dobre wynikiem.

Skutek uzyskania lepszych wyników jest również zmiana architektury sieci, a dokładniej zastosowano 4 warstwy, gdzie były 64 neurony. Lecz warto byłoby rozbudować sieć o większą ilość warstw oraz eksperymentować z hiperparametrami typu learning rate lub spróbować przeprowadzić większą ilość trenowań.

Podział Gaussa

https://github.com/anksunamoona/Uczenie_Maszynowe_Cwiczenia_2/tree/main/Zbior_Gaussa

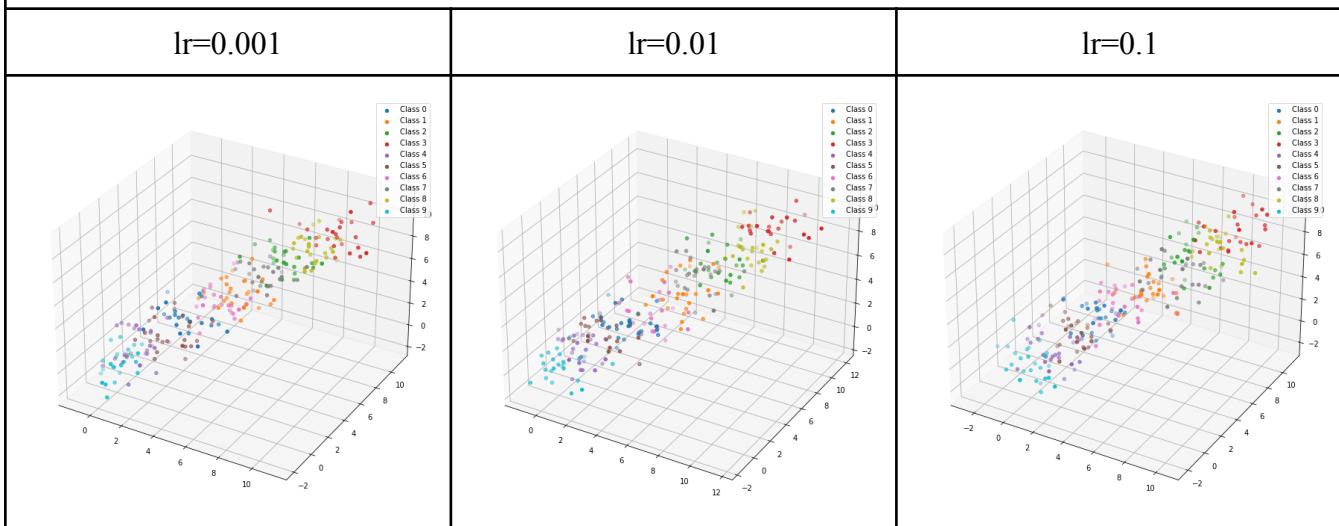
Architektura 1.

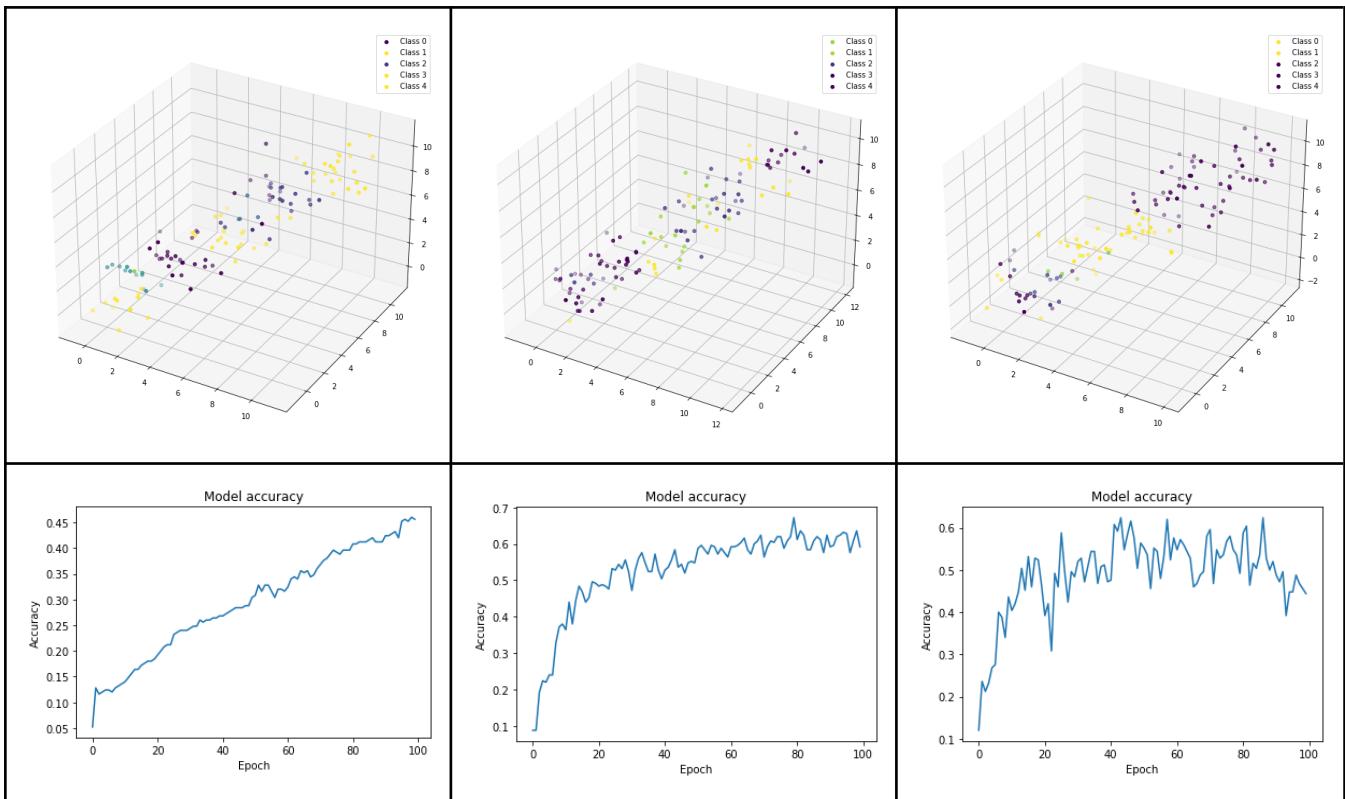
```
def experiment(num_layers=2, neurons_per_layer=16, activation='tanh',
epochs=100):
    X, y = generate_dataset()
    plot_dataset(X, y)

    classifier = Classifier(num_layers, neurons_per_layer, activation)
    model = classifier.build_model(input_shape=3)

    optimizer = Adam(lr=0.001)
    history = train(X, y, model, optimizer, epochs)
    plot_history(history)

    plot_classification_results(X, y, model)
```





ARCHITEKTURA 2

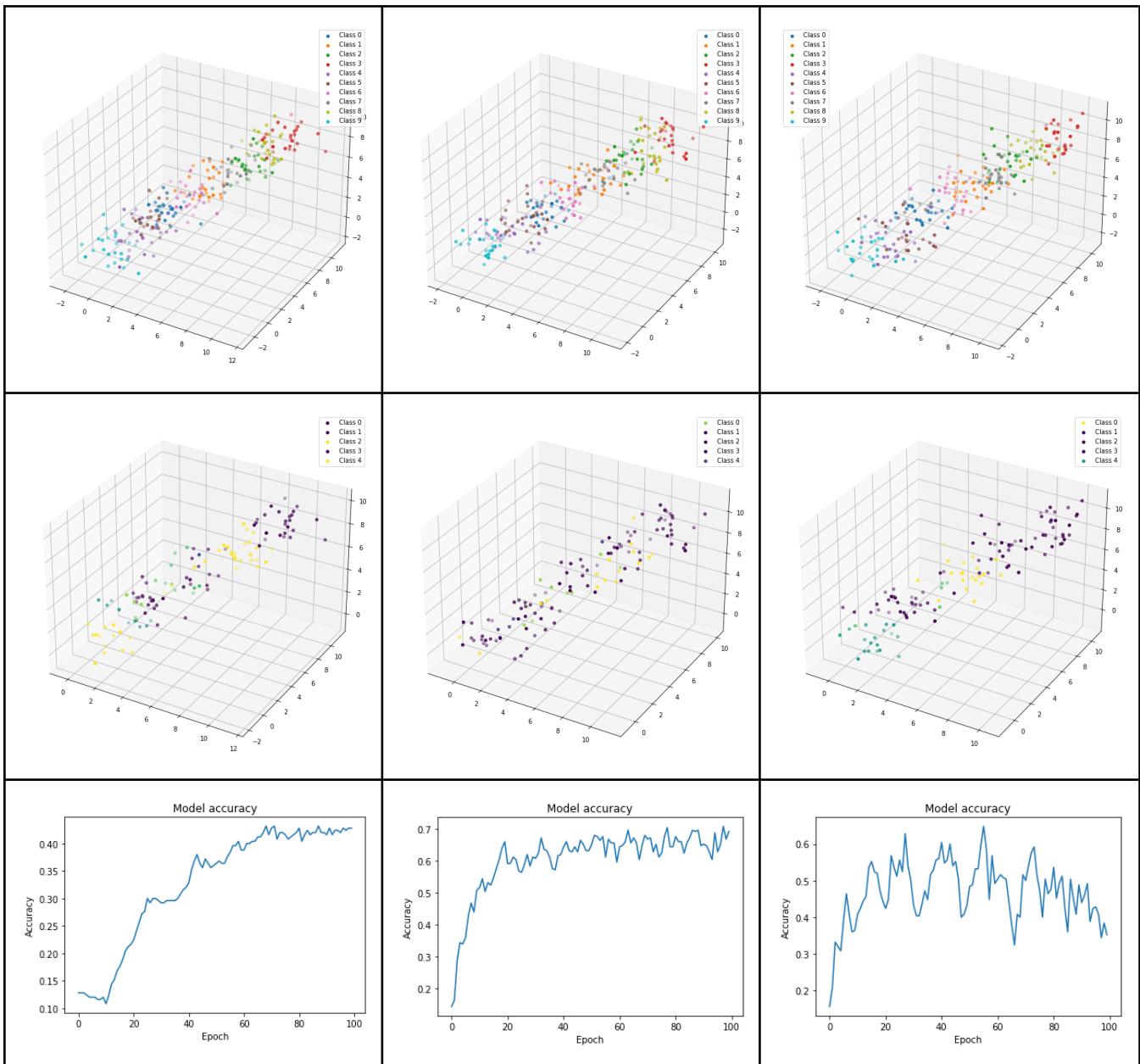
```
def experiment(num_layers=3, neurons_per_layer=32, activation='tanh',
epochs=100):
    X, y = generate_dataset()
    plot_dataset(X, y)

    classifier = Classifier(num_layers, neurons_per_layer, activation)
    model = classifier.build_model(input_shape=3)

    optimizer = Adam(lr=0.001)
    history = train(X, y, model, optimizer, epochs)
    plot_history(history)

    plot_classification_results(X, y, model)
```

| | | |
|----------|---------|--------|
| lr=0.001 | lr=0.01 | lr=0.1 |
|----------|---------|--------|



ARCHITEKTURA 3

```

def experiment(num_layers=8, neurons_per_layer=256, activation='sigmoid',
epochs=100):
    X, y = generate_dataset()
    plot_dataset(X, y)

    classifier = Classifier(num_layers, neurons_per_layer, activation)
    model = classifier.build_model(input_shape=3)

    optimizer = Adam(lr=0.001)
    history = train(X, y, model, optimizer, epochs)

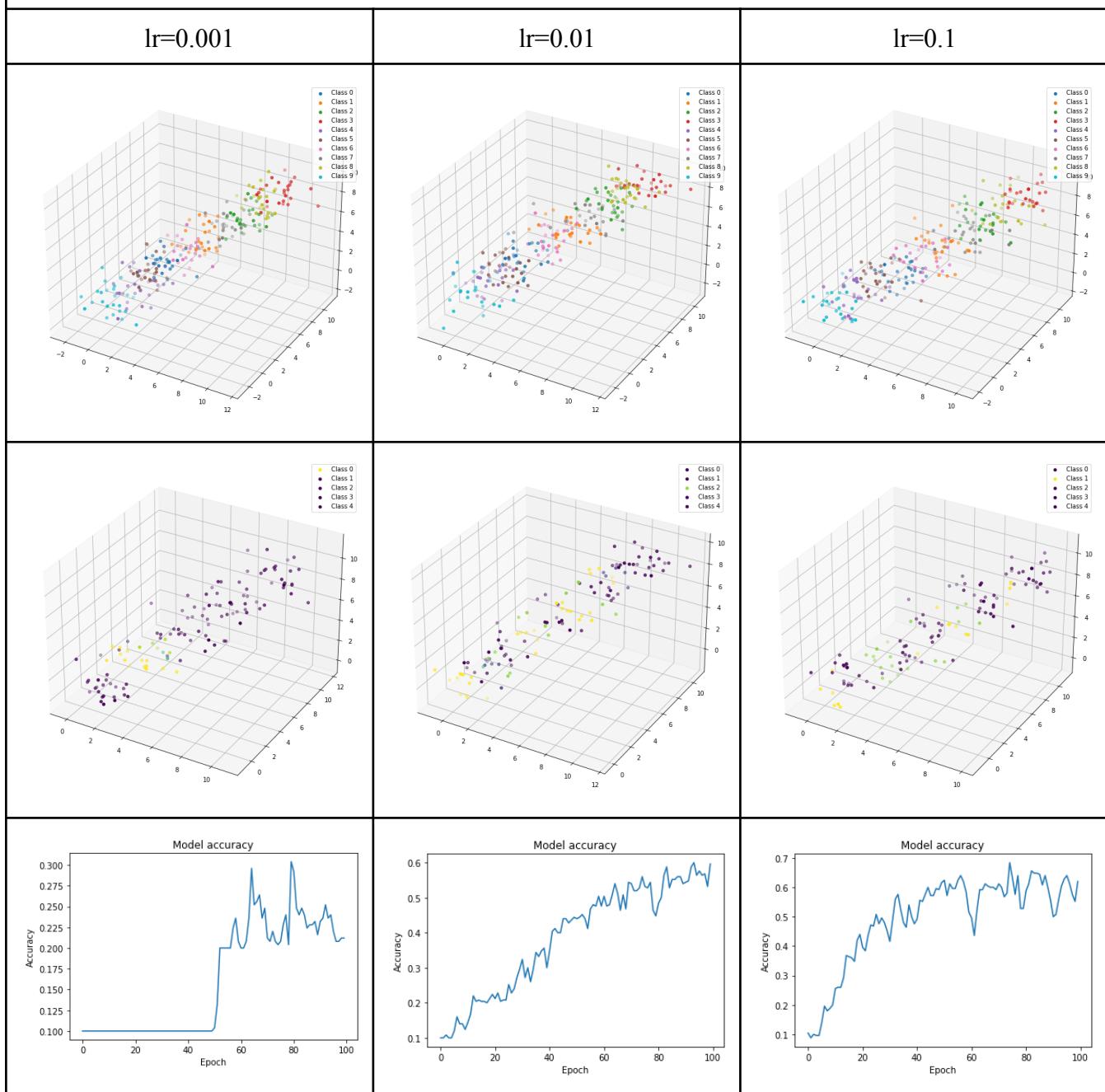
```

```

plot_history(history)

plot_classification_results(X, y, model)

```



ARCHITEKTURA 4

```

def experiment(num_layers=8, neurons_per_layer=256, activation='sigmoid',
epochs=100):
    X, y = generate_dataset()

```

```

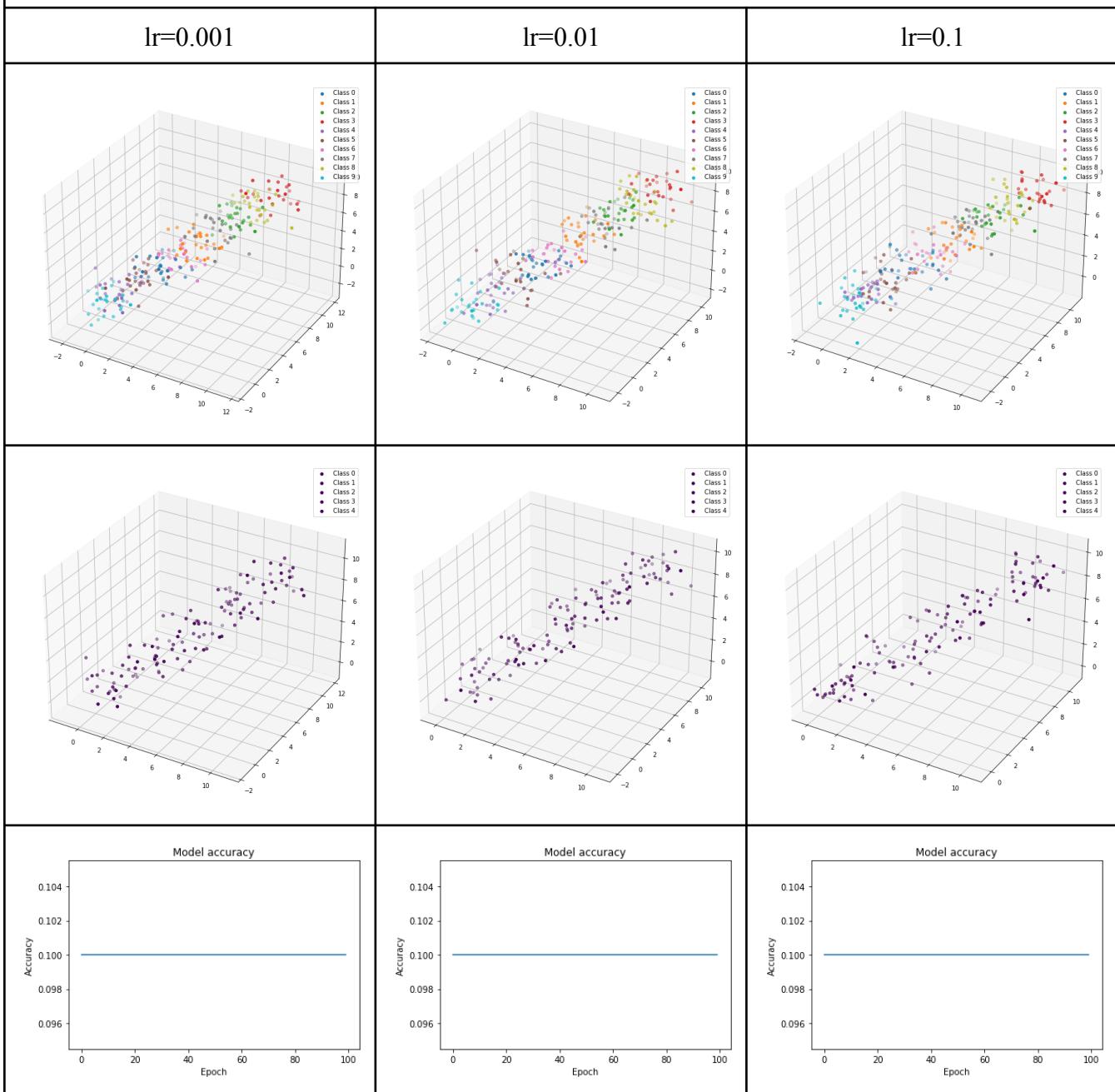
plot_dataset(X, y)

classifier = Classifier(num_layers, neurons_per_layer, activation)
model = classifier.build_model(input_shape=3)

optimizer = Adadelta(lr=0.001)
history = train(X, y, model, optimizer, epochs)
plot_history(history)

plot_classification_results(X, y, model)

