

Національний технічний університет України
«Київський політехнічний інститут ім. І. Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

Вступ до штучного інтелекту

Лабораторна робота №4
«Нейронні мережі»

Виконав:
студент групи ІО-21
Безщасний Р. Р.
Номер у списку групи: 2
Перевірив:
Кочура Юрій Петрович

Київ 2024р.

Тема: «Нейронні мережі».

Мета: удосконалити інтелектуального агента-автомобіля з попередньої лабораторної роботи – додати йому систему контролю швидкості на основі згорткової нейронної мережі. Отримати практичні навички роботи з нейронними мережами для вирішення задачі класифікації зображень.

Індивідуальне завдання:

- Отримати базові навички роботи з нейронними мережами.
- Отримати досвід вирішення задачі класифікації зображень.
- Удосконалити агента-автомобіля з попередньої лабораторної роботи

Алгоритм роботи агента

Створення агента:

Наповнюємо клас агента необхідними конкретними властивостями та даними, щоб початок, ціль та маршрут були конкретизовані або збережені в пам'яті. Також створюємо методи, що будуть окреслювати його можливості, до яких входять «наступний хід», «рух вгору», «рух вниз», «рух вправо», «рух вліво» та «повернення до попередньої точки».

Алгоритм проходження:

Самі методи окрім як «наступного ходу» просто окреслені для кращого розуміння та проведення паралелей з тим, що вимагається від агента в даній лабораторній роботі. Проте сам метод «наступний хід» наповнений великою кількістю умов, які можуть бути описані дуже просто – агент може йти в будь-якому напрямку та повернутися коли він зайшов в глухий кут, проте він завжди старатиметься спочатку шукати шлях, який приведе його ближче до кінцевої точки. Це досягається за допомогою розставлення пріоритетів напрямків ходу, тобто якщо нинішня координата менша за кінцеву, то краще почати перевірку дороги згори – теж саме повернути з усіма іншими напрямками, але якщо не сходиться напрямком ідеально, то починай перебирати – просто та суворо.

Алгоритм взаємодії з базою:

З самого початку та після кожного ходу, який робить агент він записує всі дані в свою *Базу даних*, що реалізована у вигляді окремого батьківського класу, який «наслідує» агент, відповідно після кожного ходу він використовує властивості цієї бази для

запису пройденого шляху, поточного місцезнаходження, кінцевої цілі та аналізу сусідніх точок до кожної зупинки шляху, який він пройшов. Аналіз та «визначення» сусідніх точок(а точніше можливих шляхів на перехресті) виконується завдяки методу батьківського класу `AgentBase.check_environment()`.

Пояснення взаємодії зі згортковою нейронною мережею:

Більша частина пояснень вже написана у вигляді коментарів, написаних у власний спосіб інтерпретації. Проте, якщо пояснювати послідовно, то:

- 1) Спочатку отримуємо дані для тренування та тестування моделі нейронної мережі(МНМ) з використанням бази даних з картинками цифр MNIST, після чого фіксуємо розмір батчу за допомогою `DataLoader`
- 2) Другим кроком буде формування структури МНМ з її шарами та створення її як об'єкту для подальшого тренування та використання
- 3) Встановлення оптимальних параметрів Оптимізатора та Функції Втрат, враховуючи особливості згорткової МНМ та її відмінності від звичайних Лінійних, Нелінійних та інших типів нейронних мереж, що є основою для тренувального процесу МНМ та виконання `forward pass`
- 4) Виконання самого тренувального процесу та висвітлення результату з втратами та часом тривалості тренувального процесу
- 5) Через мою особисту примху створено необов'язкову частину коду, яка служить прикладом та поясненням для висвітлення подальших результатів, бо можливо через певний «бруд в коді» буде важко зрозуміти чи дійсно я використовую дані отримані завдяки проходженню через модель. Тому в цій частині я просто порівнюю дані, які були надані для перевірки правильності роботи МНМ з отриманими та відповідно відформатованими результатами роботи `model_0`, а також задокументував код для висвітлення картинки, дані якої ми використали для отримання результату формату `int`
- 6) Потім вже йдуть попередні пункти мого коду, що включає в себе створення графу, класи бази Агента та самого Агента, а також висвітлення його дороги, разом із сусідами.
- 7) В класах Агента, а точніше в Базі, я додав декілька властивостей пов'язаних зі швидкістю та інформацією про знаки, а також модифікував метод `check_environment()` з тим, щоб він необхідні інструкції для аналізу і трансформування

картинки з цифрою в дані типу int, завдяки МНМ та записав це у властивість Бази, що називається self.speed

- 8) Потім модифікував код для висвітлення графу з позначками швидкості на пройдених ребрах у вигляді вагів
- 9) Остання частина – це функція зроблена для висвітлення картинки швидкості, яку проаналізувала модель та дані якої були записані на ребрі. Щоб скористатися просто впишіть номер ребра і функція висвітлить, що агент бачив під час проходження цього ребра перед аналізом моделі.