```



ARCHITEKTURA 5

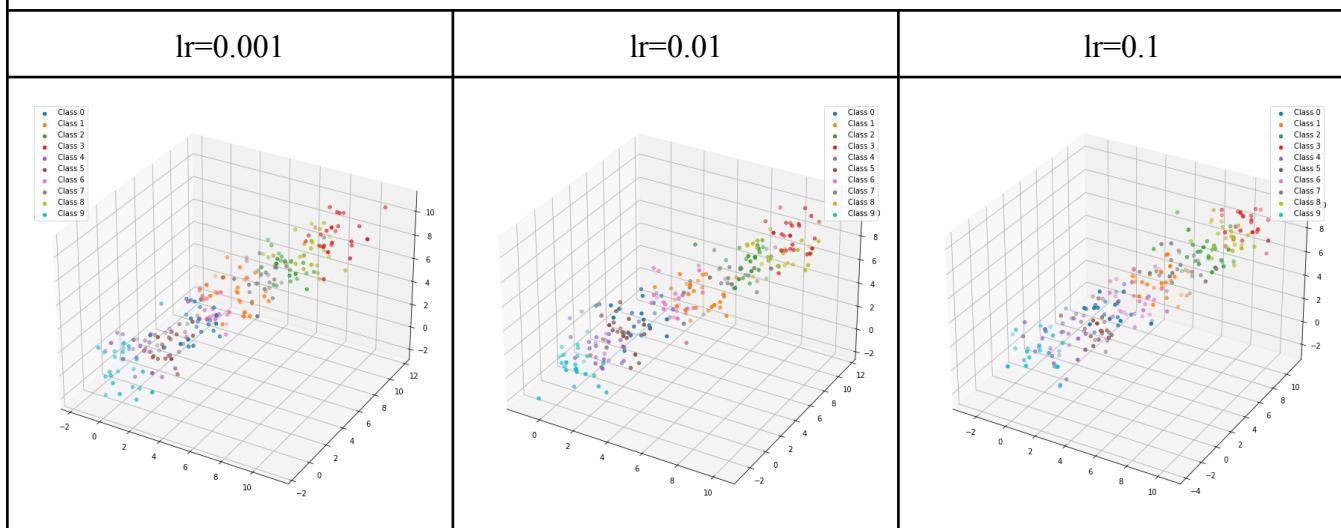
```
from keras.activations import relu

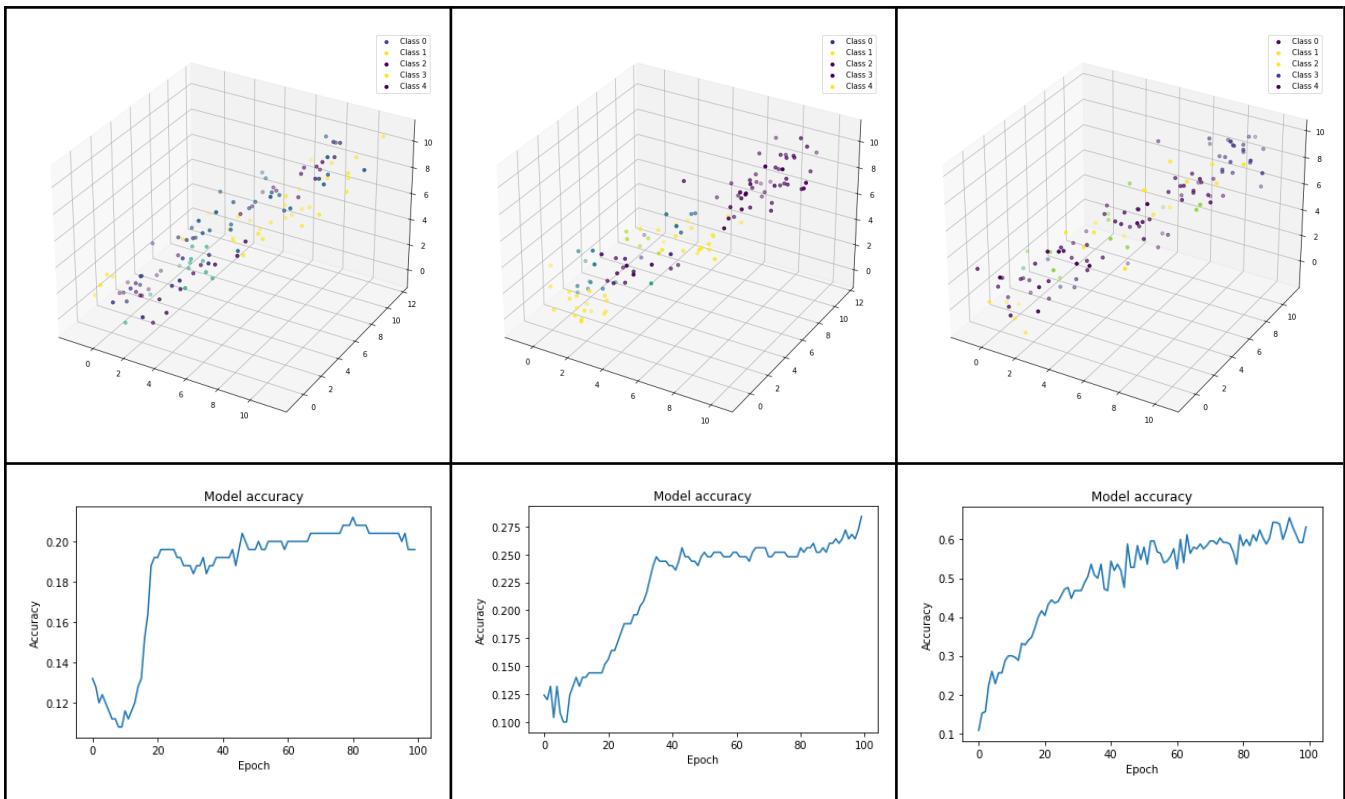
# Main function
def experiment(num_layers=8, neurons_per_layer=256, activation=relu,
epochs=100):
    X, y = generate_dataset()
    plot_dataset(X, y)

    classifier = Classifier(num_layers, neurons_per_layer, activation)
    model = classifier.build_model(input_shape=3)

    optimizer = Adagrad(lr=0.001)
    history = train(X, y, model, optimizer, epochs)
    plot_history(history)

    plot_classification_results(X, y, model)
```





ARCHITEKTURA 6

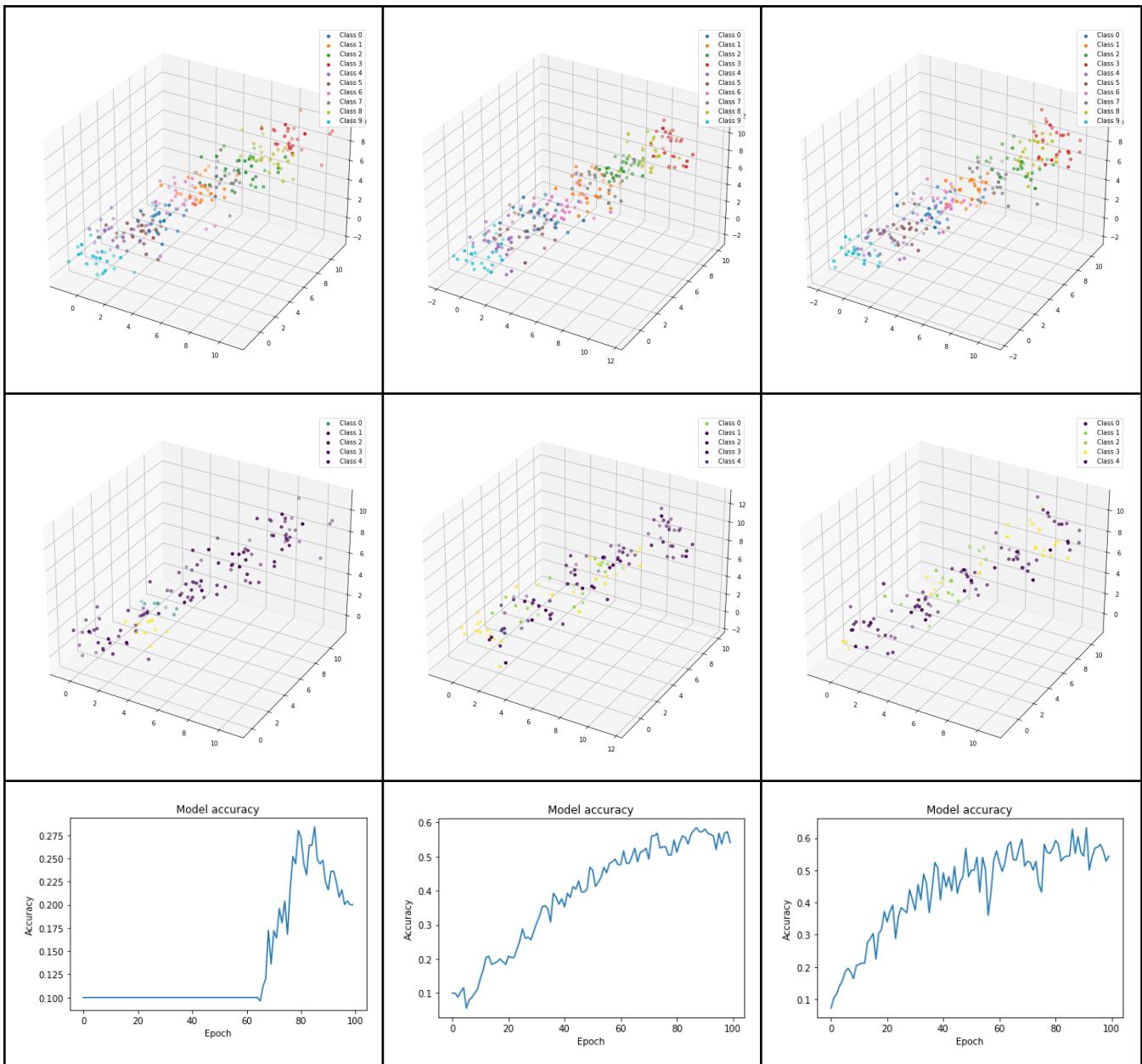
```
def experiment(num_layers=8, neurons_per_layer=256, activation='sigmoid',
epochs=100):
    X, y = generate_dataset()
    plot_dataset(X, y)

    classifier = Classifier(num_layers, neurons_per_layer, activation)
    model = classifier.build_model(input_shape=3)

    optimizer = Nadam(lr=0.001)
    history = train(X, y, model, optimizer, epochs)
    plot_history(history)

    plot_classification_results(X, y, model)
```

| | | |
|----------|---------|--------|
| lr=0.001 | lr=0.01 | lr=0.1 |
|----------|---------|--------|



ARCHITEKTURA 7

```

def experiment(num_layers=4, neurons_per_layer=32, activation='sigmoid',
epochs=100):
    X, y = generate_dataset()
    plot_dataset(X, y)

    classifier = Classifier(num_layers, neurons_per_layer, activation)
    model = classifier.build_model(input_shape=3)

    optimizer = Adagrad(lr=0.001)
    history = train(X, y, model, optimizer, epochs)

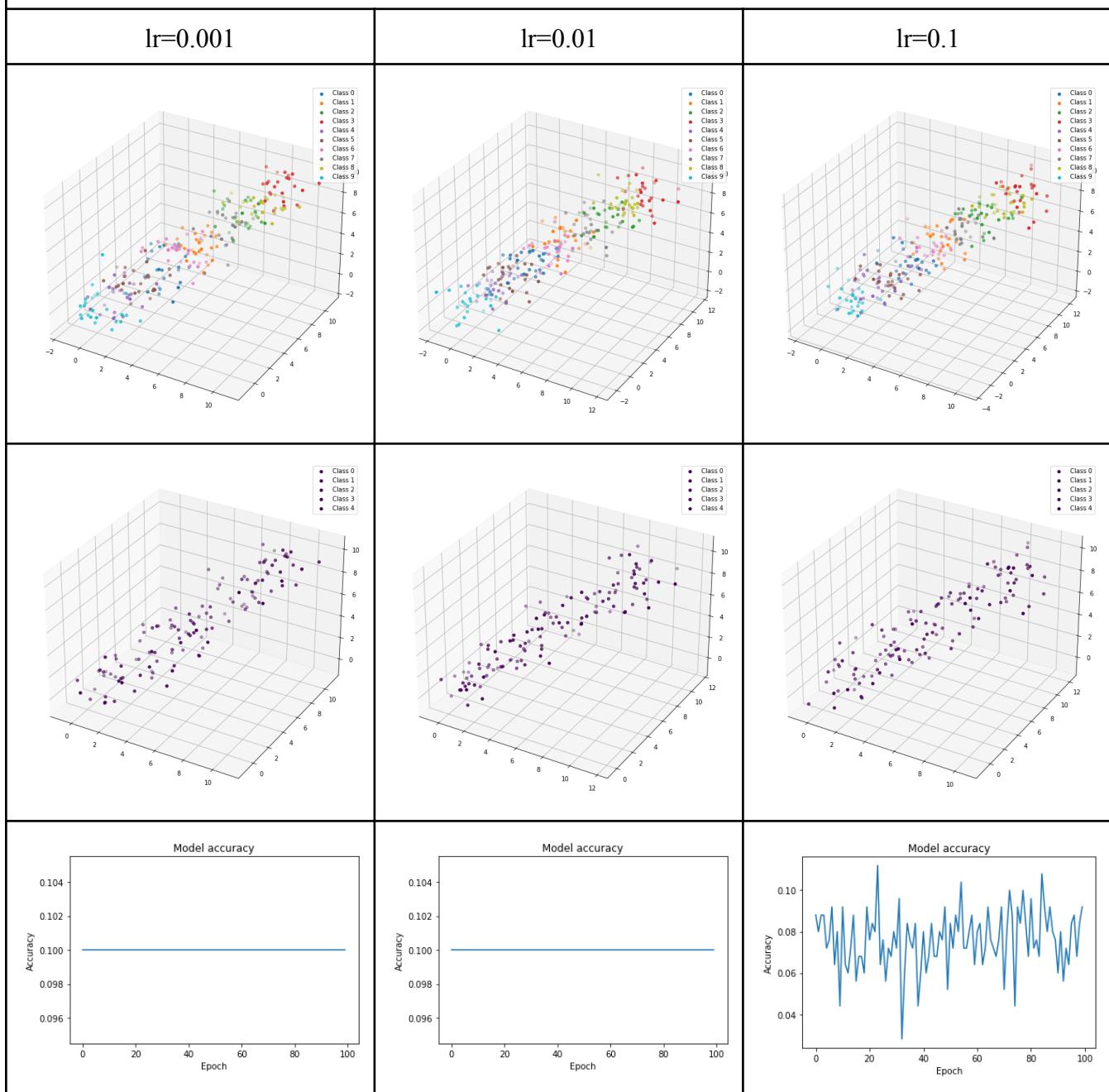
```

```

plot_history(history)

plot_classification_results(X, y, model)

```



ARCHITEKTURA 8

```

def experiment(num_layers=6, neurons_per_layer=128, activation='tanh',
epochs=100):
    X, y = generate_dataset()

```

```

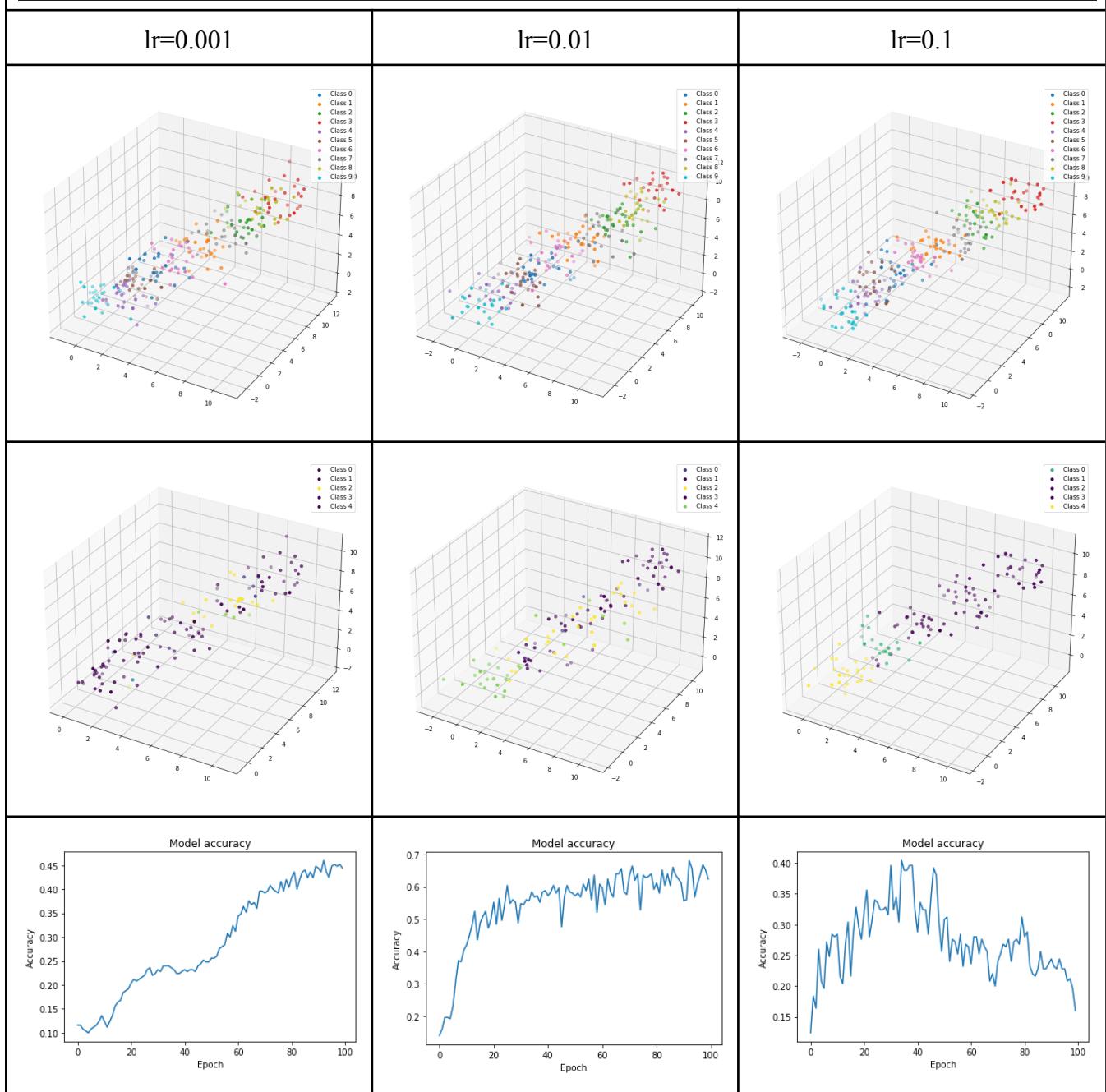
plot_dataset(X, y)

classifier = Classifier(num_layers, neurons_per_layer, activation)
model = classifier.build_model(input_shape=3)

optimizer = Nadam(lr=0.001)
history = train(X, y, model, optimizer, epochs)
plot_history(history)

plot_classification_results(X, y, model)

```



ARCHITEKTURA 9

```

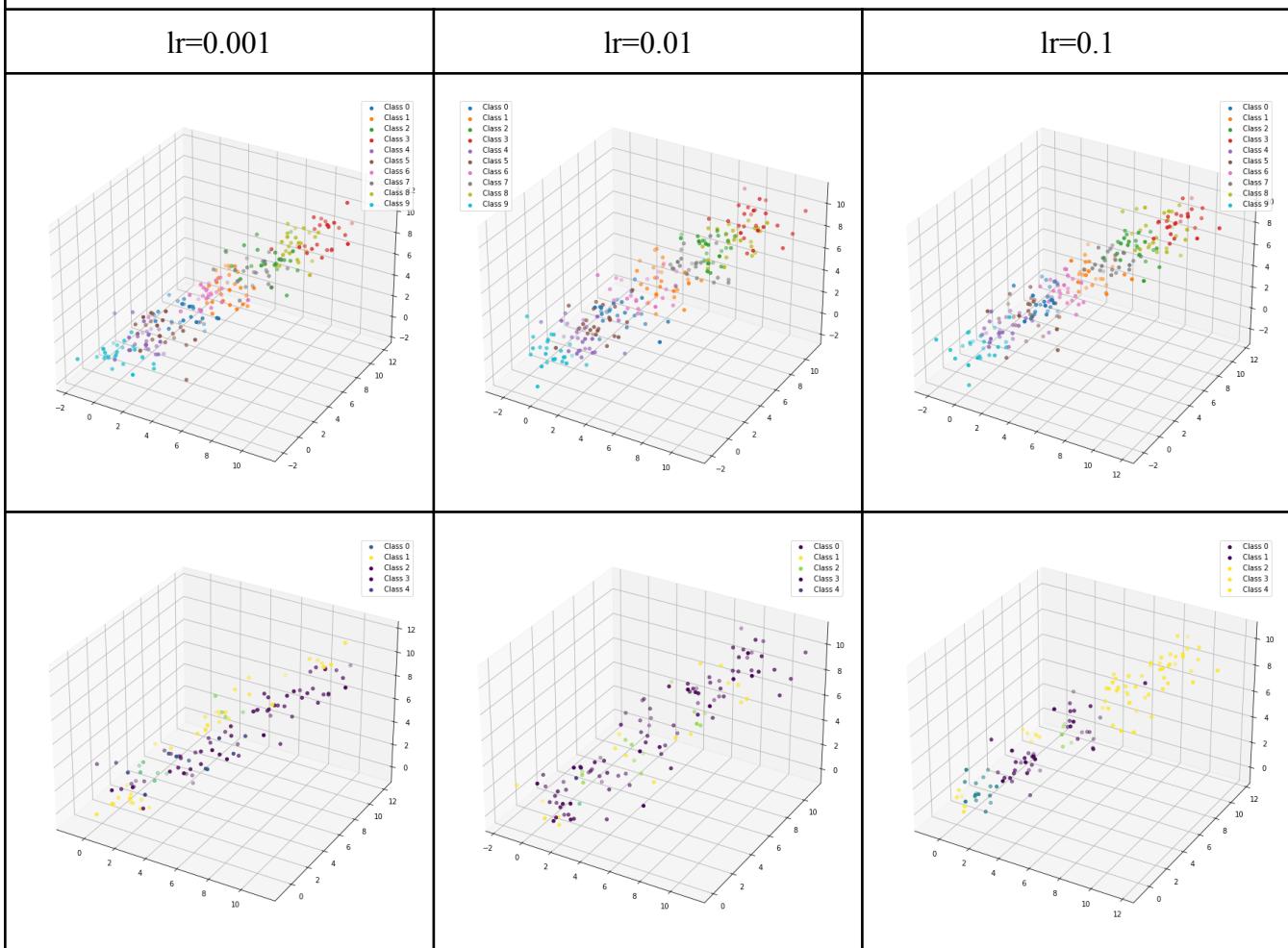
def experiment(num_layers=8, neurons_per_layer=512, activation='LeakyReLU',
epochs=100):
    X, y = generate_dataset()
    plot_dataset(X, y)

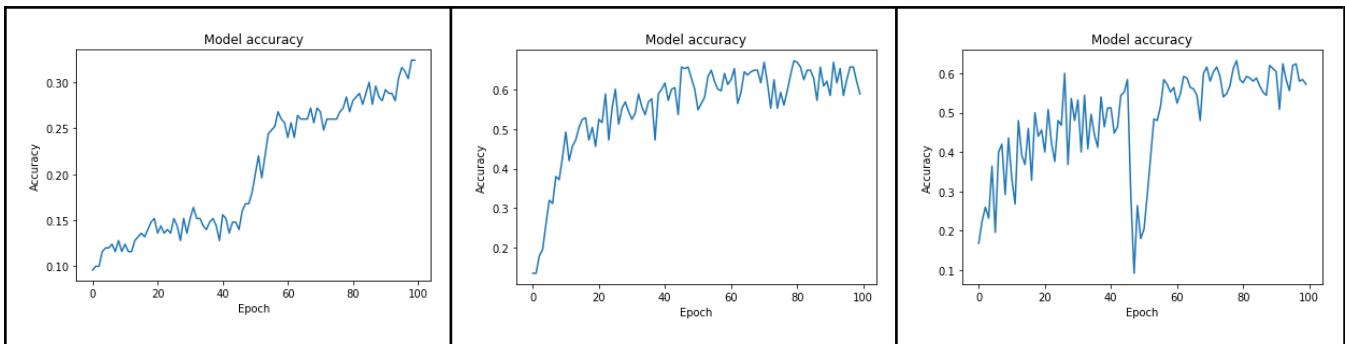
    classifier = Classifier(num_layers, neurons_per_layer, activation)
    model = classifier.build_model(input_shape=3)

    optimizer = Nadam(lr=0.001)
    history = train(X, y, model, optimizer, epochs)
    plot_history(history)

    plot_classification_results(X, y, model)

```





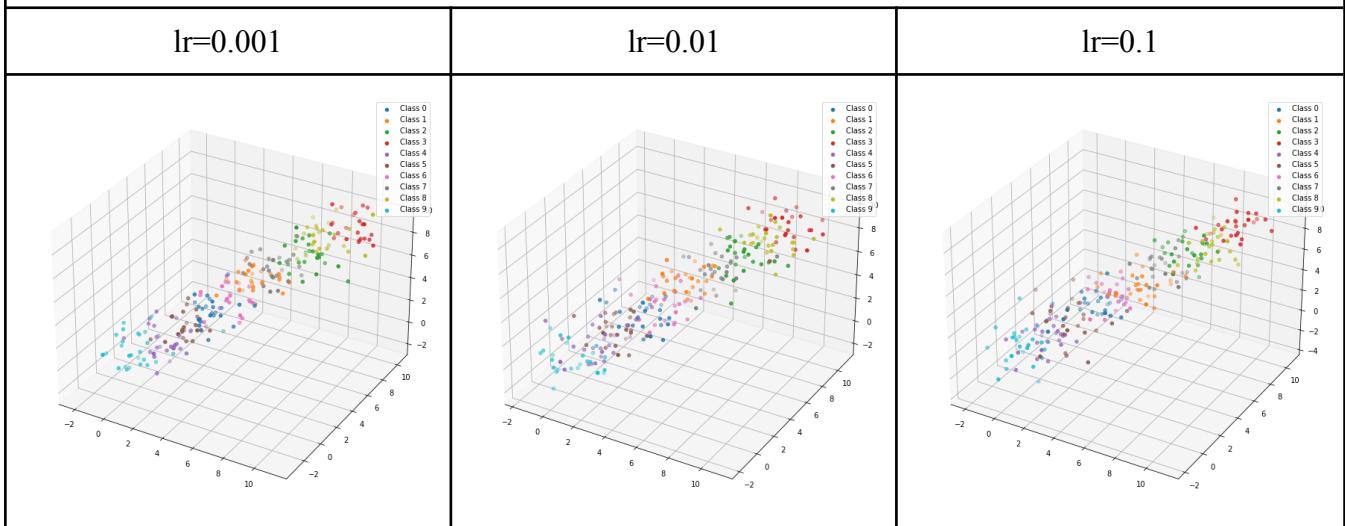
ARCHITEKTURA 10

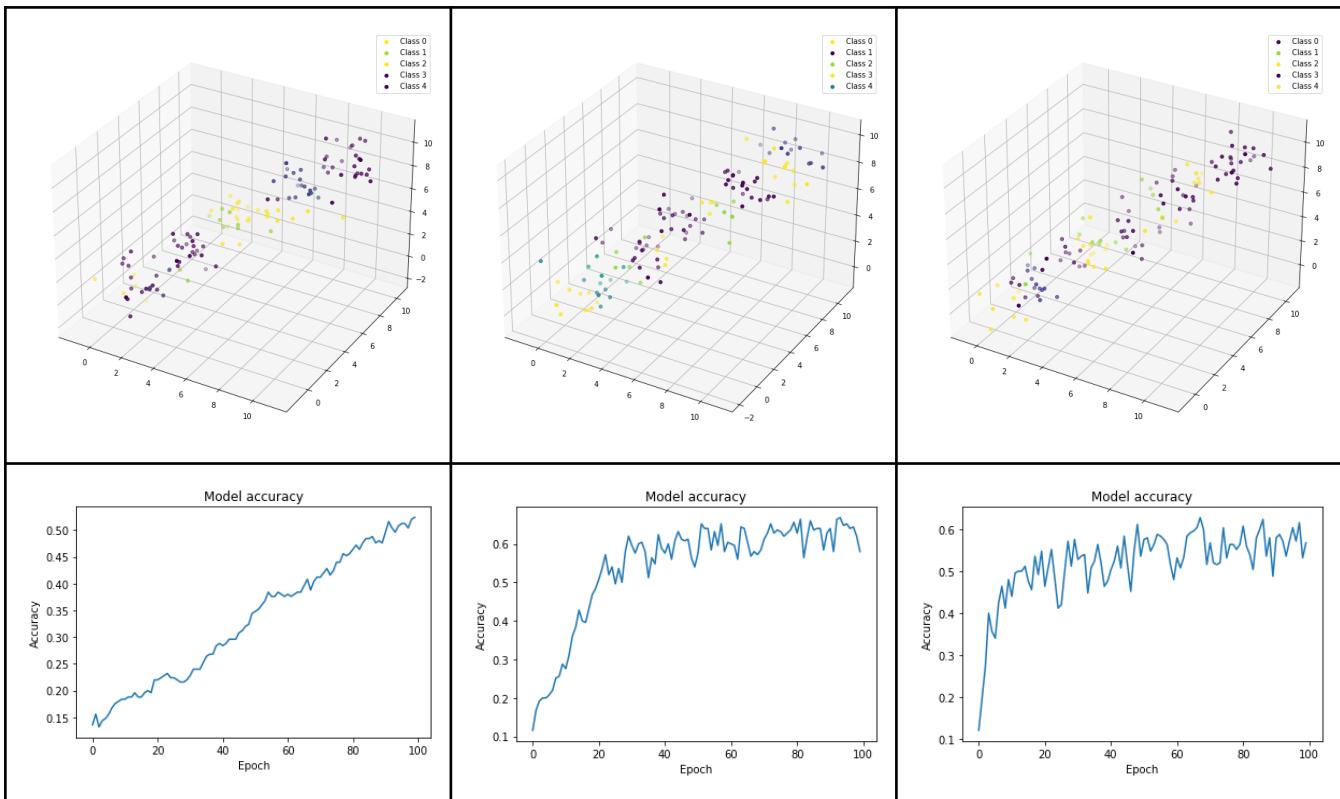
```
def experiment(num_layers=8, neurons_per_layer=256, activation='LeakyReLU',
epochs=100):
    X, y = generate_dataset()
    plot_dataset(X, y)

    classifier = Classifier(num_layers, neurons_per_layer, activation)
    model = classifier.build_model(input_shape=3)

    optimizer = Adam(lr=0.001)
    history = train(X, y, model, optimizer, epochs)
    plot_history(history)

    plot_classification_results(X, y, model)
```





Odpowiedzieć na pytania:

1. Dlaczego wyniki trenowania dają lepsze rezultaty niż w poprzednim zbiorze. Uzasadnić odpowiedź.

Polepszenie otrzymanych wyników, jest spowodowane uproszczeniem nauki dla sieci neuronowej, gdyż w danym zbiorze danych klasy są bardziej oddzielone i mają wyraźnie różne wartości średnie. Każda z klas ma też różne macierze kowariancji, co oznacza, że mają różne rozkłady w przestrzeni trójwymiarowej. To umożliwia sieci neuronowej lepsze rozróżnienie i klasyfikację elementów.

2. Czy architektura sieci ma znaczny wpływ na jakość klasyfikacji.

Tak, gdyż to architektura sieci neuronowej określa liczbę warstw, liczbę neuronów w warstwach, rodzaj funkcji aktywacji, sposób połączeń między warstwami itp. Wszystkie te elementy wpływają na zdolność sieci do reprezentacji złożonych zależności w danych i nauki odpowiednich cech dla poprawnej klasyfikacji.

3. Czy szybkość trenowania ma wpływ na jakość klasyfikacji.

Tak, gdyż zbyt niska szybkość trenowania może spowodować, że sieć będzie uczyć się wolno i może mieć problemy z dostosowaniem się do złożonych zależności w danych.

4. Czy rodzaj optymalizatora ma wpływ na jakość klasyfikacji.

Tak, gdyż optymalizator jest odpowiedzialny za aktualizację wag sieci neuronowej w trakcie procesu uczenia.

Inne zbiory

Zbiór geometryczny nr 3

https://github.com/anksunamoona/Uczenie_Maszynowe_Cwiczenie_2/tree/main/Zbior_Geometryczny_3

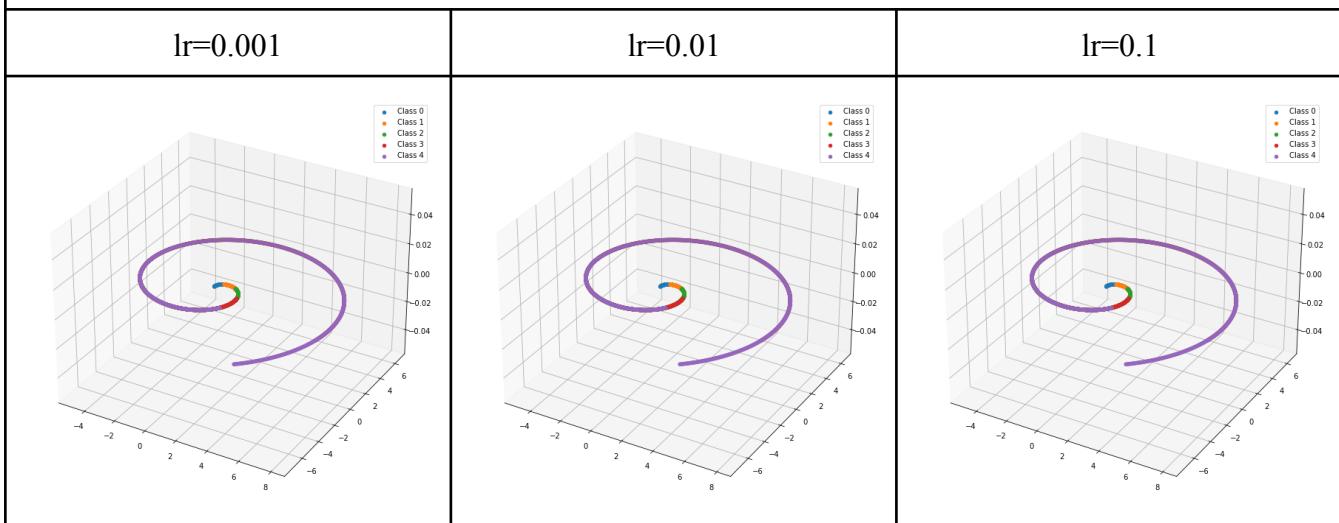
ARCHITEKTURA 1

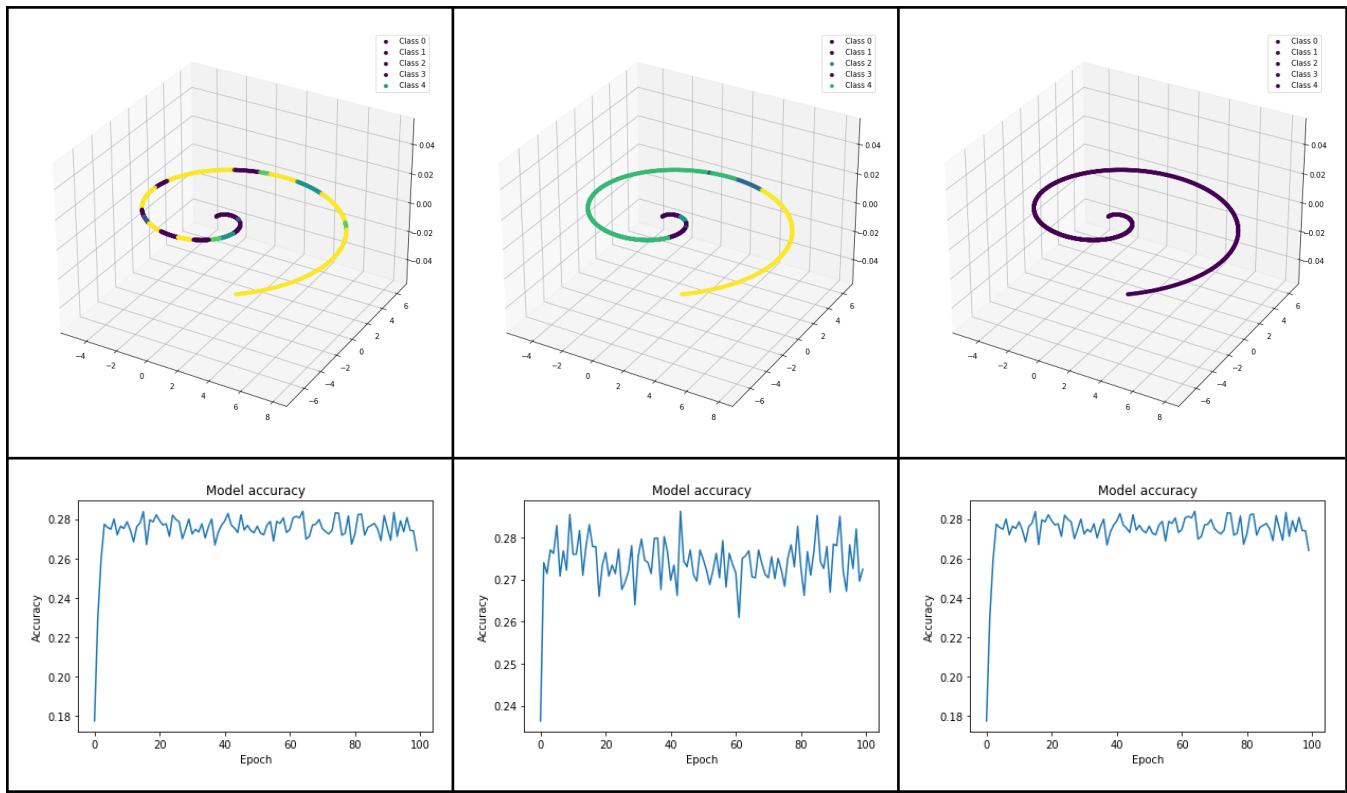
```
def experiment(num_layers=3, neurons_per_layer=32, activation='relu',
epochs=100):
    X, y = generate_dataset()
    plot_dataset(X, y)

    classifier = Classifier(num_layers, neurons_per_layer, activation)
    model = classifier.build_model(input_shape=2)

    optimizer = Adam(lr=0.001)
    history = train(X, y, model, optimizer, epochs)
    plot_history(history)

    plot_classification_results(X, y, model)
```





ARCHITEKTURA 2

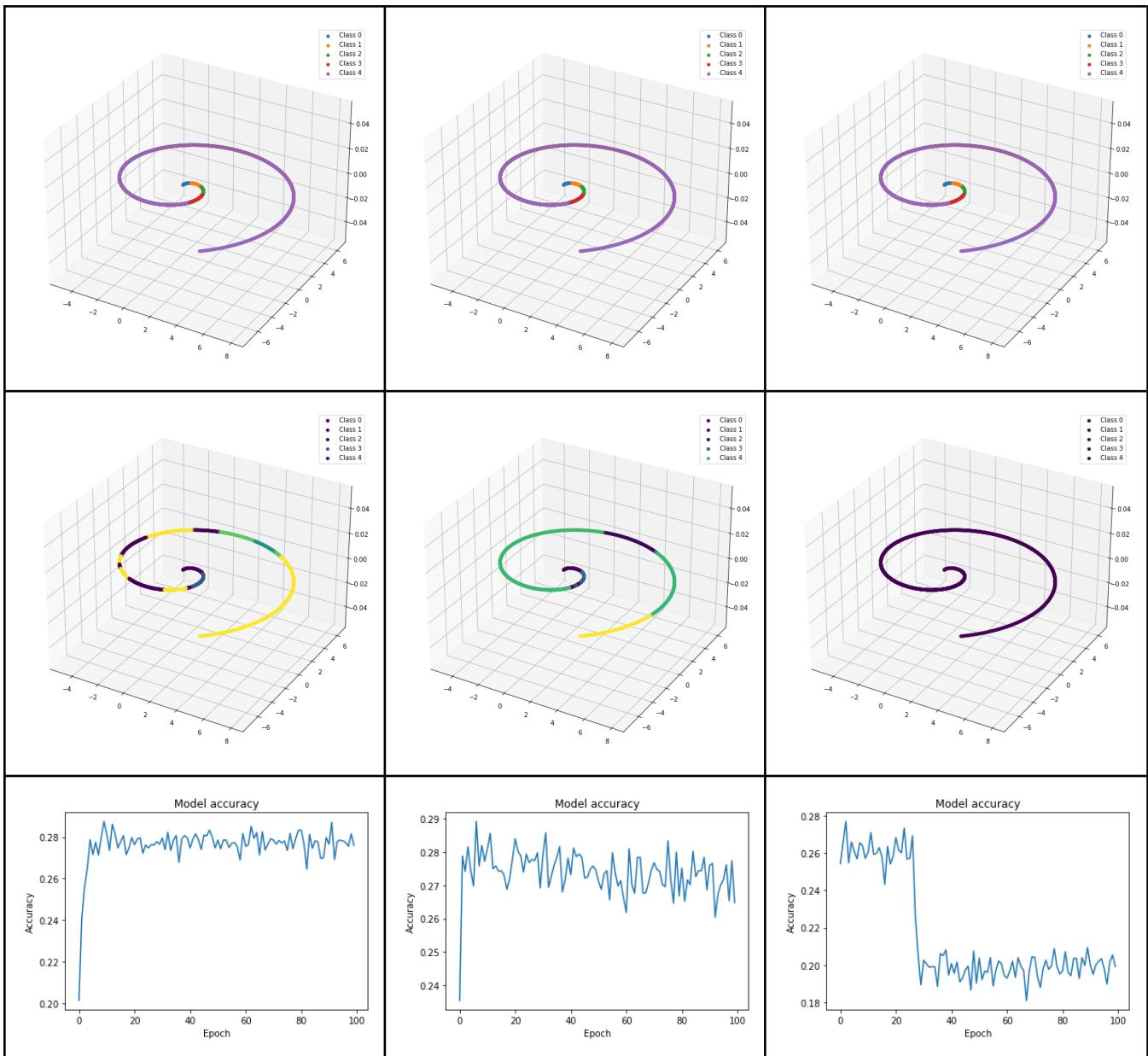
```
def experiment(num_layers=4, neurons_per_layer=64, activation='relu',
epochs=100):
    X, y = generate_dataset()
    plot_dataset(X, y)

    classifier = Classifier(num_layers, neurons_per_layer, activation)
    model = classifier.build_model(input_shape=2)

    optimizer = Adam(lr=0.001)
    history = train(X, y, model, optimizer, epochs)
    plot_history(history)

    plot_classification_results(X, y, model)
```

| | | |
|----------|---------|--------|
| lr=0.001 | lr=0.01 | lr=0.1 |
|----------|---------|--------|



ARCHITEKTURA 3

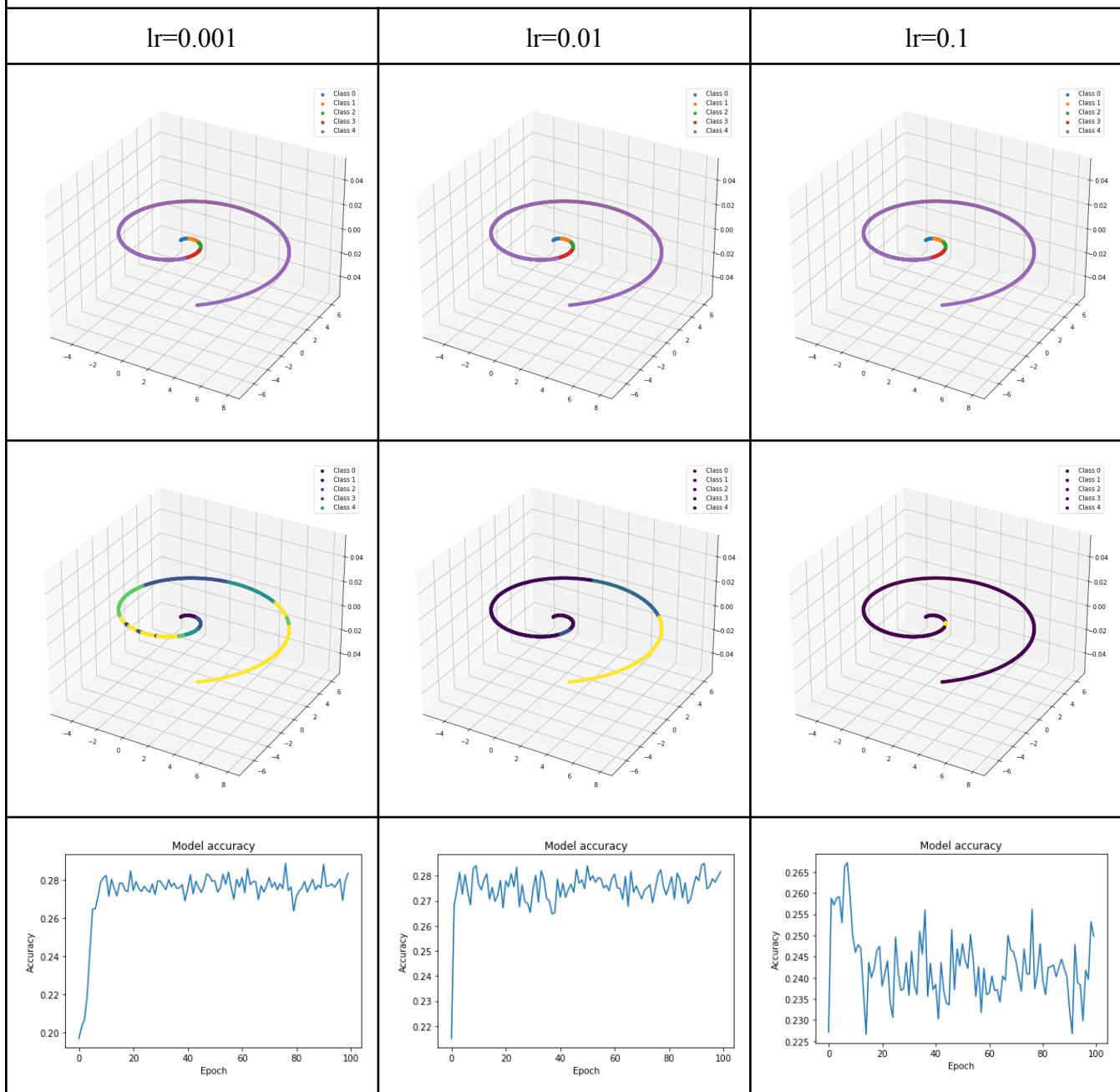
```
def experiment(num_layers=2, neurons_per_layer=64, activation='relu',
epochs=100):
    X, y = generate_dataset()
    plot_dataset(X, y)

    classifier = Classifier(num_layers, neurons_per_layer, activation)
    model = classifier.build_model(input_shape=2)

    optimizer = Adam(lr=0.001)
    history = train(X, y, model, optimizer, epochs)
```

```
plot_history(history)

plot_classification_results(X, y, model)
```



ARCHITEKTURA 4

```
def experiment(num_layers=2, neurons_per_layer=64, activation='relu',
epochs=100):
    X, y = generate_dataset()
    plot_dataset(X, y)
```