Для отримання доступу до коду напряму, будь ласка, перейдіть за цим посиланням на colab.research.google:

<https://colab.research.google.com/drive/1WAtXelx7NMuRH5BbXVMQTN08pROPxmdM?usp=sharing>

Код виконання завдання з результатами

```
import torch
from torch import nn
import torchvision
from torchvision import datasets, transforms
from torch.utils.data import DataLoader, Subset
from torchvision.transforms import ToTensor
import matplotlib.pyplot as plt
import networkx as nx
from random import choice, random, sample, seed
from tqdm.auto import tqdm
from timeit import default_timer as timer

'''Зібрання необхідних даних для тренування та тестування моделі'''

train_data = datasets.MNIST(
    root="data",
    train=True,
    download=True,
    transform=ToTensor(),
    target_transform=None)

test_data = datasets.MNIST(
    root="data",
    train=False,
    download=True,
    transform=ToTensor())
```

```

transform = transforms.ToTensor()

print(f"Залишено {len(train_data)} прикладів після фільтрації.")
print(f"Залишено {len(test_data)} прикладів після фільтрації.")

data_loader_train = DataLoader(train_data, batch_size=64,
                                shuffle=True)
data_loader_test = DataLoader(test_data, batch_size=64,
                               shuffle=False)

for images, labels in data_loader_train:
    print(f"Розмір батчу: {images.size()}, Мітки: {labels.unique()}")
    break

for images, labels in data_loader_test:
    print(f"Розмір батчу: {images.size()}, Мітки: {labels.unique()}")
    break

train_features, train_labels = next(iter(data_loader_train))

'''Сама модель з її шарами та функцією forward'''

from torch import nn

class CNNModel(nn.Module):
    def __init__(self, input_shape: int, hidden_units: int,
                  output_shape: int):
        super().__init__()
        self.block_1 = nn.Sequential(
            nn.Conv2d(in_channels=input_shape,
                      out_channels=hidden_units,
                      kernel_size=3,
                      stride=1,
                      padding=1),
            nn.ReLU(),
            nn.Conv2d(in_channels=hidden_units,
                      out_channels=hidden_units,
                      kernel_size=3,
                      stride=1,
                      padding=1),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2,
                          stride=2)
        )
        self.block_2 = nn.Sequential(

```

```

        nn.Conv2d(hidden_units, hidden_units, 3, padding=1),
        nn.ReLU(),
        nn.Conv2d(hidden_units, hidden_units, 3, padding=1),
        nn.ReLU(),
        nn.MaxPool2d(2)
    )
    self.classifier = nn.Sequential(
        nn.Flatten(),
        nn.Linear(in_features=hidden_units*7*7,
                  out_features=output_shape)
    )

    def forward(self, x: torch.Tensor):
        x = self.block_1(x)
        x = self.block_2(x)
        x = self.classifier(x)
        return x

torch.manual_seed(42)
model_0 = CNNModel(input_shape=1,
                    hidden_units=10,
                    output_shape=len(train_data.classes))
model_0

'''Додаткові функції для тренування та висвітлення результатів
моделі'''

loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(params=model_0.parameters(), lr=0.01)

def print_train_time(start: float, end: float):
    total_time = end - start
    print(f"Train time: {total_time:.3f} seconds")
    return total_time

def train_step(model: torch.nn.Module,
               data_loader: torch.utils.data.DataLoader,
               loss_fn: torch.nn.Module,
               optimizer: torch.optim.Optimizer):
    train_loss, train_acc = 0, 0
    for batch, (X, y) in enumerate(data_loader):
        y_pred = model(X)
        loss = loss_fn(y_pred, y)
        train_loss += loss
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