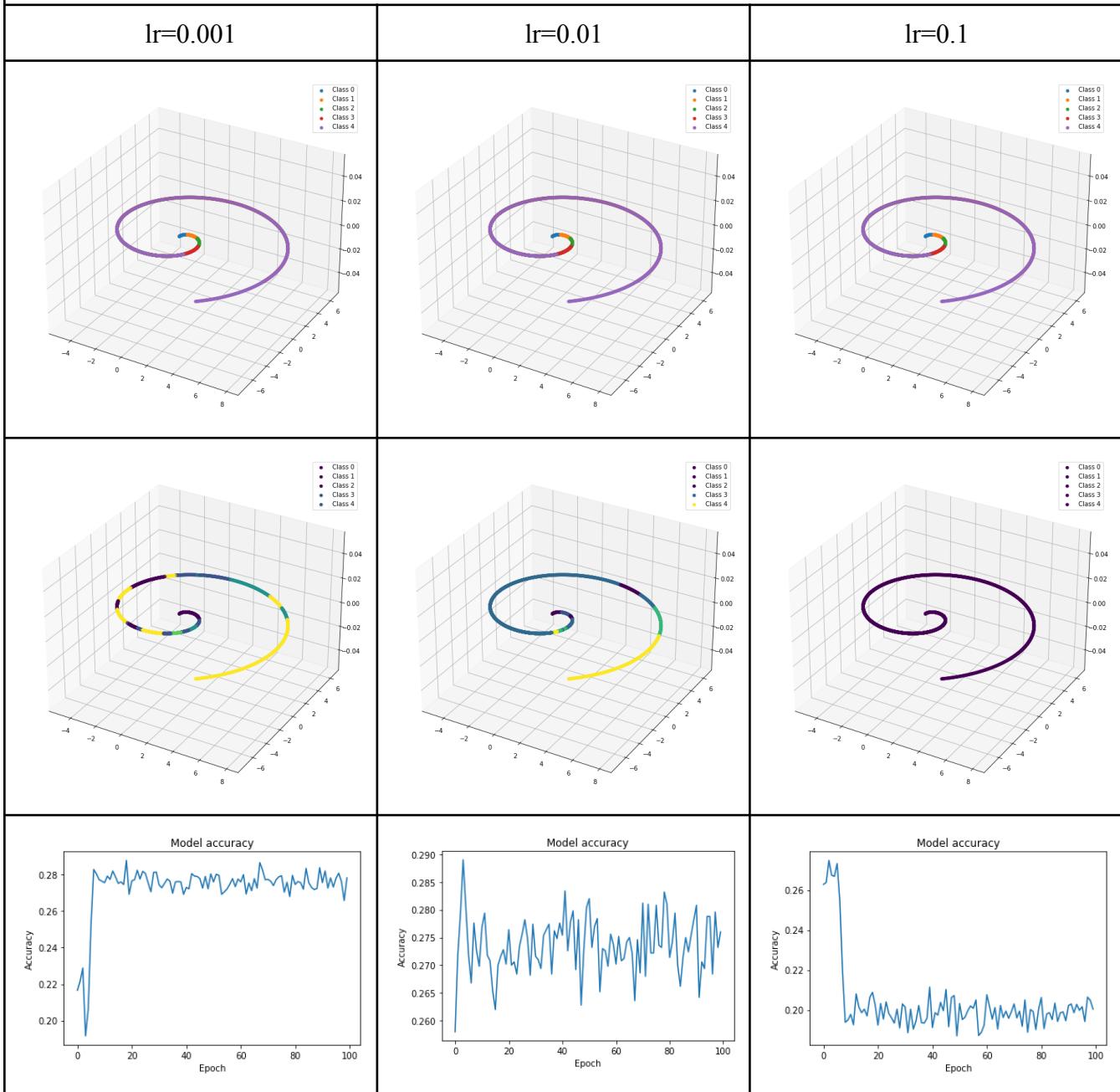
```

classifier = Classifier(num_layers, neurons_per_layer, activation)
model = classifier.build_model(input_shape=2)

optimizer = Nadam(lr=0.1)
history = train(X, y, model, optimizer, epochs)
plot_history(history)

plot_classification_results(X, y, model)

```



ARCHITEKTURA 5

```

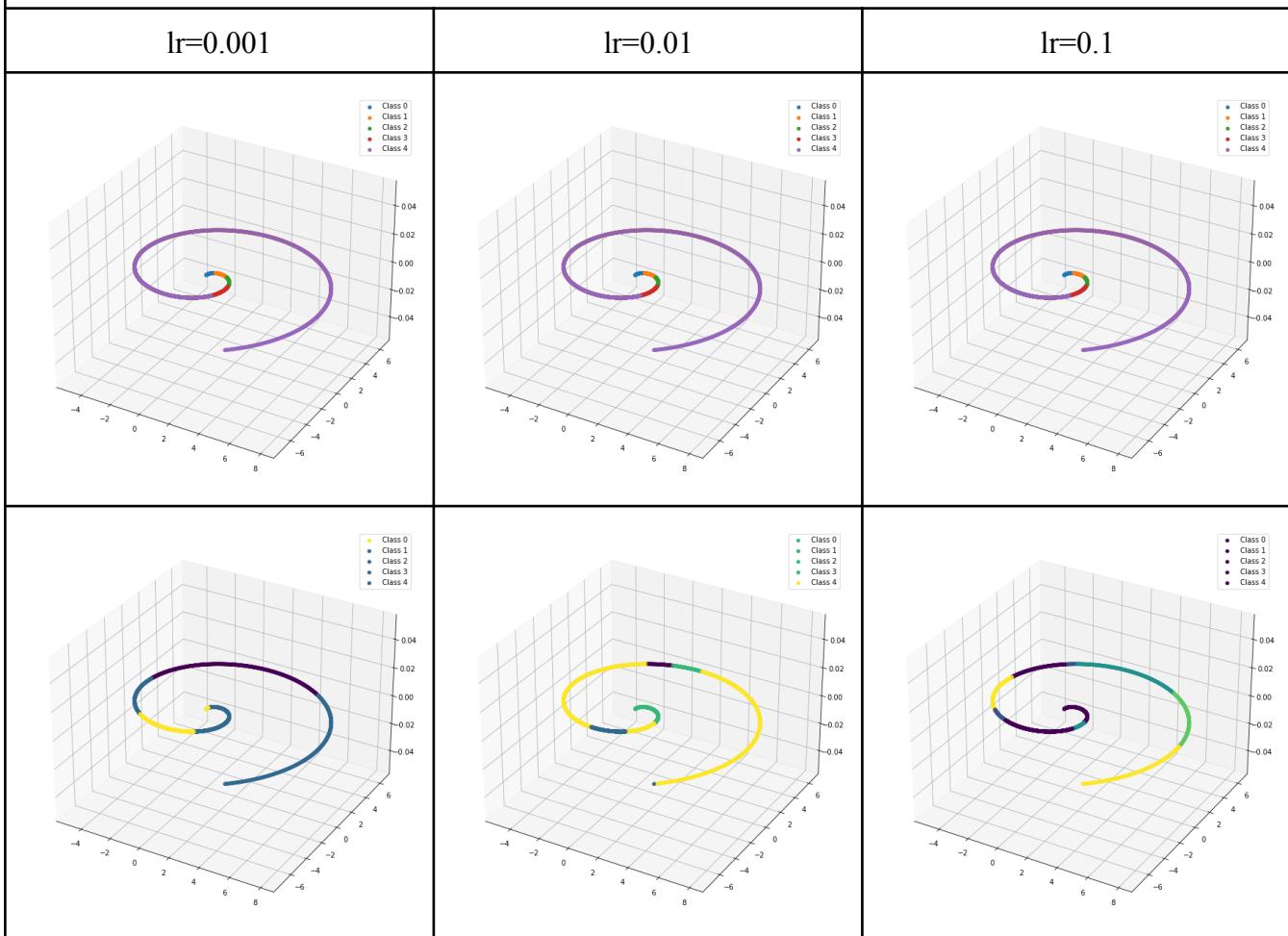
def experiment(num_layers=2, neurons_per_layer=64, activation='relu',
epochs=100):
    X, y = generate_dataset()
    plot_dataset(X, y)

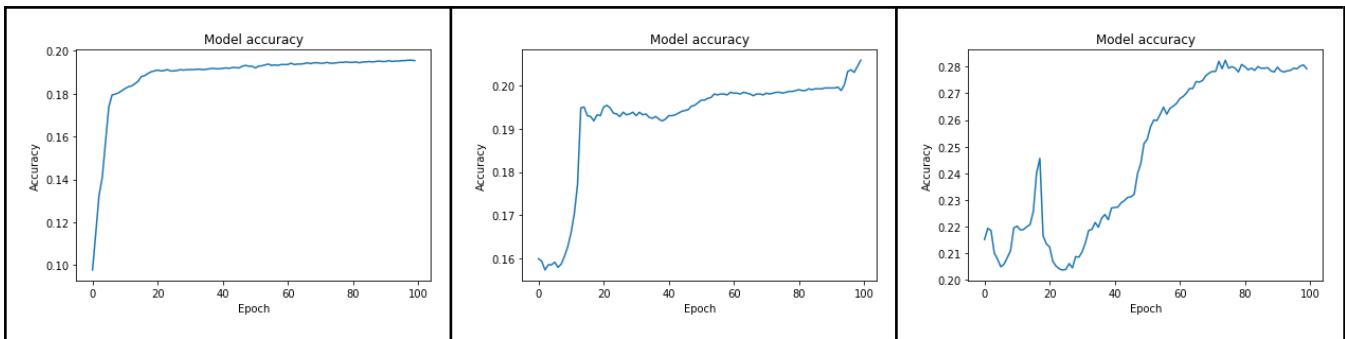
    classifier = Classifier(num_layers, neurons_per_layer, activation)
    model = classifier.build_model(input_shape=2)

    optimizer = Adadelta(lr=0.001)
    history = train(X, y, model, optimizer, epochs)
    plot_history(history)

    plot_classification_results(X, y, model)

```





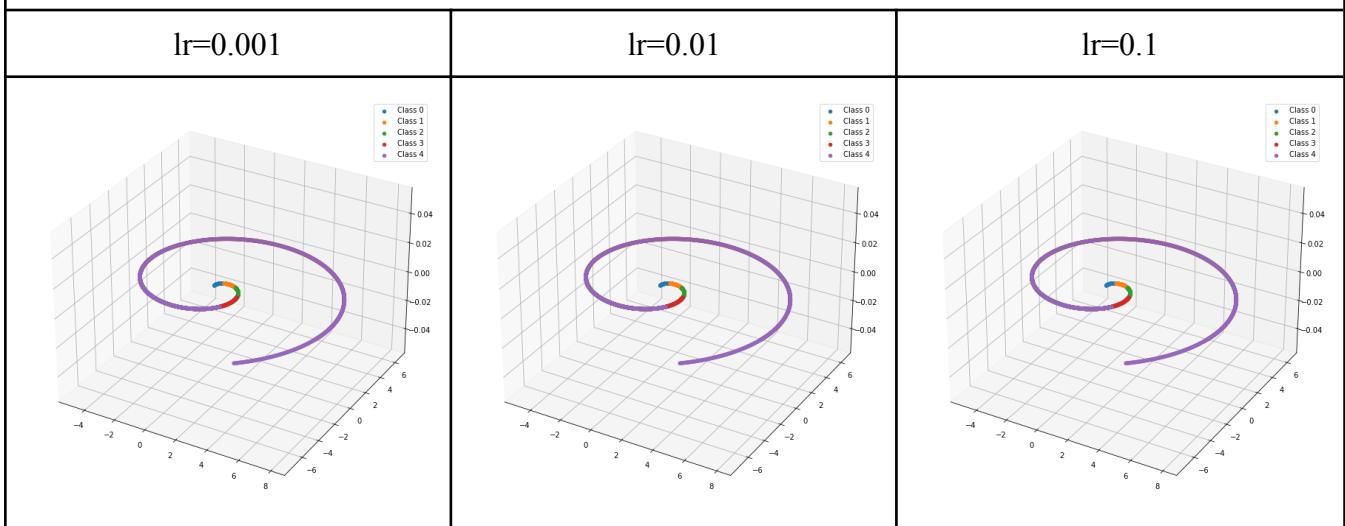
ARCHITEKTURA 6

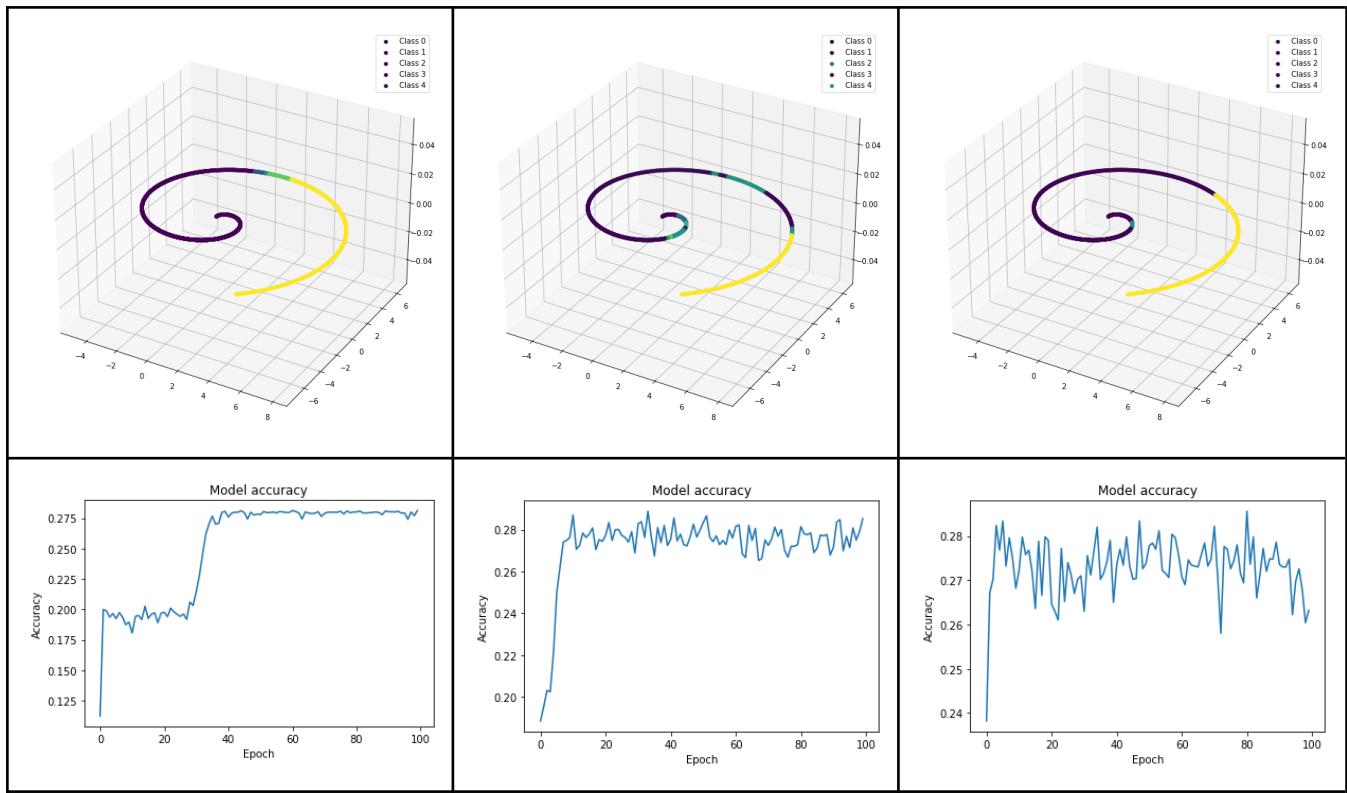
```
def experiment(num_layers=2, neurons_per_layer=64, activation='sigmoid',
epochs=100):
    X, y = generate_dataset()
    plot_dataset(X, y)

    classifier = Classifier(num_layers, neurons_per_layer, activation)
    model = classifier.build_model(input_shape=2)

    optimizer = Nadam(lr=0.001)
    history = train(X, y, model, optimizer, epochs)
    plot_history(history)

    plot_classification_results(X, y, model)
```





ARCHITEKTURA 7

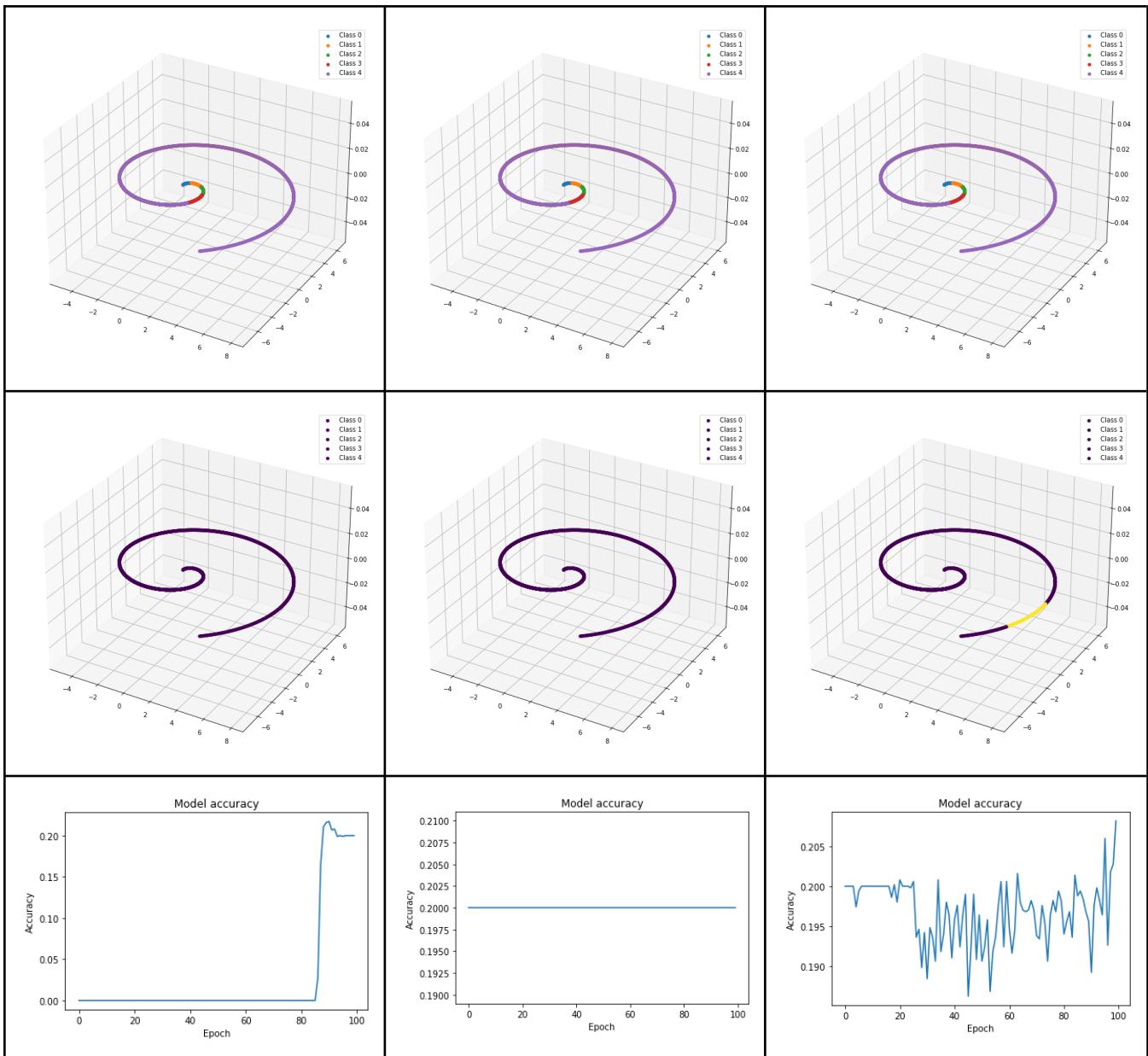
```
def experiment(num_layers=2, neurons_per_layer=64, activation='sigmoid',
epochs=100):
    X, y = generate_dataset()
    plot_dataset(X, y)

    classifier = Classifier(num_layers, neurons_per_layer, activation)
    model = classifier.build_model(input_shape=2)

    optimizer = Adadelta(lr=0.001)
    history = train(X, y, model, optimizer, epochs)
    plot_history(history)

    plot_classification_results(X, y, model)
```

| | | |
|----------|---------|--------|
| lr=0.001 | lr=0.01 | lr=0.1 |
|----------|---------|--------|



ARCHITEKTURA 8

```
def experiment(num_layers=2, neurons_per_layer=64, activation='tanh',
epochs=100):
    X, y = generate_dataset()
    plot_dataset(X, y)
```

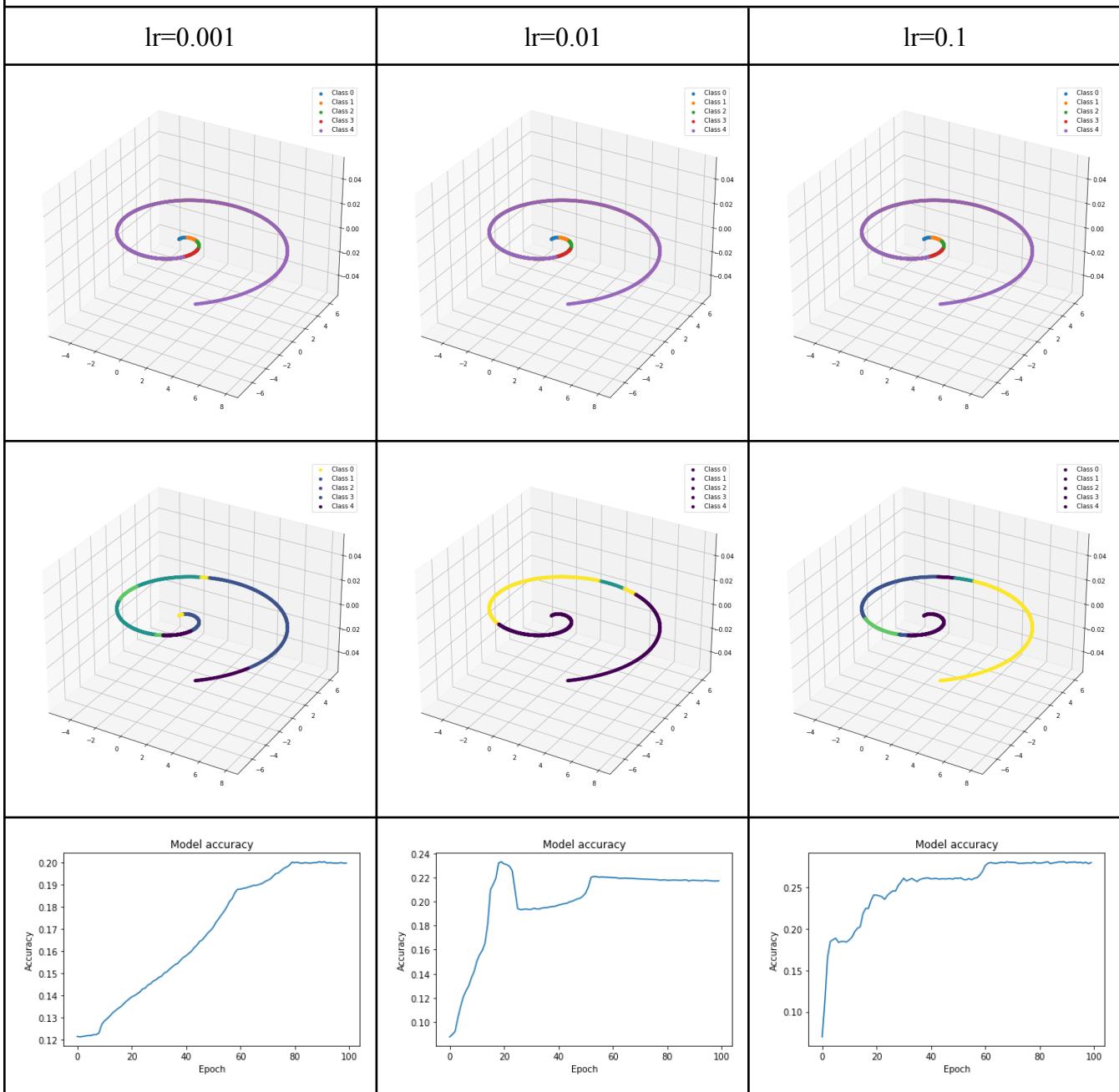
```
classifier = Classifier(num_layers, neurons_per_layer, activation)
model = classifier.build_model(input_shape=2)
```

```

optimizer = Adadelta(lr=0.001)
history = train(X, y, model, optimizer, epochs)
plot_history(history)

plot_classification_results(X, y, model)

```



ARCHITEKTURA 9

```

def experiment(num_layers=4, neurons_per_layer=128, activation='tanh',
epochs=100):
    X, y = generate_dataset()
    plot_dataset(X, y)

```

```

classifier = Classifier(num_layers, neurons_per_layer, activation)
model = classifier.build_model(input_shape=2)

```

```

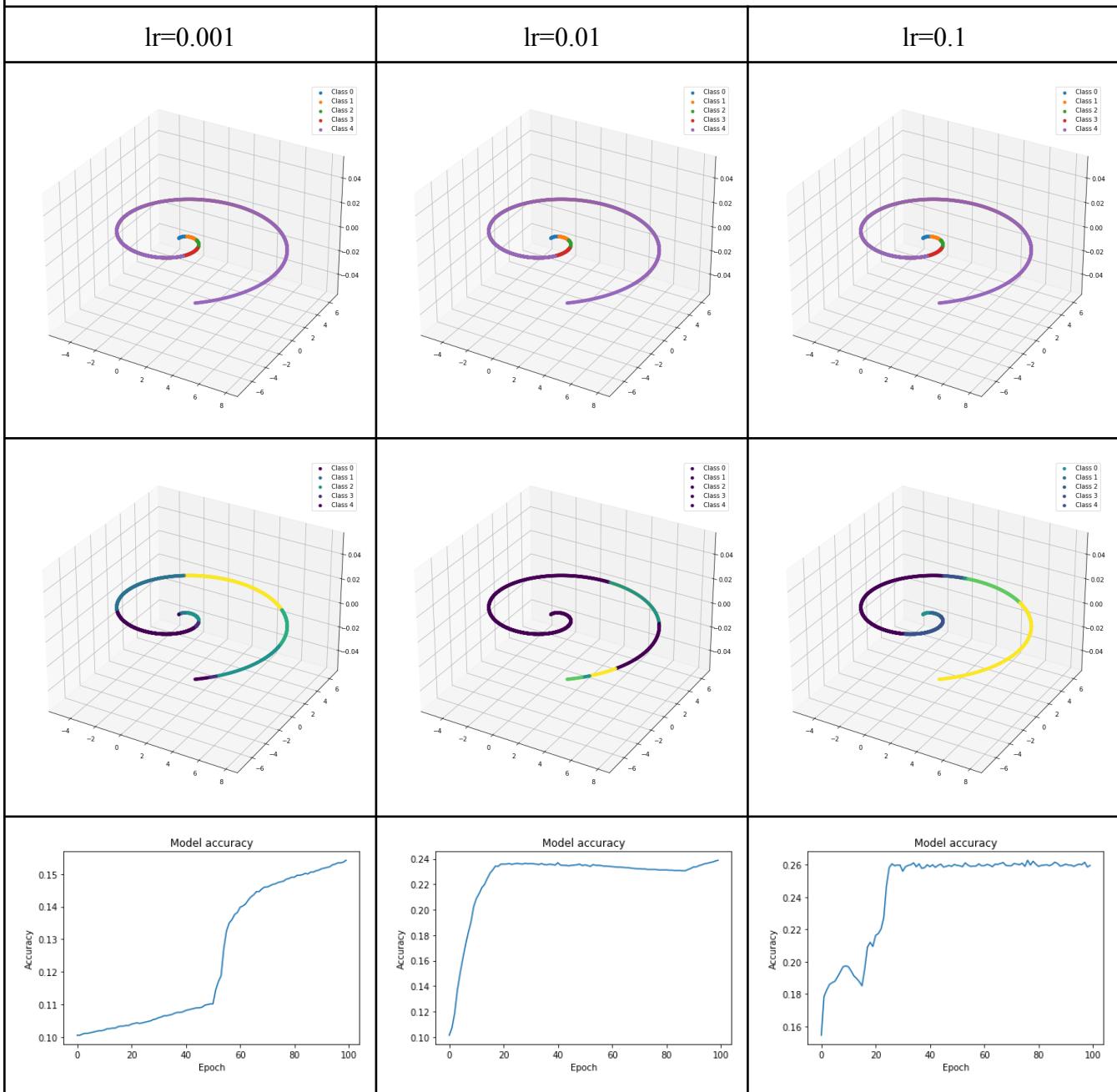
optimizer = Adadelta(lr=0.1)
history = train(X, y, model, optimizer, epochs)
plot_history(history)

```

```

plot_classification_results(X, y, model)

```



ARCHITEKTURA 10

```

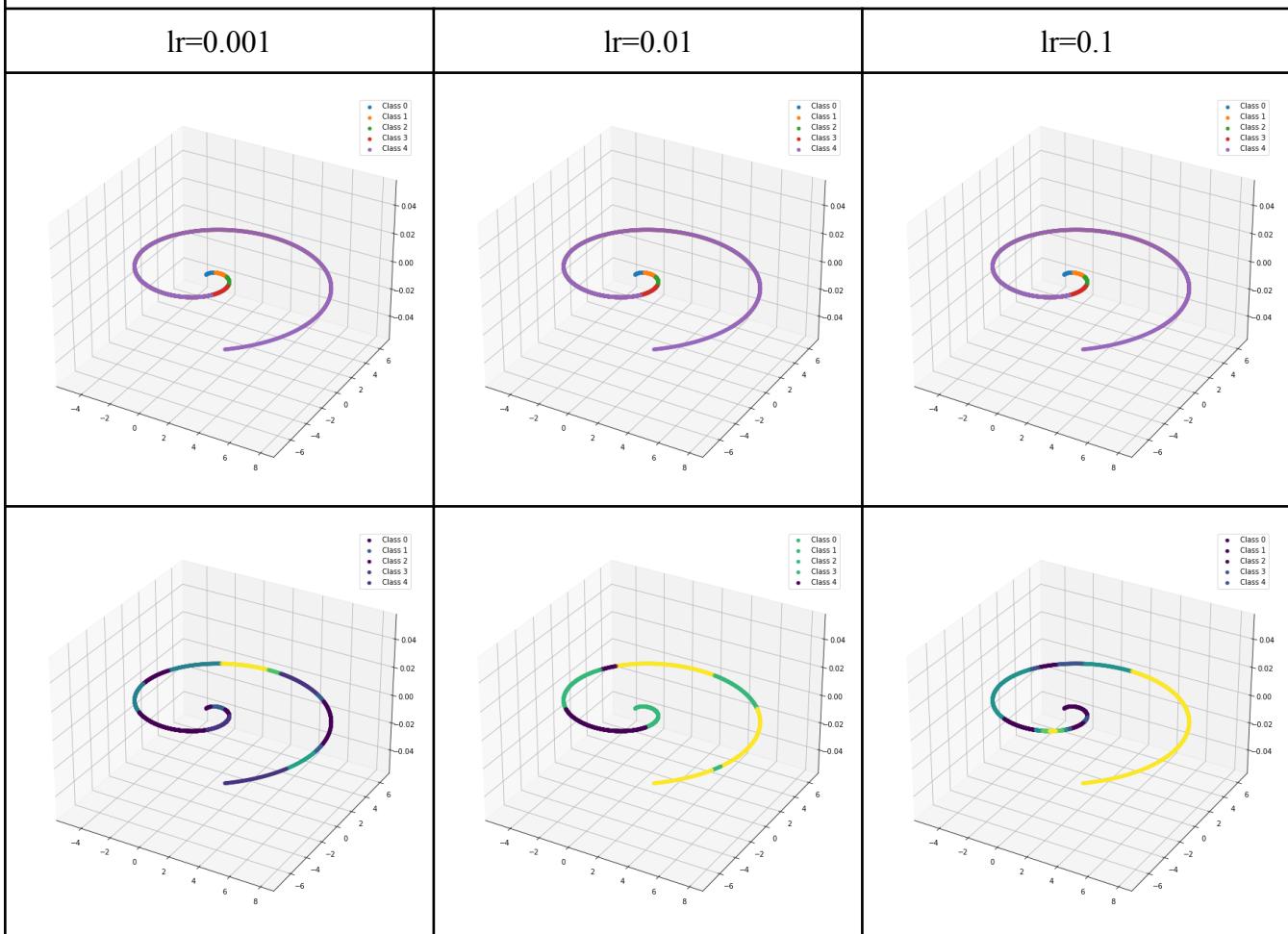
def experiment(num_layers=6, neurons_per_layer=512, activation='tanh',
epochs=100):
    X, y = generate_dataset()
    plot_dataset(X, y)

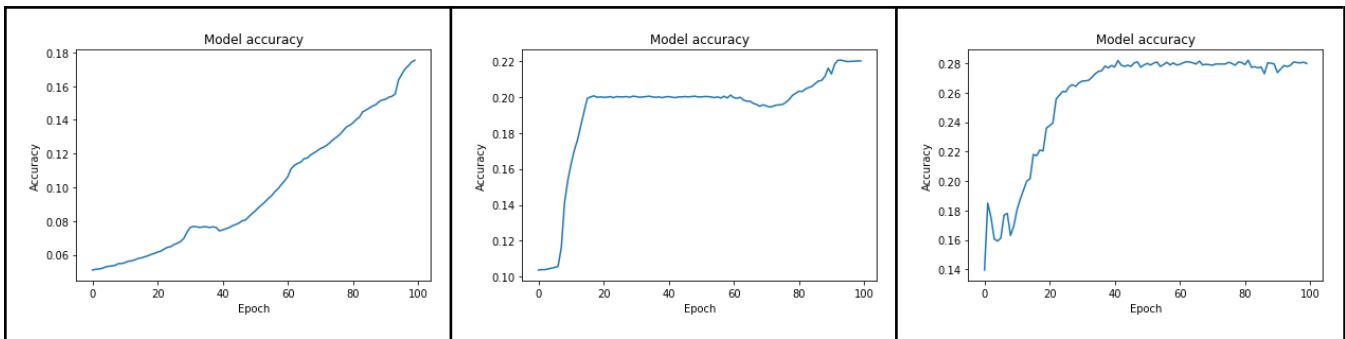
    classifier = Classifier(num_layers, neurons_per_layer, activation)
    model = classifier.build_model(input_shape=2)

    optimizer = Adadelta(lr=0.001)
    history = train(X, y, model, optimizer, epochs)
    plot_history(history)

    plot_classification_results(X, y, model)

```





Zbiór geometryczny nr 4

Do danego zbioru dołączono notatniki w formacie *ipynb* natomiast wyniki otrzymane zapisane zostały również poniżej.

https://github.com/anksunamoona/Uczenie_Maszynowe_Cwiczenie_2/tree/main/Zbior_Geometryczny_4

ARCHITEKTURA 1

```
def experiment(num_layers=3, neurons_per_layer=32, activation='relu',
epochs=100):
    X, y = generate_dataset()
    plot_dataset(X, y)

    classifier = Classifier(num_layers, neurons_per_layer, activation)
    model = classifier.build_model(input_shape=3)

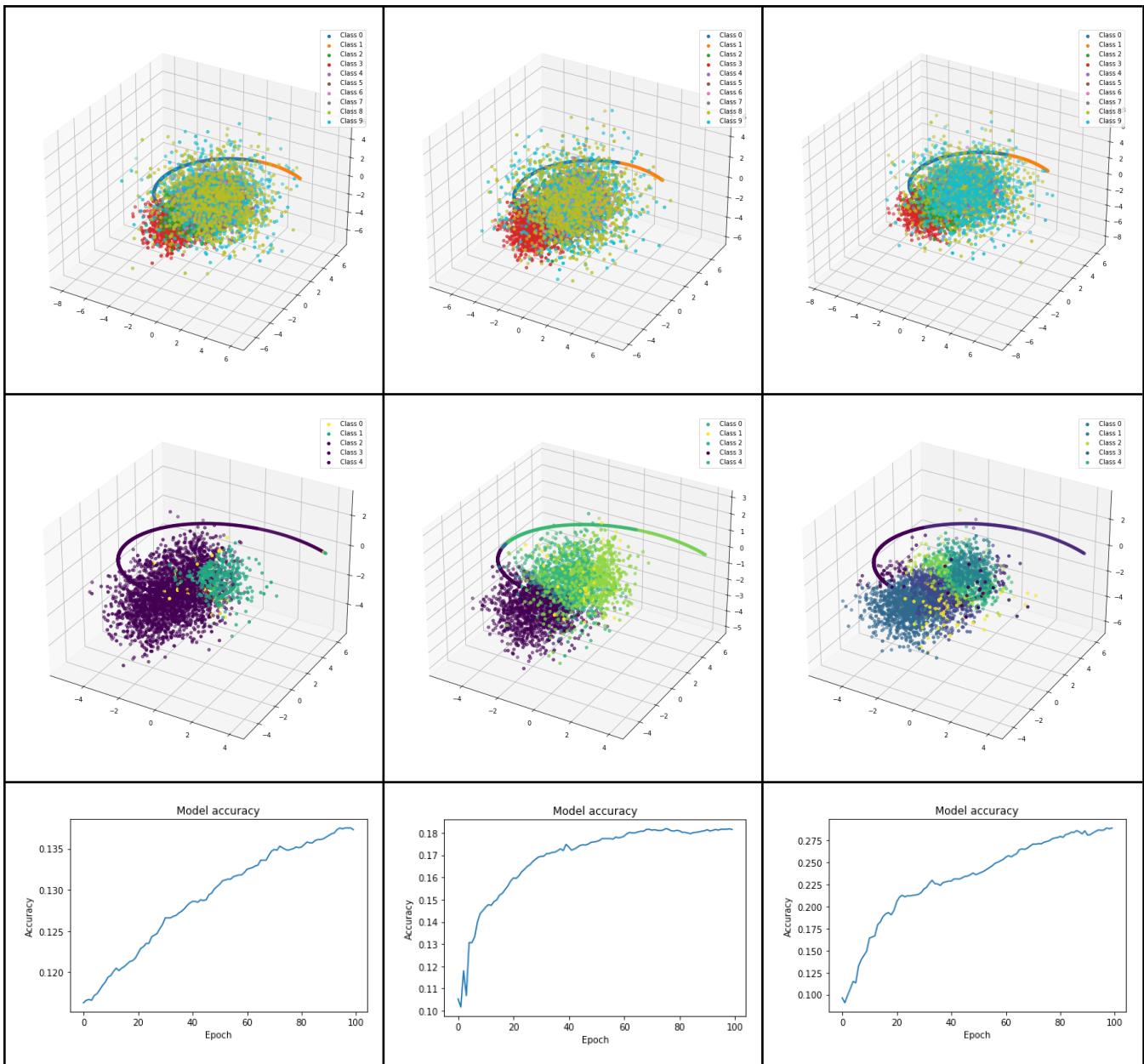
    optimizer = Adadelta(lr=0.001)
    history = train(X, y, model, optimizer, epochs)
    plot_history(history)

    plot_classification_results(X, y, model)
```

lr=0.001

lr=0.01

lr=0.1



ARCHITEKTURA 2

```

def experiment(num_layers=3, neurons_per_layer=32, activation='tanh',
epochs=100):
    X, y = generate_dataset()
    plot_dataset(X, y)

    classifier = Classifier(num_layers, neurons_per_layer, activation)
    model = classifier.build_model(input_shape=3)

    optimizer = Adadelta(lr=0.1)
    history = train(X, y, model, optimizer, epochs)

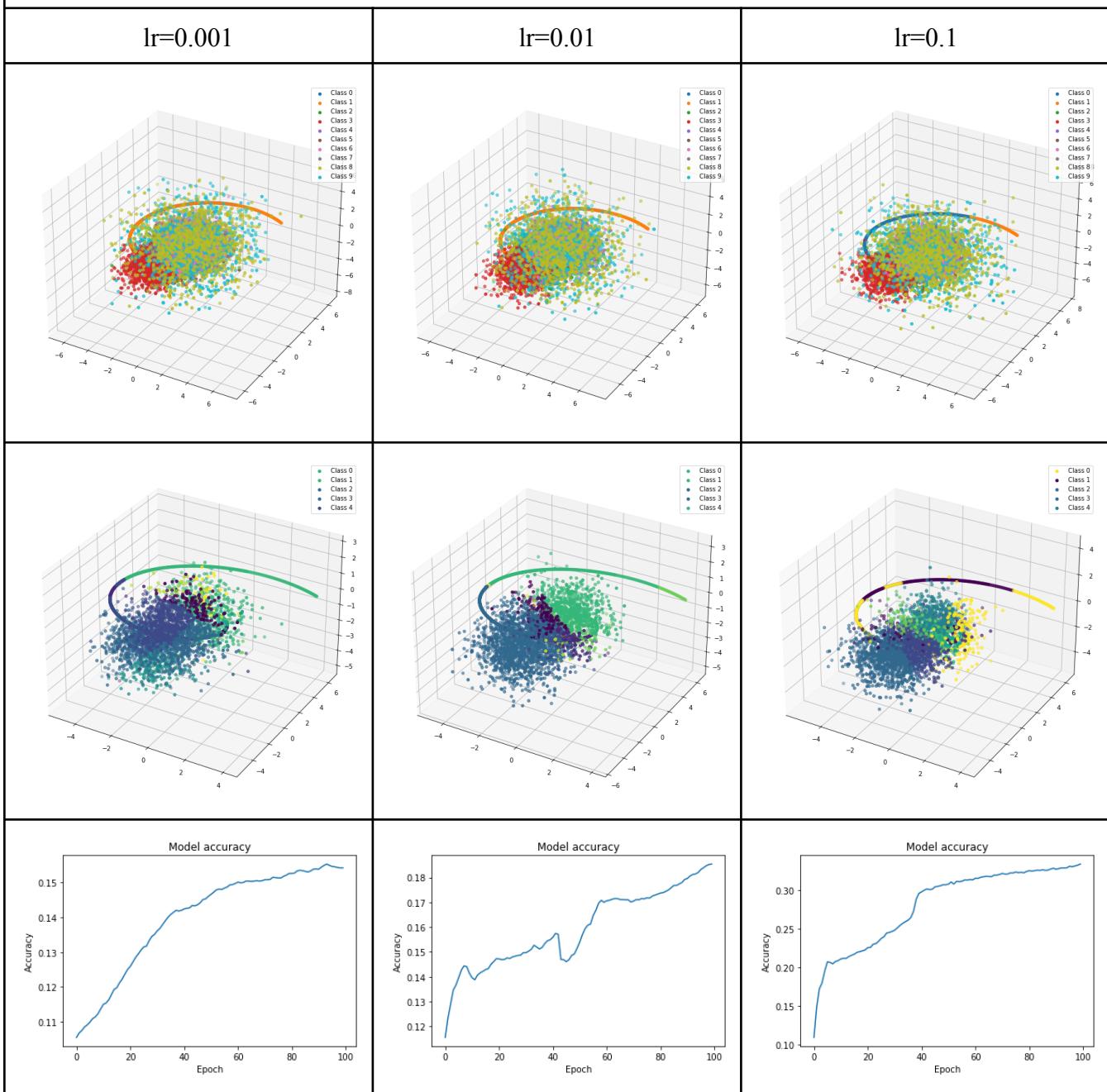
```

```

plot_history(history)

plot_classification_results(X, y, model)

```



ARCHITEKTURA 3

```

def experiment(num_layers=3, neurons_per_layer=32, activation='tanh',
epochs=100):
    X, y = generate_dataset()
    plot_dataset(X, y)