```

```

train_loss /= len(data_loader)
print(f"Train loss: {train_loss:.5f}")

def test_step(data_loader: torch.utils.data.DataLoader,
              model: torch.nn.Module,
              loss_fn: torch.nn.Module):
    test_loss, test_acc = 0, 0
    model.eval()
    with torch.inference_mode():
        for X, y in data_loader:
            test_pred = model(X)
            test_loss += loss_fn(test_pred, y)
        test_loss /= len(data_loader)
        print(f"Test loss: {test_loss:.5f}\n")

'''Тренування моделі та висвітлення втрат пі час тренування'''

# torch.manual_seed(42)
y_loss = []
train_time_start_on_gpu = timer()

epochs = 5
for epoch in tqdm(range(epochs)):
    print(f"Epoch: {epoch}\n-----")
    train_step(data_loader=data_loader_train,
               model=model_0,
               loss_fn=loss_fn,
               optimizer=optimizer)

    test_step(data_loader=data_loader_test,
              model=model_0,
              loss_fn=loss_fn)

train_time_end_on_gpu = timer()
total_train_time_model_1 =
print_train_time(start=train_time_start_on_gpu,
                  end=train_time_end_on
_gpu)

plt.plot(list(range(5)), [float(i) for i in y_loss])
plt.title('Функція втрат від епохи')
plt.show();

# seed(42)

```

```

'''Перевірка на правильність роботи моделі, якщо потрібен
фіксований результат, то можна розкоментувати команду seed(42)
Якщо ж необхідно розглянути сам семпл(картинку), що розібрана на
тенсори, то треба розкоментувати та запустити код нижче'''

test_samples = [choice(test_data) for _ in range(3)]
unsqueezed_samples = [sample[0].unsqueeze(dim=0) for sample in
test_samples]
res = []
for i in unsqueezed_samples:
    res.append(model_0(i))
res_prob = torch.stack([torch.softmax(i.squeeze(), dim=0) for i
in res])
# image, label = test_samples[2]
# print(f"Image shape: {image.shape}")
# plt.imshow(image.squeeze()) # image shape is [1, 28, 28]
# (colour channels, height, width)
# print(test_samples[2][1])
# print(res)
# plt.title(label);
pred_classes = res_prob.argmax(dim=1)
pred_classes, [i[1] for i in test_samples]

def create_graph(f, s):
    '''Функція для створення графу'''
    Gr = nx.Graph()
    nodes = [(i, j) for i in range(f)] for j in range(s)]
    edges = []
    for i in range(s - 1):
        for j in range(f - 1):
            edges.append((nodes[i][j], nodes[i][j+1]))
            edges.append((nodes[i][j], nodes[i+1][j]))
            edges.append((nodes[i][f-1], nodes[i+1][f-1]))
    Gr.add_edges_from(edges)
    return Gr

def delete_edges(n, graphs):
    '''Функція для видалення випадкових ребер з графу так,
щоб граф при цьому залишався зв'язним'''
    seq = list(G.edges)
    for i in range(n):
        while seq:
            ran = choice(seq)
            graphs.remove_edge(ran[0], ran[1])
            seq.remove(ran)
            if nx.is_connected(G):
                break

```



```

        graphs.add_edge(ran[0], ran[1])
    return graphs

m = 15
ln1 = 5
ln2 = 5
G = create_graph(ln1, ln2)
G = delete_edges(m, G)
edge_labels = dict()
for i in G.edges:
    edge_labels[i] = 1
node_colors = ['blue' for _ in range(len(G.nodes))]
pos = {node: node for node in G.nodes}
nx.draw(G, pos, with_labels=True, node_color=node_colors)

'''Класи бази Агента, що містить в собі всі властивості та дані,
а також сам клас Агента, що містить всі необхідні методи для
функціонування'''

class AgentBase:
    '''Клас бази даних Агента'''
    def __init__(self, start_pos, end_pos):
        self.new_edges_speed = dict()
        self.new_edges_speed_copy = dict()
        self.current_pos = start_pos
        self.depth = {1: self.current_pos}
        self.start_pos = start_pos
        self.end_pos = end_pos
        self.pos_for_neighbours = {}
        self.check_environment(start_pos)
        self.path = [start_pos]
        self.length = ln1
        self.width = ln2
        self.speed = None
        self.point_for_speed = None
        self.lis = list(G.edges)
        ln = 0
        while len(self.new_edges_speed) != len(self.lis):
            new = choice(test_data)
            if new[1] >= 2:
                self.new_edges_speed.update({self.lis[ln]: new})
                ln += 1
        self.new_edges_speed_copy = self.new_edges_speed.copy()

        for j in range(len(self.lis)):
            self.lis[j] = list(self.lis[j])
            self.lis[j][0] = list(self.lis[j][0])