```

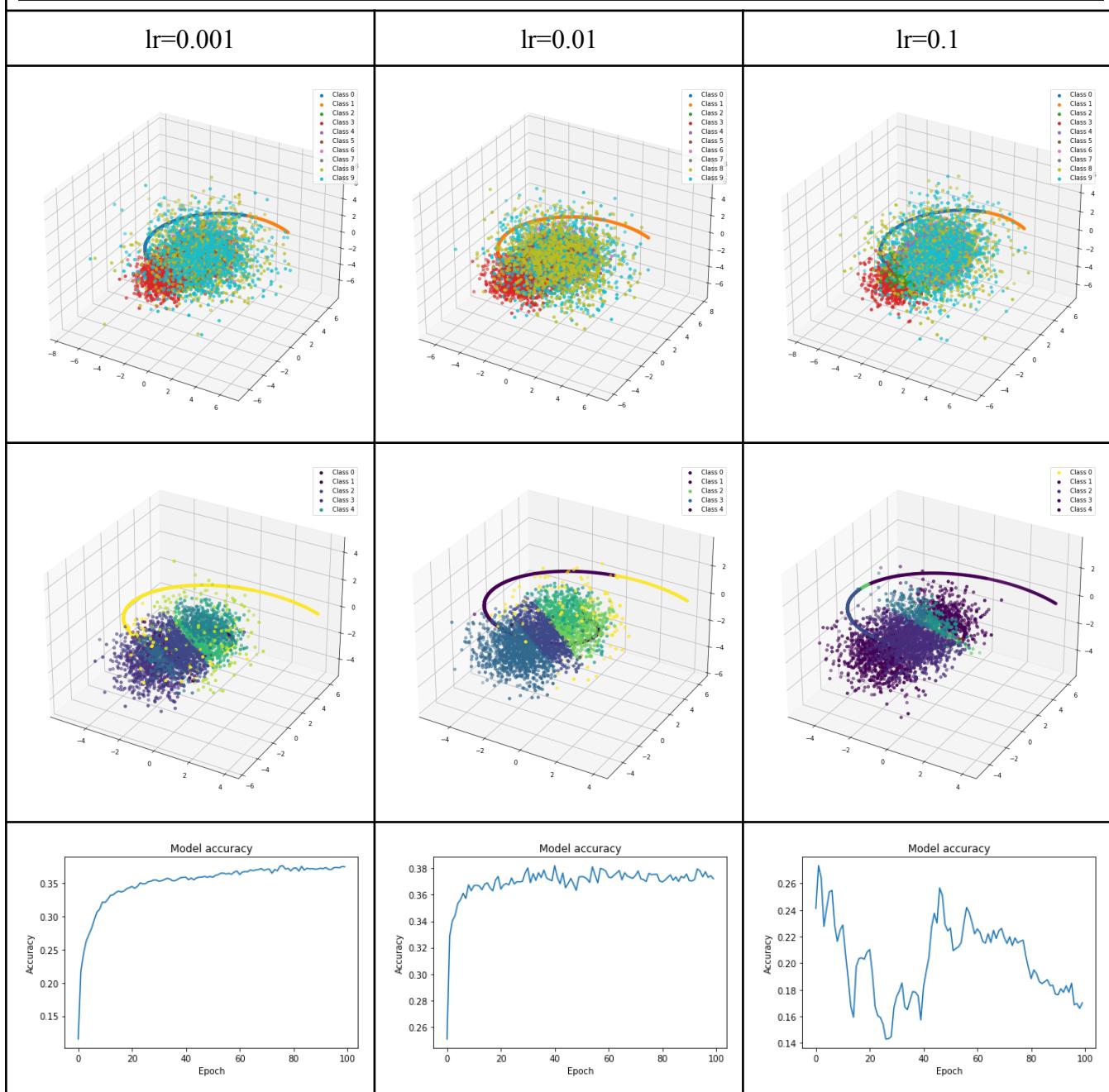
```

classifier = Classifier(num_layers, neurons_per_layer, activation)
model = classifier.build_model(input_shape=3)

optimizer = Nadam(lr=0.1)
history = train(X, y, model, optimizer, epochs)
plot_history(history)

plot_classification_results(X, y, model)

```



ARCHITEKTURA 4

```

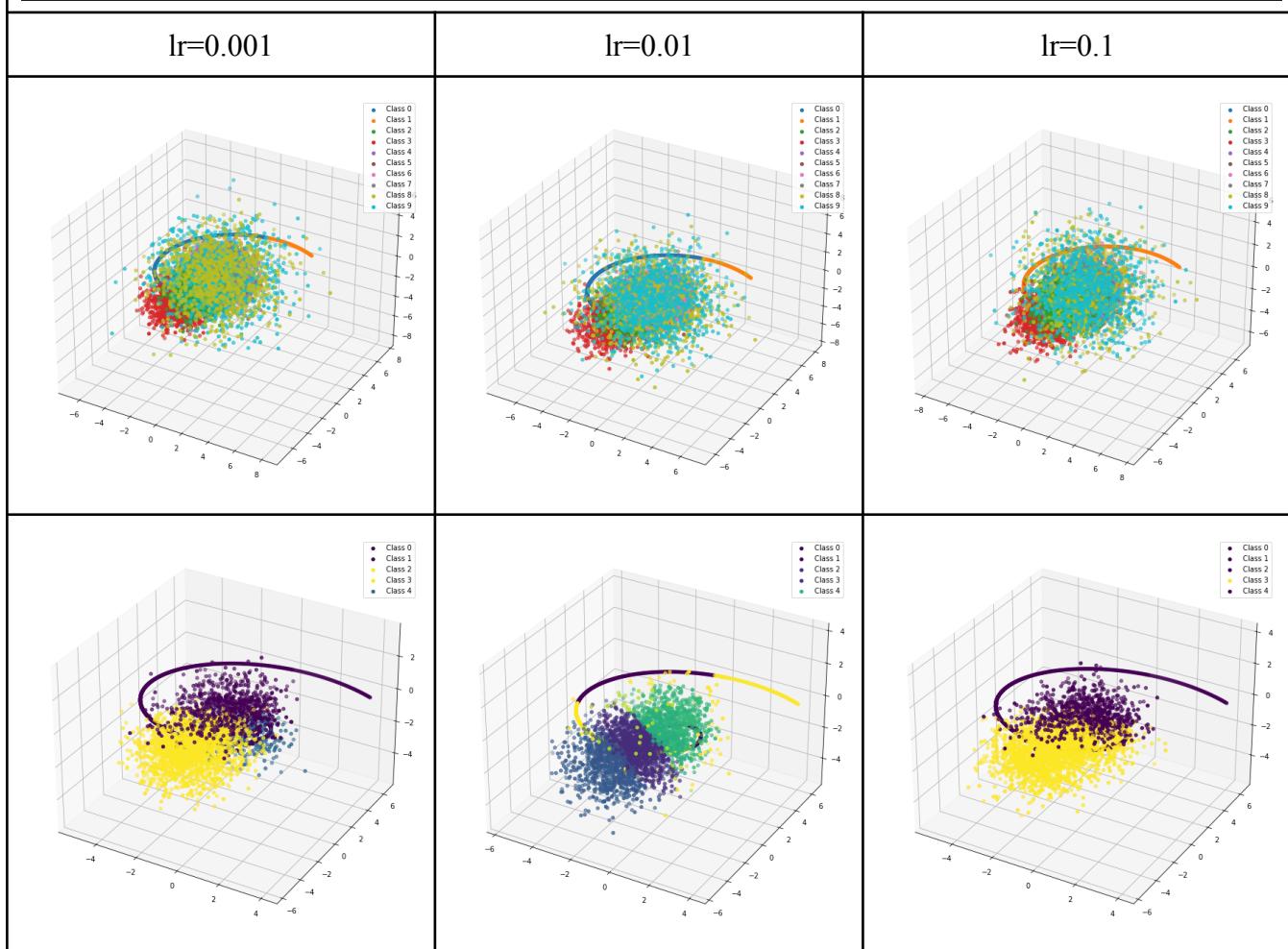
def experiment(num_layers=4, neurons_per_layer=216, activation='tanh',
epochs=100):
    X, y = generate_dataset()
    plot_dataset(X, y)

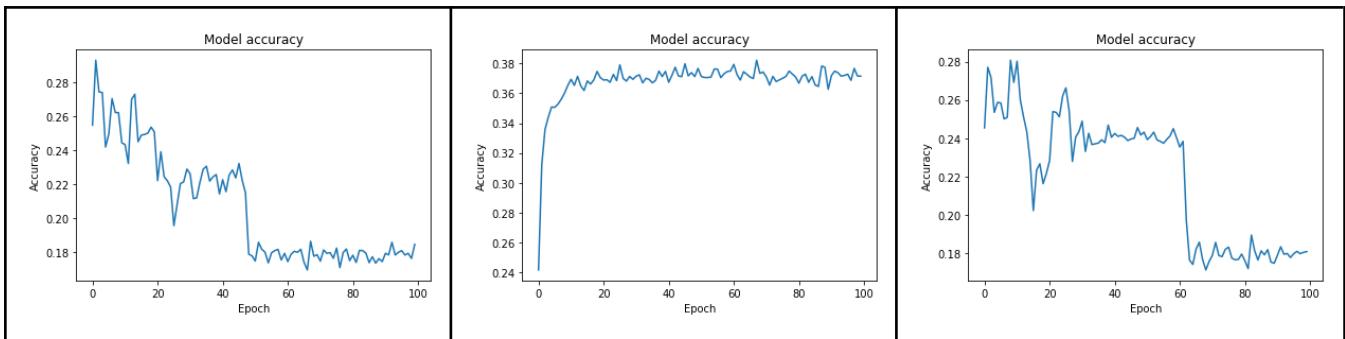
    classifier = Classifier(num_layers, neurons_per_layer, activation)
    model = classifier.build_model(input_shape=3)

    optimizer = Nadam(lr=0.1)
    history = train(X, y, model, optimizer, epochs)
    plot_history(history)

    plot_classification_results(X, y, model)

```





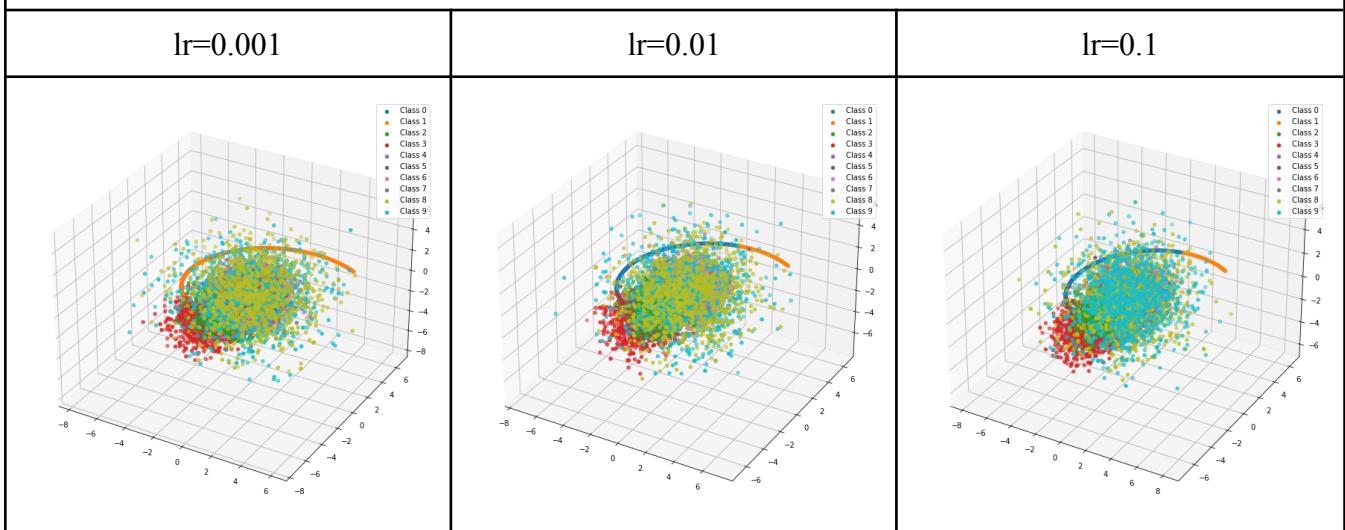
ARCHITEKTURA 5

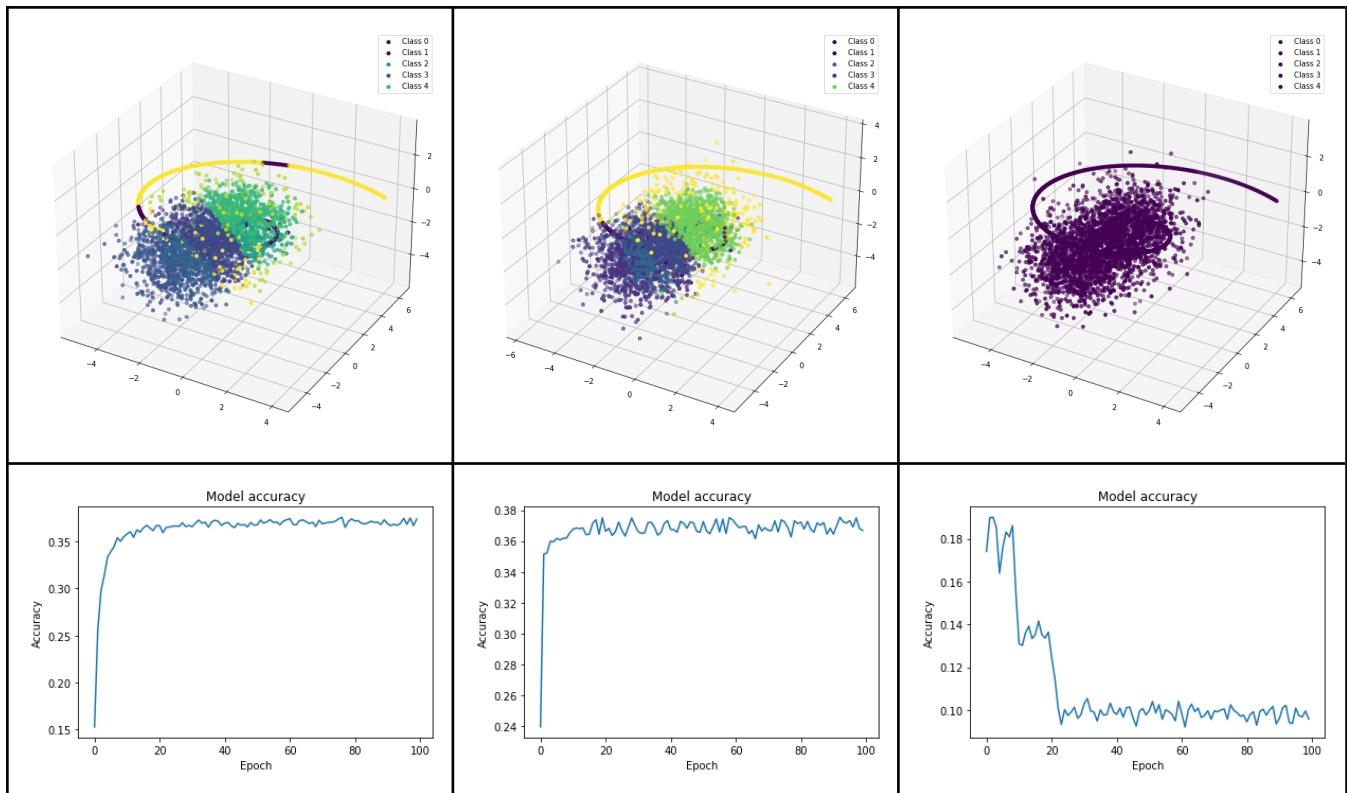
```
def experiment(num_layers=4, neurons_per_layer=216, activation='relu',
epochs=100):
    X, y = generate_dataset()
    plot_dataset(X, y)

    classifier = Classifier(num_layers, neurons_per_layer, activation)
    model = classifier.build_model(input_shape=3)

    optimizer = Nadam(lr=0.1)
    history = train(X, y, model, optimizer, epochs)
    plot_history(history)

    plot_classification_results(X, y, model)
```





ARCHITEKTURA 6

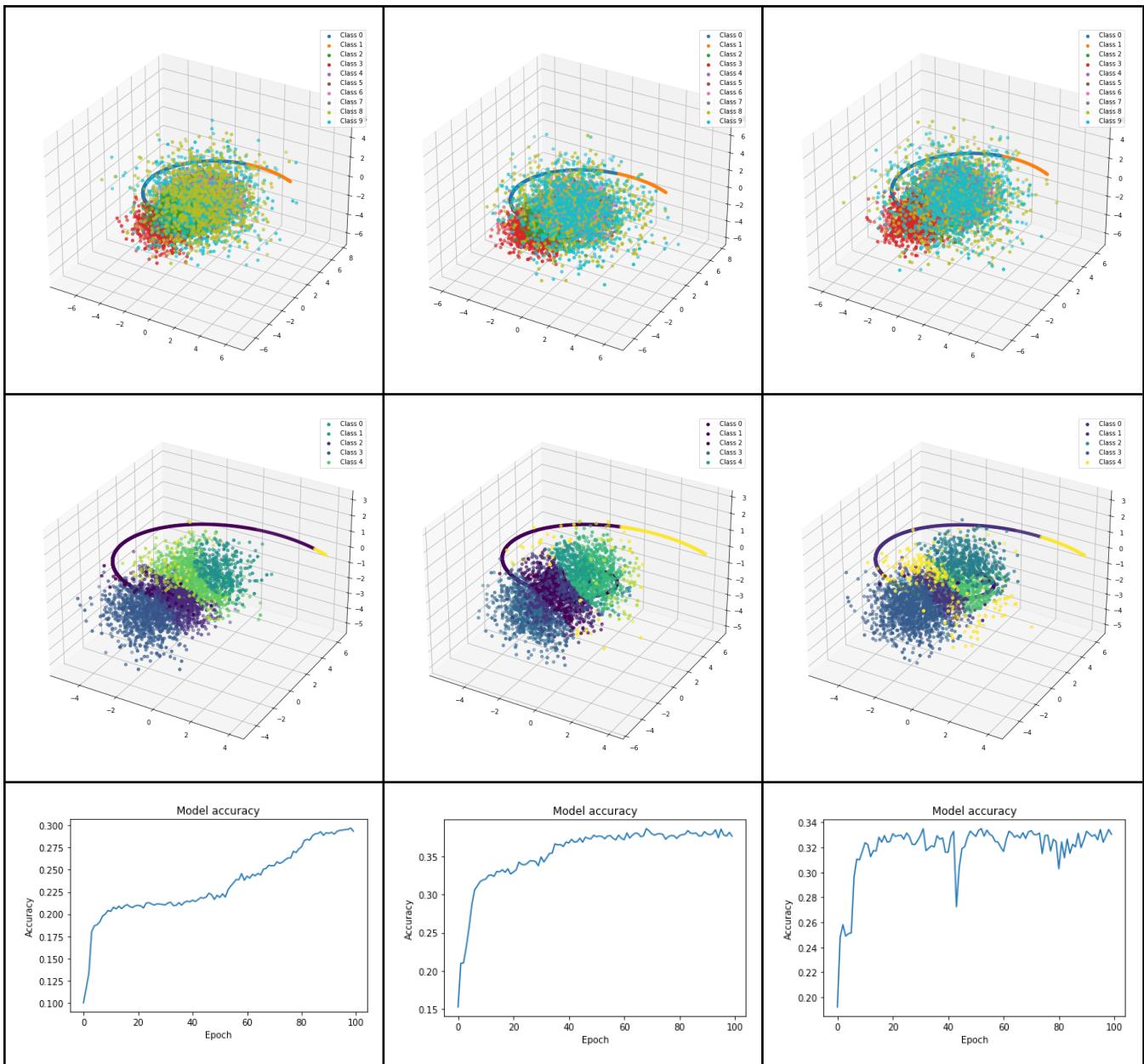
```
def experiment(num_layers=4, neurons_per_layer=216, activation='sigmoid',
epochs=100):
    X, y = generate_dataset()
    plot_dataset(X, y)

    classifier = Classifier(num_layers, neurons_per_layer, activation)
    model = classifier.build_model(input_shape=3)

    optimizer = Nadam(lr=0.001)
    history = train(X, y, model, optimizer, epochs)
    plot_history(history)

    plot_classification_results(X, y, model)
```

| | | |
|----------|---------|--------|
| lr=0.001 | lr=0.01 | lr=0.1 |
|----------|---------|--------|



ARCHITEKTURA 7

```

def experiment(num_layers=4, neurons_per_layer=216, activation='sigmoid',
epochs=100):
    X, y = generate_dataset()
    plot_dataset(X, y)

    classifier = Classifier(num_layers, neurons_per_layer, activation)
    model = classifier.build_model(input_shape=3)

    optimizer = SGD(lr=0.001)
    history = train(X, y, model, optimizer, epochs)

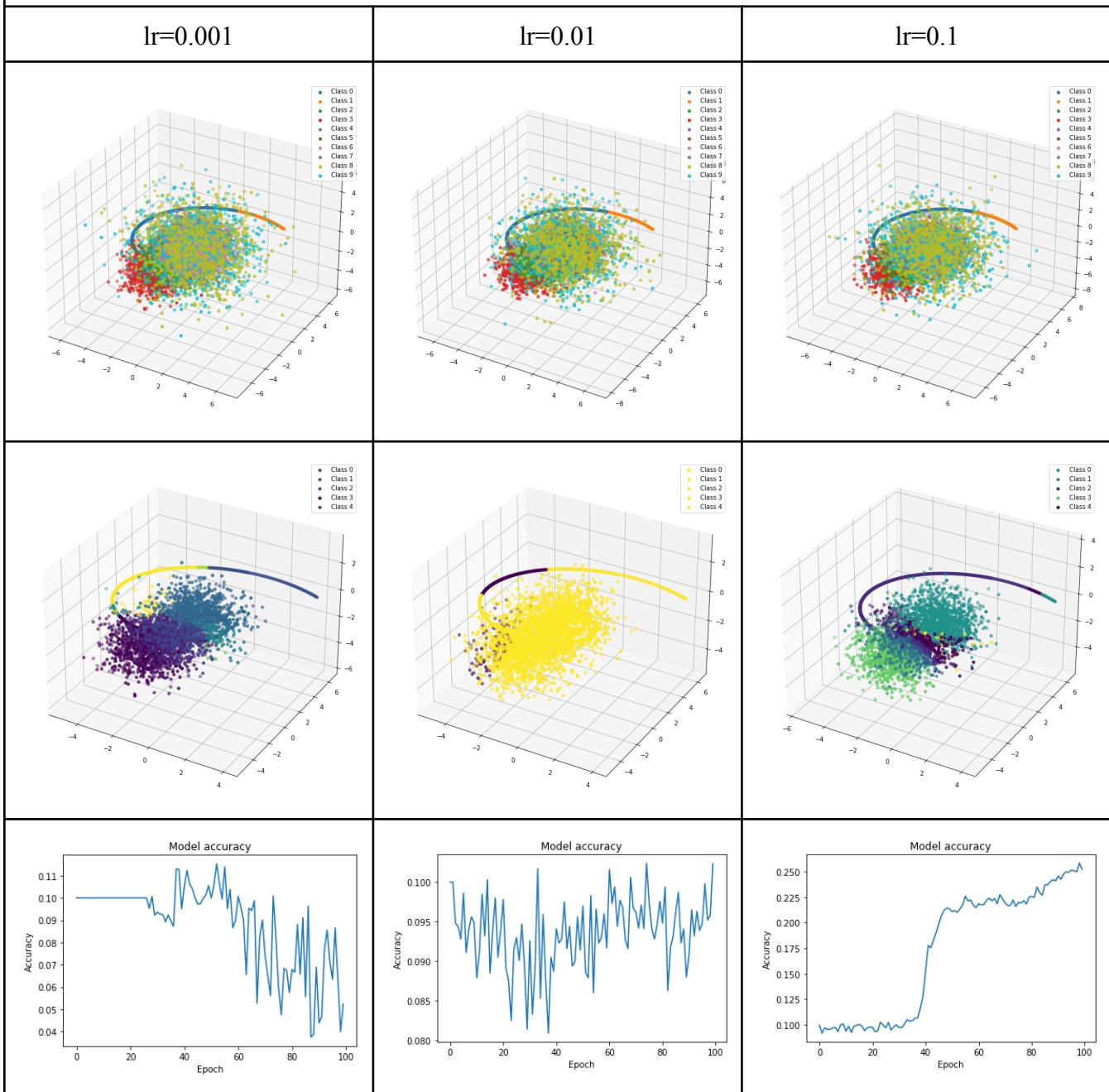
```

```

plot_history(history)

plot_classification_results(X, y, model)

```



ARCHITEKTURA 8

```

def experiment(num_layers=4, neurons_per_layer=216, activation='tanh',
epochs=100):
    X, y = generate_dataset()
    plot_dataset(X, y)

```

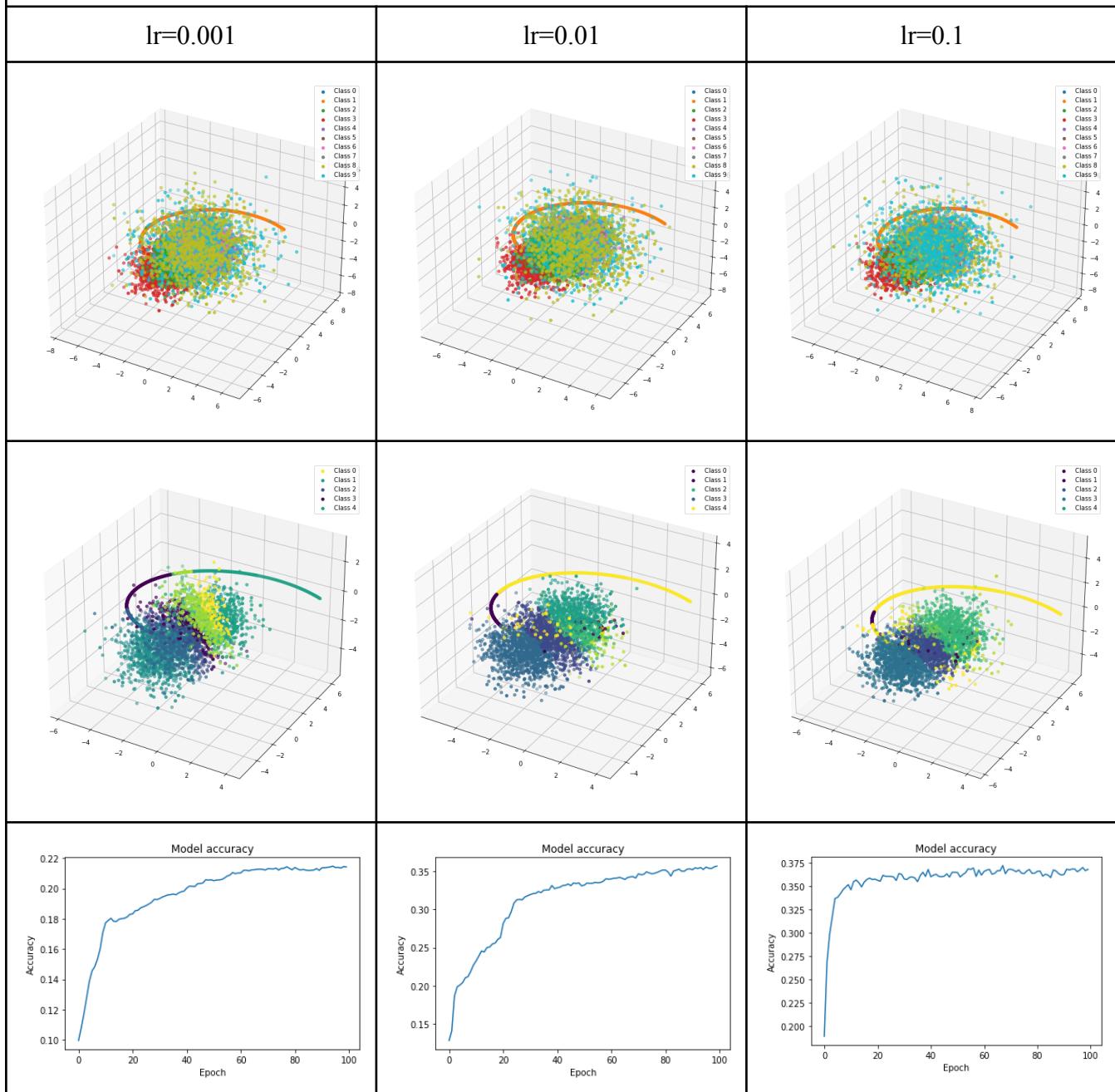
```

classifier = Classifier(num_layers, neurons_per_layer, activation)
model = classifier.build_model(input_shape=3)

optimizer = SGD(lr=0.1)
history = train(X, y, model, optimizer, epochs)
plot_history(history)

plot_classification_results(X, y, model)

```



ARCHITEKTURA 9

```

def experiment(num_layers=4, neurons_per_layer=216, activation='relu',

```

```

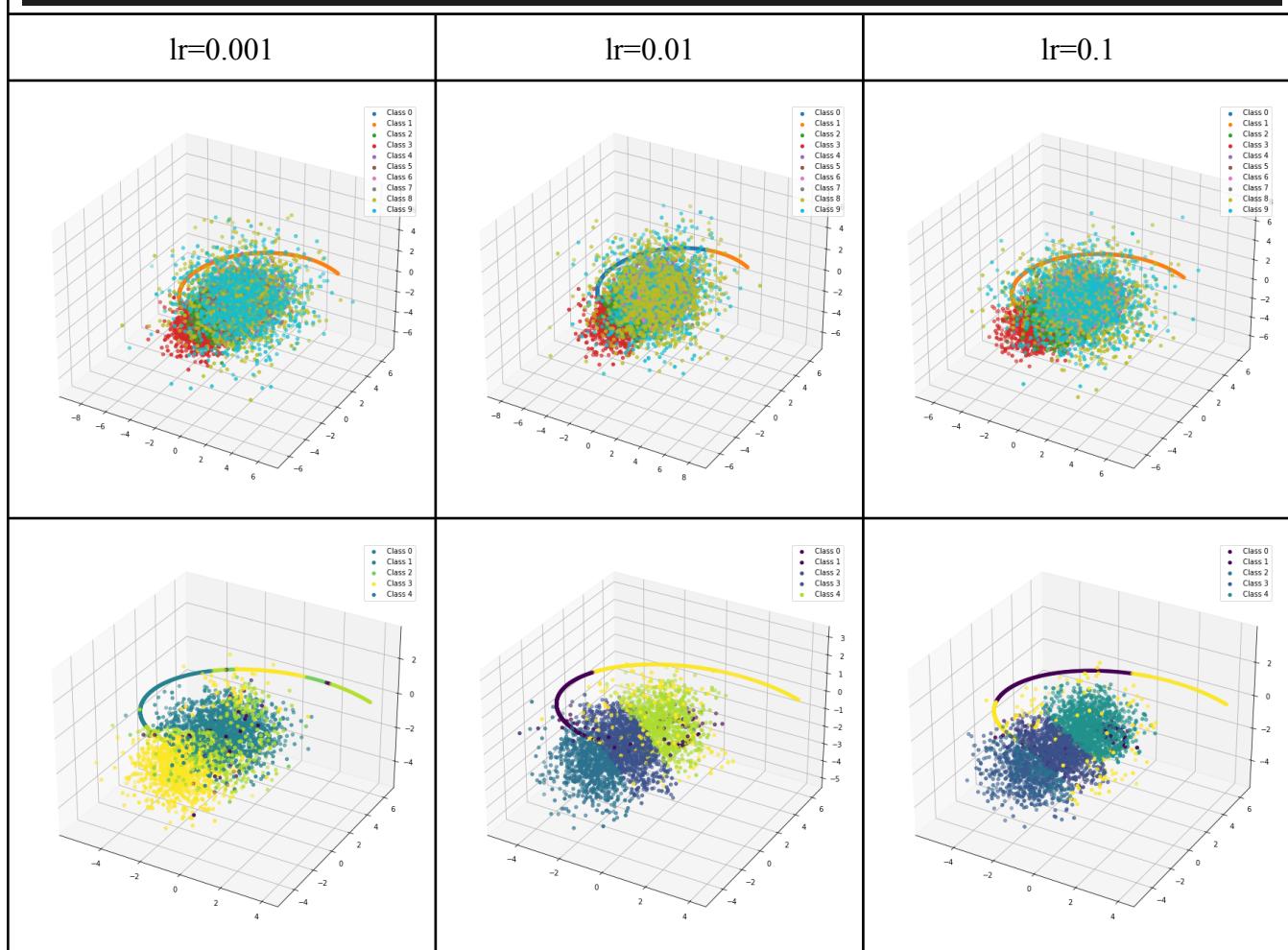
epochs=100):
    X, y = generate_dataset()
    plot_dataset(X, y)

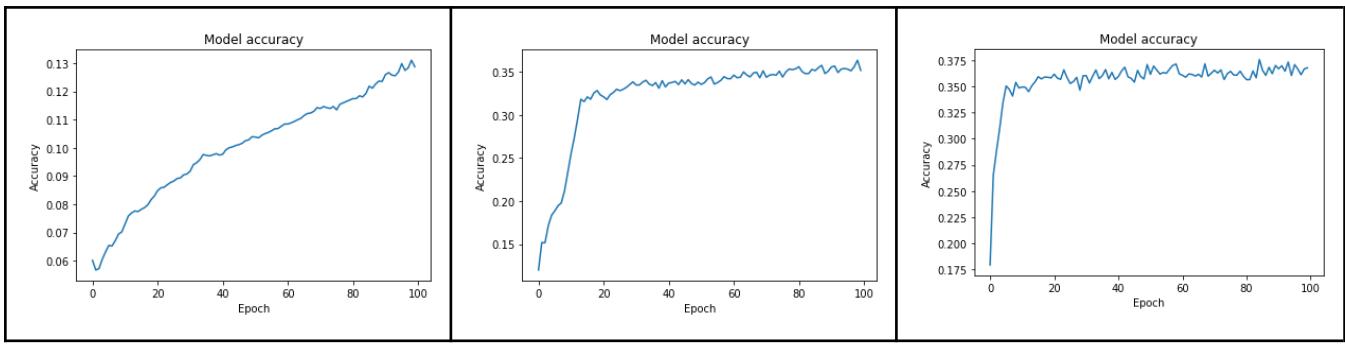
    classifier = Classifier(num_layers, neurons_per_layer, activation)
    model = classifier.build_model(input_shape=3)

    optimizer = SGD(lr=0.001)
    history = train(X, y, model, optimizer, epochs)
    plot_history(history)

    plot_classification_results(X, y, model)

```





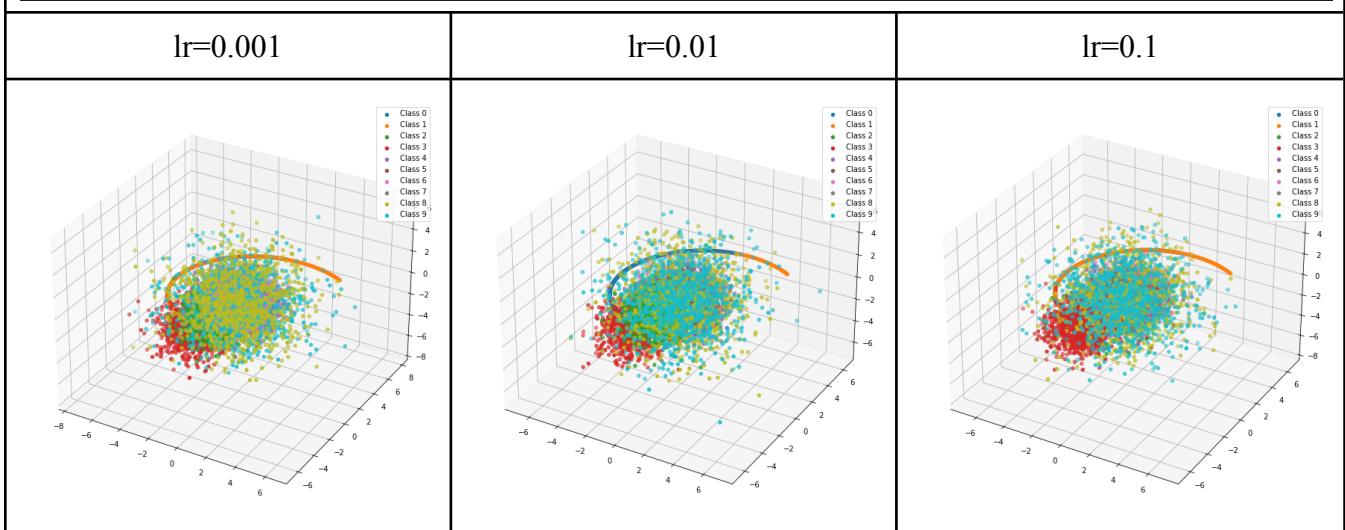
ARCHITEKTURA 10

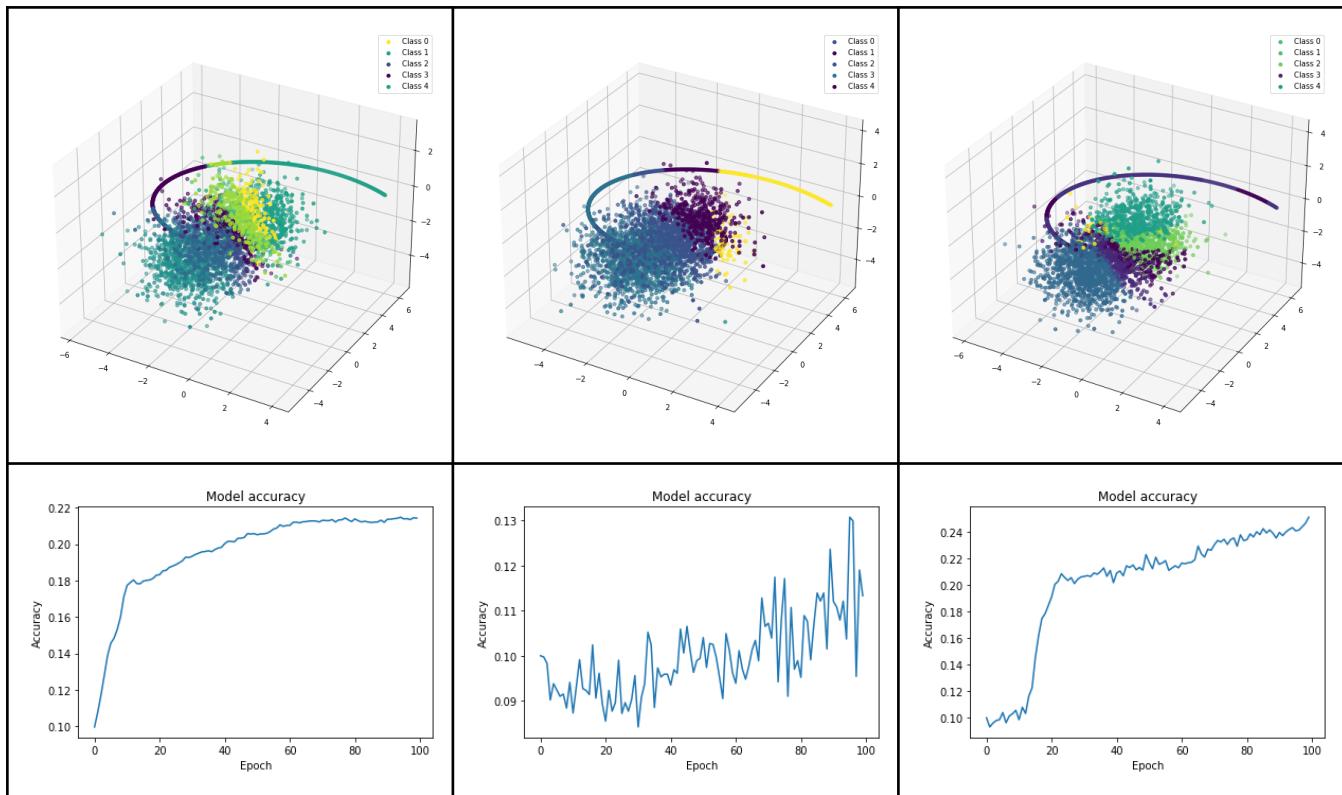
```
def experiment(num_layers=4, neurons_per_layer=216, activation='sigmoid',
epochs=100):
    X, y = generate_dataset()
    plot_dataset(X, y)

    classifier = Classifier(num_layers, neurons_per_layer, activation)
    model = classifier.build_model(input_shape=3)

    optimizer = SGD(lr=0.001)
    history = train(X, y, model, optimizer, epochs)
    plot_history(history)

    plot_classification_results(X, y, model)
```





ODPOWIEDZI NA PYTANIA:

- Dlaczego wyniki trenowania dają lepsze rezultaty niż w poprzednim zbiorze. Uzasadnić odpowiedź.

Wyniki trenowania mogą dawać lepsze rezultaty w tym przypadku ze względu na zmianę w generowaniu zbioru danych. W poprzednim zbiorze, klasy były generowane w sposób bardziej regularny, co mogło utrudnić klasyfikację, zwłaszcza dla niektórych architektur sieci. Natomiast w tym przypadku, dane są bardziej zróżnicowane, co może prowadzić do lepszej separacji klas i uzyskania wyższej jakości klasyfikacji.

- Czy architektura sieci ma znaczny wpływ na jakość klasyfikacji.

Tak, architektura sieci może mieć znaczący wpływ na jakość klasyfikacji. Różne architektury sieci neuronowych posiadają różne zdolności do modelowania złożonych relacji między danymi. Przez zmianę liczby warstw, liczby neuronów w warstwach, rodzaju aktywacji i innych parametrów, możemy wpływać na zdolności modelu do nauki i generalizacji. Optymalna architektura sieci zależy od konkretnej problematyki i dostępnych danych.

- Czy szybkość trenowania ma wpływ na jakość klasyfikacji.

Szybkość trenowania może mieć wpływ na jakość klasyfikacji. Zbyt szybkie uczenie może prowadzić do przeuczenia (overfitting), gdzie model doskonale dopasowuje się do danych treningowych, ale słabo generalizuje na nowe dane. Zbyt wolne uczenie może skutkować

tym, że model nie będzie w stanie dobrze nauczyć się zależności w danych. Optymalna szybkość trenowania zależy od specyfiki problemu i dostępnych danych.

4. Czy rodzaj optymalizatora ma wpływ na jakość klasyfikacji.

Rodzaj optymalizatora również może mieć wpływ na jakość klasyfikacji. Różne optymalizatory mają różne strategie aktualizacji wag sieci w procesie uczenia. Niektóre optymalizatory, takie jak SGD (Stochastic Gradient Descent), mogą mieć trudności z efektywnym znalezieniem optymalnego minimum globalnego funkcji kosztu. Inne optymalizatory, takie jak Adam czy Adadelta, oferują bardziej zaawansowane metody aktualizacji wag, które mogą przyspieszyć proces uczenia i pomóc w uniknięciu lokalnych minimów. Wybór optymalizatora zależy od specyfiki problemu, rozmiaru zbioru danych i preferencji.

Rozdział 2 – Klasyfikacja obrazów

2.1. Klasyfikacja klasyczna:

Wykonany notatnik:

[https://github.com/anksunamoona/Uczenie_Maszynowe_Cwiczenie_2
/blob/main/Rozdział%202.1/Rozdział_2_1%20trenowanie.ipynb](https://github.com/anksunamoona/Uczenie_Maszynowe_Cwiczenie_2/blob/main/Rozdział%202.1/Rozdział_2_1%20trenowanie.ipynb)

Na podstawie podanych wyników treningu możemy zrobić kilka wniosków:Dokładność na zbiorze treningowym (accuracy: 0.9150) i zbiorze walidacyjnym (val_accuracy: 0.3600) różnią się znacznie. To może wskazywać na przeuczenie (overfitting) modelu, czyli sytuację, gdy model dobrze radzi sobie z danymi treningowymi, ale słabo generalizuje na nowe dane.Zarówno funkcja straty (loss) jak i dokładność (accuracy) na zbiorze treningowym maleją z każdą kolejną epoką, co sugeruje, że model może mieć problemy z nauką na tych danych.Dokładność na zbiorze walidacyjnym również nie wykazuje znaczącego wzrostu z każdą epoką, co może świadczyć o braku postępu w uczeniu się modelu.Na podstawie tych obserwacji można stwierdzić, że wyniki otrzymane po 10 epokach nie są jeszcze zadowalające. Model może wymagać dalszej optymalizacji i dostosowania, aby poprawić skuteczność klasyfikacji.