```

```

        self.lis[j][1] = list(self.lis[j][1])

    def check_environment(self, new_pos):
        if new_pos != self.current_pos:
            speed_sample = None
            if (tuple(self.current_pos), tuple(new_pos)) in
self.new_edges_speed.keys():
                speed_sample =
self.new_edges_speed[(tuple(self.current_pos), tuple(new_pos))]
                self.point = 0
            if (tuple(new_pos), tuple(self.current_pos)) in
self.new_edges_speed.keys():
                speed_sample =
self.new_edges_speed[(tuple(new_pos), tuple(self.current_pos))]
                self.point = 1
            prediction =
torch.softmax(model_0(speed_sample[0].unsqueeze(dim=0)).squeeze()
, dim=0).argmax(dim=0)
            # print(int(prediction))
            # print(speed_sample[1])
            self.speed = int(prediction)
            if self.point:
                self.new_edges_speed.update({(tuple(new_pos),
tuple(self.current_pos)): int(prediction)})
            else:
                self.new_edges_speed.update({(tuple(self.current_
pos), tuple(new_pos)): int(prediction)})
            self.current_pos = new_pos
            self.depth[len(self.depth) + 1] = new_pos
            self.pos_for_neighbours[tuple(new_pos)] =
list(G.neighbors(tuple(new_pos)))
            for a in list(G.neighbors(tuple(new_pos))):
                node_colors[list(G.nodes).index(a)] = 'red'

class Agent(AgentBase):
    '''Клас самого Агента'''
    def __init__(self, start_pos, end_pos):
        super().__init__(start_pos, end_pos)

    def next_move(self):
        if self.current_pos[0] < self.end_pos[0]:
            if ([self.current_pos, [self.current_pos[0] + 1,
self.current_pos[1]]] in self.lis and
                [self.current_pos[0] + 1,
self.current_pos[1]] not in self.path):
                self.step_up()

```

```

        elif (self.current_pos[1] != self.width - 1 and
              ([self.current_pos, [self.current_pos[0],
self.current_pos[1] + 1]] in self.lis and
              [self.current_pos[0], self.current_pos[1] + 1]
not in self.path)):
            self.step_right()

        elif self.current_pos[1] != 0 and (
              [[self.current_pos[0], self.current_pos[1] -
1], self.current_pos] in self.lis and
              [self.current_pos[0], self.current_pos[1] -
1] not in self.path):
            self.step_left()

        elif self.current_pos[0] != 0 and (
              [[self.current_pos[0] - 1,
self.current_pos[1]], self.current_pos] in self.lis and
              [self.current_pos[0] - 1,
self.current_pos[1]] not in self.path):
            self.step_down()

        else:
            self.go_back()

        elif self.current_pos[0] > self.end_pos[0]:
            if ([self.current_pos[0] - 1, self.current_pos[1]],
self.current_pos] in self.lis and
              [self.current_pos[0] - 1,
self.current_pos[1]] not in self.path):
                self.step_down()

        elif self.current_pos[1] != self.width - 1 and (
              [self.current_pos, [self.current_pos[0],
self.current_pos[1] + 1]] in self.lis and
              [self.current_pos[0], self.current_pos[1] +
1] not in self.path):
            self.step_right()

        elif self.current_pos[1] != 0 and (
              [[self.current_pos[0], self.current_pos[1] -
1], self.current_pos] in self.lis and
              [self.current_pos[0], self.current_pos[1] -
1] not in self.path):
            self.step_left()

        elif self.current_pos[0] != self.length - 1 and (
              [self.current_pos, [self.current_pos[0] + 1,
self.current_pos[1]]] in self.lis and

```

```

        [self.current_pos[0] + 1,
self.current_pos[1]] not in self.path):
            self.step_up()

        else:
            self.go_back()

        elif self.current_pos[1] < self.end_pos[1]:
            if ([self.current_pos, [self.current_pos[0],
self.current_pos[1] + 1]] in self.lis and
                [self.current_pos[0], self.current_pos[1] +
1] not in self.path):
                self.step_right()

            elif self.current_pos[0] != 0 and (
                [[self.current_pos[0] - 1,
self.current_pos[1]], self.current_pos] in self.lis and
                [self.current_pos[0] - 1,
self.current_pos[1]] not in self.path):
                self.step_down()

            elif self.current_pos[1] != 0 and (
                [[self.current_pos[0], self.current_pos[1] -
1], self.current_pos] in self.lis and
                [self.current_pos[0], self.current_pos[1] -
1] not in self.path):
                self.step_left()

            elif self.current_pos[0] != self.length - 1 and (
                [self.current_pos, [self.current_pos[0] + 1,
self.current_pos[1]]] in self.lis and
                [self.current_pos[0] + 1,
self.current_pos[1]] not in self.path):
                self.step_up()

        else:
            self.go_back()

        elif self.current_pos[1] > self.end_pos[1]:
            if ([self.current_pos[0], self.current_pos[1] - 1],
self.current_pos] in self.lis and
                [self.current_pos[0], self.current_pos[1] -
1] not in self.path):
                self.step_left()

            elif self.current_pos[0] != 0 and (
                [[self.current_pos[0] - 1,
self.current_pos[1]], self.current_pos] in self.lis and