2.1.2. Powtarzanie trenowań:

Wynik pierwszego trenowania znajduje się w pliku:

[https://github.com/anksunamoona/Uczenie_Maszynowe_Cwiczenie_2
/blob/main/Rozdział%202.1/Rozdział_2_1_2_1.ipynb](https://github.com/anksunamoona/Uczenie_Maszynowe_Cwiczenie_2/blob/main/Rozdział%202.1/Rozdział_2_1_2_1.ipynb)

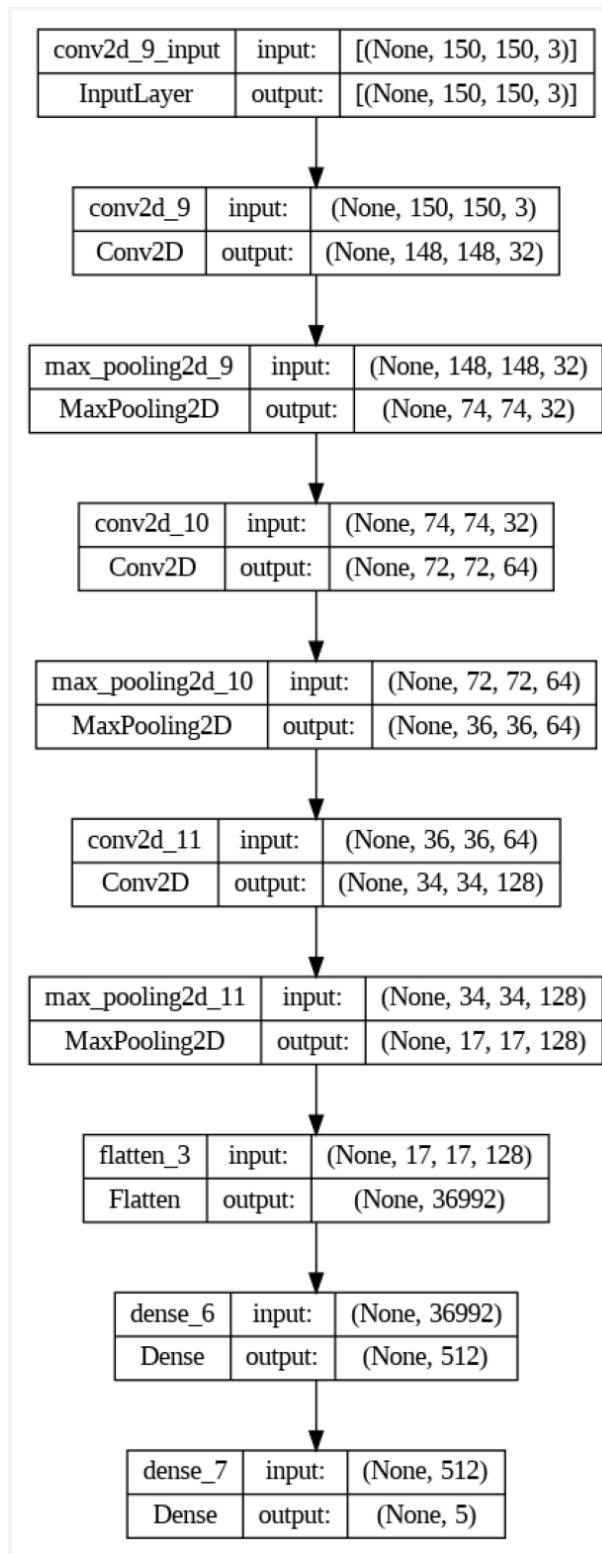
Wynik drugiego trenowania znajduje się w pliku:

[https://github.com/anksunamoona/Uczenie_Maszynowe_Cwiczenie_2
/blob/main/Rozdział%202.1/Rozdział_2_1_2_2.ipynb](https://github.com/anksunamoona/Uczenie_Maszynowe_Cwiczenie_2/blob/main/Rozdział%202.1/Rozdział_2_1_2_2.ipynb)

Wynik trzeciego trenowania znajduje się w pliku:

[https://github.com/anksunamoona/Uczenie_Maszynowe_Cwiczenie_2
blob/main/Rozdział%202.1/Rozdział_2_1_2_3.ipynb](https://github.com/anksunamoona/Uczenie_Maszynowe_Cwiczenie_2/blob/main/Rozdział%202.1/Rozdział_2_1_2_3.ipynb)

2.1.3. Rysunek wygenerowanej sieci



[https://github.com/anksunamoona/Uczenie_Maszynowe_Cwiczeni
e_2/blob/main/Rozdział%202.1/Rozdział_2_1_3.ipynb](https://github.com/anksunamoona/Uczenie_Maszynowe_Cwiczeni
e_2/blob/main/Rozdział%202.1/Rozdział_2_1_3.ipynb)

2.1.4. Sprawdzić wpływ learning rate na uczenie.

Model z wartością learning rate 0.001

https://github.com/anksunamoona/Uczenie_Maszynowe_Cwiczenie_2/blob/main/Rozdział%202.1/Rozdział_2_1_4_1_learning_rate%3D0_001.ipynb

Model z wartością learning rate 0.03

https://github.com/anksunamoona/Uczenie_Maszynowe_Cwiczenie_2/blob/main/Rozdział%202.1/Rozdział_2_1_4_2_learning_rate%3D0_03.ipynb

Model z wartością learning rate 0.3

https://github.com/anksunamoona/Uczenie_Maszynowe_Cwiczenie_2/blob/main/Rozdział%202.1/Rozdział_2_1_4_3_learning_rate%3D0_3.ipynb

2.1.5. Dodać wizualizację 5 losowych obrazów.

https://github.com/anksunamoona/Uczenie_Maszynowe_Cwiczenie_2/blob/main/Rozdział%202.1/Rozdział_2_1_5.ipynb

2.2. Usprawnienie modelu:

Weryfikacja różnych architektur:

I. Model

W celu usprawnienia modelu wprowadzono kilka poprawek do powyższego kodu.

[https://github.com/anksunamoona/Uczenie_Maszynowe_Cwiczenie_2
/blob/main/Rozdział%202.2/Rozdział_2_2_1.ipynb](https://github.com/anksunamoona/Uczenie_Maszynowe_Cwiczenie_2/blob/main/Rozdział%202.2/Rozdział_2_2_1.ipynb)

W danym przypadku została dodana dodatkowa warstwa konwolucyjna. Także zmieniono funkcję aktywacji w ostatniej warstwie, z *softmax* na *sigmoid* gdyż tę funkcję najczęściej wykorzystywane dla problemów klasyfikacji wieloklasowej.

Otrzymane wyniki wskazują że dokładność na zbiorze treningowym wynoszą około 0.54, podczas gdy na zbiorze walidacyjnym wynosi około 0.44. Jest to oznaka overfittingu, czyli model nauczył się dobrze dopasować do danych treningowych, ale nie generalizuje dobrze na nowych danych.

II. Model

W tym modelu zmieniono optymalizator Adam na SGD, w celu weryfikacji czy uda się w ten sposób poprawić wynik. Także w celu rozbudowy i zwiększenia złożoności sieci dodano dodatkowe warstwy ukryte.

[https://github.com/anksunamoona/Uczenie_Maszynowe_Cwiczenie_2
/blob/main/Rozdział%202.2/Rozdział_2_2_2.ipynb](https://github.com/anksunamoona/Uczenie_Maszynowe_Cwiczenie_2/blob/main/Rozdział%202.2/Rozdział_2_2_2.ipynb)

III. Model

W poniższym modelu także zmieniono optymalizator, tym razem wykorzystano Adagrad. Warstwy ukryte także były zmodyfikowane.

[https://github.com/anksunamoona/Uczenie_Maszynowe_Cwiczenie_2
/blob/main/Rozdział%202.2/Rozdział_2_2_3.ipynb](https://github.com/anksunamoona/Uczenie_Maszynowe_Cwiczenie_2/blob/main/Rozdział%202.2/Rozdział_2_2_3.ipynb)

IV. Model

W tym modelu, przywrócona optymalizator Adam w ostatniej warstwie wykorzystano funkcję *softmax*.

[https://github.com/anksunamoona/Uczenie_Maszynowe_Cwiczenie_2/blo
b/main/Rozdział%202.2/Rozdział_2_2_2.ipynb](https://github.com/anksunamoona/Uczenie_Maszynowe_Cwiczenie_2/blob/main/Rozdział%202.2/Rozdział_2_2_2.ipynb)

Na podstawie ostatniego modelu odbywały się następne trenowania sieci

2.2.2. Klasyfikacja z innym modelem

Wynik pierwszego trenowania znajduje się w pliku:

https://github.com/anksunamona/Uczenie_Maszynowe_Cwiczenie_2/blob/main/Rozdział%202.2/Rozdział_2_2_2_1.ipynb

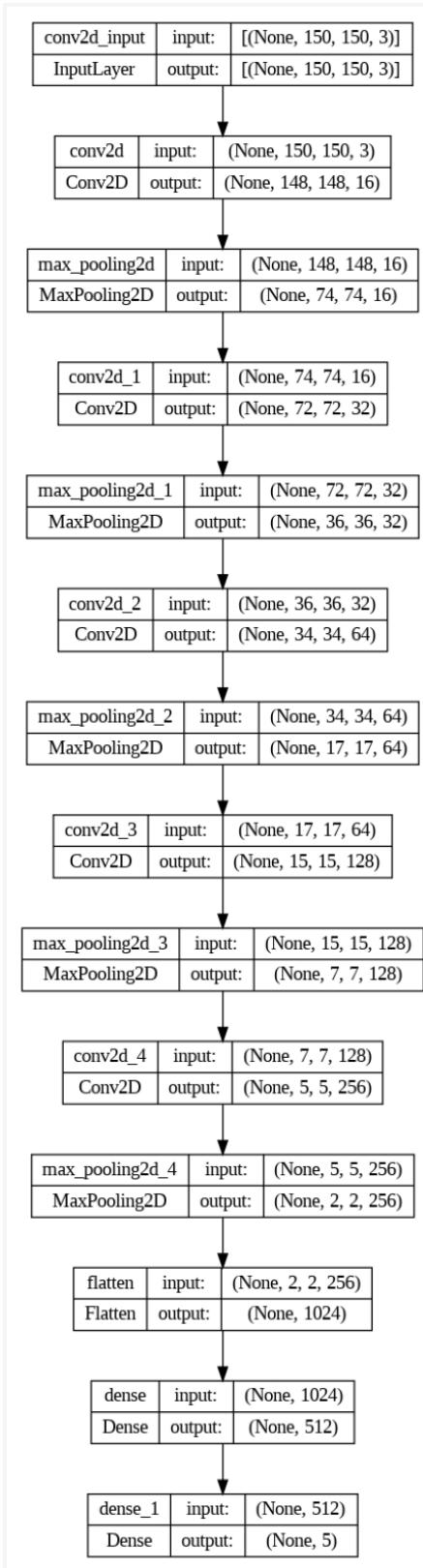
Wynik drugiego trenowania znajduje się w pliku:

https://github.com/anksunamona/Uczenie_Maszynowe_Cwiczenie_2/blob/main/Rozdział%202.2/Rozdział_2_2_2_2.ipynb

Wynik trzeciego trenowania znajduje się w pliku:

https://github.com/anksunamona/Uczenie_Maszynowe_Cwiczenie_2/blob/main/Rozdział%202.2/Rozdział_2_2_2_3.ipynb

2.2.3. Rysunek wygenerowanej sieci



[https://github.com/anksunamoona/Uczenie_Maszynowe_Cwiczenie_2
/blob/main/Rozdzial%202.2/Rozdzial_2_3.ipynb](https://github.com/anksunamoona/Uczenie_Maszynowe_Cwiczenie_2/blob/main/Rozdzial%202.2/Rozdzial_2_3.ipynb)

2.2.4. Sprawdzić wpływ learning rate na uczenie.

Model z wartością learning rate 0.001

https://github.com/anksunamoona/Uczenie_Maszynowe_Cwiczenie_2/blob/main/Rozdział%202.2/Rozdział_2_4_learning_rate%3D0_001.ipynb

Model z wartością learning rate 0.03

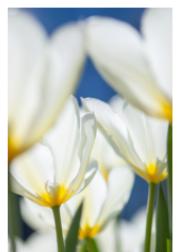
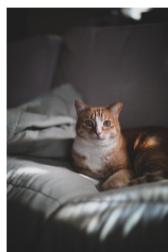
https://github.com/anksunamoona/Uczenie_Maszynowe_Cwiczenie_2/blob/main/Rozdział%202.2/Rozdział_2_4_learning_rate%3D0_03.ipynb

Model z wartością learning rate 0.3

https://github.com/anksunamoona/Uczenie_Maszynowe_Cwiczenie_2/blob/main/Rozdział%202.2/Rozdział_2_4_learning_rate%3D0_3.ipynb

2.2.5. Dodać wizualizację 5 losowych obrazów.

https://github.com/anksunamoona/Uczenie_Maszynowe_Cwiczenie_2/blob/main/Rozdział%202.2/Rozdział_2_2_5_learning_rate%3D0_001.ipynb



2.3. Klasyfikacja z uczeniem przez transfer

1. Zastosowania sieci VGG16 do trenowania z transferem.

W celu zastosowania trenowania z transferem przy użyciu modelu VGG16 do szkolenia, została zmodyfikowana metoda `create_model` w klasie `ImageClassifier`:

https://github.com/anksunamoona/Uczenie_Maszynowe_Cwiczenie_2/blob/main/Rozdział%202.3/2_3_1_1.ipynb

2. Sprawdzić czy można poprawić wynik działania sieci (podać parametry optymalnych ustawień sieci wstępnej). Wykonać trenowanie co najmniej 3 razy.

Wynik pierwszego trenowania znajduje się w pliku:

https://github.com/anksunamoona/Uczenie_Maszynowe_Cwiczenie_2/blob/main/Rozdział%202.3/2_3_1_1.ipynb

Wynik drugiego trenowania znajduje się w pliku:

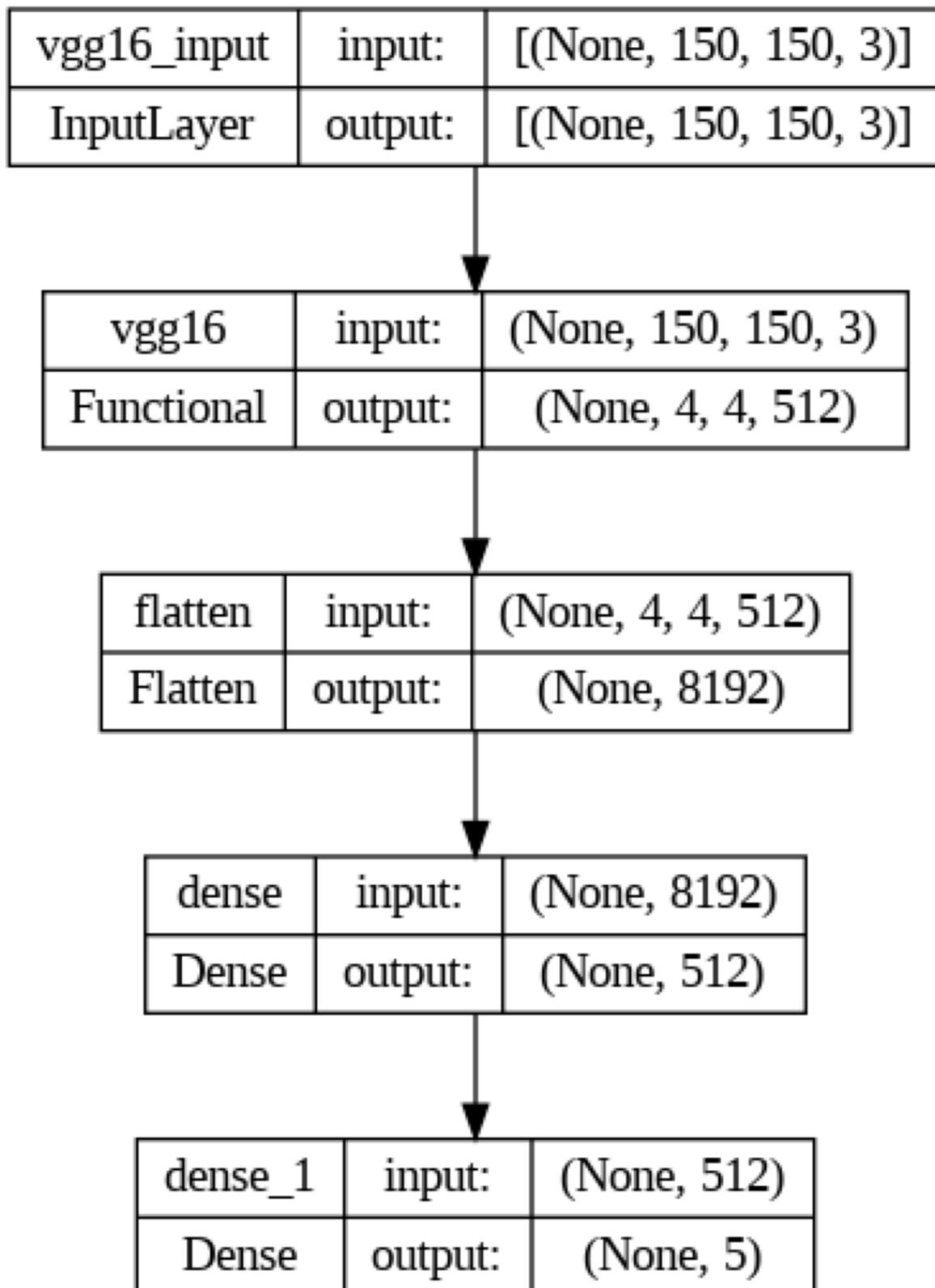
https://github.com/anksunamoona/Uczenie_Maszynowe_Cwiczenie_2/blob/main/Rozdział%202.3/2_3_1_2.ipynb

Wynik trzeciego trenowania znajduje się w pliku:

https://github.com/anksunamoona/Uczenie_Maszynowe_Cwiczenie_2/blob/main/Rozdział%202.3/2_3_1_3.ipynb

3. Narysować strukturę sieci.

https://github.com/anksunamoona/Uczenie_Maszynowe_Cwiczenie_2/blob/main/Rozdzial%202.3/2_3_3.ipynb



4. Sprawdzić wpływ learning rate na uczenie.

Model z wartością learning rate 0.001

https://github.com/anksunamoona/Uczenie_Maszynowe_Cwiczenie_2/blob/main/Rozdział%202.3/2_3_4_learning_rate%3D0_001.ipynb

Model z wartością learning rate 0.03

https://github.com/anksunamoona/Uczenie_Maszynowe_Cwiczenie_2/blob/main/Rozdział%202.3/2_3_4_learning_rate%3D0_03.ipynb

Model z wartością learning rate 0.3

https://github.com/anksunamoona/Uczenie_Maszynowe_Cwiczenie_2/blob/main/Rozdział%202.3/2_3_4_learning_rate%3D0_3.ipynb

5. Dodać wizualizację 5 losowych obrazów.

https://github.com/anksunamoona/Uczenie_Maszynowe_Cwiczenie_2/blob/main/Rozdział%202.3/2_3_5_learning_rate%3D1.ipynb



Odpowiedzi na pytania:

1. Jaki wpływ ma struktura sieci na rozpoznawanie i klasyfikację?

Podczas tworzenia sieci neuronowej, w zależności od funkcji którą sieć będzie wykonywała trzeba dostosować jej strukturę, ponieważ struktura odgrywa znaczącą rolę w trenowaniu sieci. Na podstawie przeprowadzonych badań, można wydzielić kilka punktów. Przykładowo funkcja aktywacji, w zależności od wybieranej funkcji aktywacji możemy spodziewać się lepszych lub gorszych wyników, gdyż jedna funkcja sobie lepiej radzi z problemem klasyfikacji, a inną gorzej.

Także wybranie warstw sieci, przykładowo w badanej sieci wykorzystano typ wartwy konwolucyjnej, gdyż ona lepiej sobie radzi z analizą obrazów. Wartwy konwolucyjne potrafią wykrywać lokalne wzorce i hierarchicznie budować reprezentacje. Innym typem warstw są warstwy rekurencyjne, które są użyteczne w przypadku danych sekwencyjnych, takich jak tekst, gdzie uwzględniają kontekst historyczny.

Istotną rolę także odgrywa ilość warstw, ilość neuronów w warstwach oraz połączenia między nimi, gdyż te dane mają wpływ na otrzymywany przez nas wynik.

2. Jaki wpływ ma zastosowanie warstw dropout na klasyfikację?

Warstwy Dropout są stosowane w celu regularyzacji modelu i zapobiegania nadmiernemu dopasowaniu (*overfitting*). Warstwa Dropout działa poprzez losowe wyłączanie pewnego procentu neuronów w danej warstwie w trakcie treningu. Procent ten jest zwykle określany jako parametr, np. 0,2 oznacza, że 20% neuronów zostanie wyłączonych podczas jednej iteracji treningowej.

3. Jaki wpływ ma szybkość uczenia na klasyfikację?

Szybkość uczenia określa, jak szybko model sieci neuronowej dostosowuje swoje parametry na podstawie błędów predykcji w procesie treningu. Wybór odpowiedniej szybkości uczenia jest kluczowy dla osiągnięcia optymalnych wyników klasyfikacyjnych.

Szybkość uczenia ma bezpośredni wpływ na zdolność modelu do dopasowania się do danych treningowych. W przypadku zbyt wysokiej szybkości uczenia, model może dopasować się zbyt dokładnie do danych treningowych (nadmiernie dopasowanie) i nie będzie w stanie ogólniejszości na nowe dane testowe. Z kolei zbyt niska szybkość uczenia może prowadzić do niedopasowania modelu, czyli braku wystarczającego dopasowania do danych treningowych.

4. Jaki wpływ ma jakość zbioru trenującego na uczenie?

W trenowaniu sieci neuronowej, jakość zbioru ma kluczową rolę w nauce zbioru. Dane te powinny być reprezentatywne, czyli zawierać różnorodne przypadki i scenariusze, które mogą wystąpić w rzeczywistych warunkach. Także dane w zbiorze trenującym nie powinny zawierać szumów, błędów ani nieprawidłowych etykiet. Błędne lub nieprawdziwe dane mogą wprowadzać model w błąd i prowadzić do nieprawidłowych wyników. Ważne także brać pod uwagę wielkość zbioru trenującego, który ma istotny wpływ na zdolność modelu do generalizacji. W ogólności, im większy zbiór trenujący, tym lepsza jest zdolność modelu do uchwycenia złożoności i różnorodności danych. Małe zbiory trenujące mogą prowadzić do przeuczenia, czyli sytuacji, w której model idealnie dopasowuje się do danych trenujących, ale nie radzi sobie dobrze na nowych danych..

5. Czy sieć zawsze będzie dobrze trenować?

Sieć nie zawsze będzie dobrze trenować, na otrzymywane wyniki składa się wiele różnych czynników, jak już wspomniano powyżej. Złe dopasowanie hiperparametrów, lub zanieczyszczony błędymi danymi zbiór trenujący, który zawiera mało danych do nauki i nie

jest reprezentatywny, także może doprowadzić do tego że sieć nie będzie trenowana poprawnie. Także może być zbyt skomplikowany model sieci neuronowej, gdzie będzie duża ilość parametrów i warstw, pojawia się ryzyko spotkania się z problemem *overfitting*.

6. Czy ilość epok miała wpływ na trenowanie obrazów?

W zależności od analizowanej modeli, można zobaczyć że ilość epok może mieć jak pozytywny wpływ na trenowanie sieci, gdzie w ostatniej epoce otrzymamy najlepszy wynik pośród innych, jak również ilość epok, może w ogóle nie mieć żadnego wpływu na trenowanie.

7. Czy uczenie z transferem daje dobre wyniki?

Tak. Jeśli przeanalizujemy wyniki otrzymane po pierwszym trenowaniu, można zauważyć że wartość funkcji straty na zbiorze treningowym i walidacyjnym systematycznie maleje, podczas gdy dokładność rośnie w kolejnych epokach. Początkowo, na początku treningu, wartość funkcji straty na zbiorze treningowym wynosiła 3.4570, a dokładność wynosiła 0.2400. Wraz z kolejnymi epokami, wartość funkcji straty maleje do 0.0448, a dokładność wzrasta do 1.0000. Podobnie, wartość funkcji straty na zbiorze walidacyjnym maleje z 1.8627 do 0.9220, a dokładność wzrasta z 0.4000 do 0.6400.