```

```

        [self.current_pos[0] - 1,
self.current_pos[1]] not in self.path):
        self.step_down()

        elif self.current_pos[1] != self.width - 1 and (
            [self.current_pos, [self.current_pos[0],
self.current_pos[1] + 1]] in self.lis and
            [self.current_pos[0], self.current_pos[1] +
1] not in self.path):
            self.step_right()

        elif self.current_pos[0] != self.length - 1 and (
            [self.current_pos, [self.current_pos[0] + 1,
self.current_pos[1]]] in self.lis and
            [self.current_pos[0] + 1,
self.current_pos[1]] not in self.path):
            self.step_up()

        else:
            self.go_back()

        elif self.current_pos == self.end_pos:
            print(f"Program has finished it's work with success
at the point {self.current_pos}")
            return False

        else:
            print('Wrong start or wrong code')
            return False

    return True

def step_up(self):
    new_pos = [self.current_pos[0] + 1, self.current_pos[1]]
    self.check_environment(new_pos)
    self.path.append(new_pos)

def step_down(self):
    new_pos = [self.current_pos[0] - 1, self.current_pos[1]]
    self.check_environment(new_pos)
    self.path.append(new_pos)

def step_right(self):
    new_pos = [self.current_pos[0], self.current_pos[1] + 1]
    self.check_environment(new_pos)
    self.path.append(new_pos)

```

```

def step_left(self):
    new_pos = [self.current_pos[0], self.current_pos[1] - 1]
    self.check_environment(new_pos)
    self.path.append(new_pos)

def go_back(self):
    self.depth.pop(len(self.depth))
    new_pos = self.depth[len(self.depth)]
    self.path.append(new_pos)
    self.current_pos = new_pos

pos1 = pos.copy()

'''Запуск роботи Агента та виконання всіх необхідних інструкцій
для демонстрації його шляху проходження,
а також висвітлення на його ребрах швидкостей, з якими він
пересувався по цим "ділянкам" (ребрам)'''

agent = Agent([4, 1], [1, 4])
while True:
    if not agent.next_move():
        break

print(f"Agent path: {agent.path}")

for i in range(len(agent.path)):
    node_colors[list(G.nodes).index(tuple(agent.path[i]))] =
'green'

for i in list(G.nodes):
    if node_colors[list(G.nodes).index(i)] == 'blue':
        node_colors.pop(list(G.nodes).index(i))
        G.remove_node(i)

pos = {node: node for node in G.nodes}
plt.figure(figsize=(8, 6))
nx.draw(G, pos, with_labels=True, node_color=node_colors)

edge_labels = dict()
for key, value in agent.new_edges_speed.items():
    if type(value) is int:
        edge_labels.update({key: value})

nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels,
font_size=10)
plt.title("Graph with Weighted Edges", fontsize=15)
plt.show()

```

```

'''Щоб подивитися як виглядає картинка швидкості для певного
ребра оберіть номер проходження ребра у функції та запустіть
її'''

def check_image(order):
    if (tuple(agent.path[order]), tuple(agent.path[order+1])) in
agent.new_edges_speed_copy.keys():
        image, label =
agent.new_edges_speed_copy[(tuple(agent.path[order]),
tuple(agent.path[order+1]))]
        if (tuple(agent.path[order+1]), tuple(agent.path[order])) in
agent.new_edges_speed_copy.keys():
            image, label =
agent.new_edges_speed_copy[(tuple(agent.path[order+1]),
tuple(agent.path[order]))]
        print(f"Image shape: {image.shape}")
        plt.imshow(image.squeeze())
        print((tuple(agent.path[order+1]), tuple(agent.path[order])))
        plt.title(label);

check_image(2)

```

Результати компіляції коду

```

Downloading https://oss-ci-datasets.s3.amazonaws.com/mnist/train-images-idx3-ubyte.gz
Downloading https://oss-ci-datasets.s3.amazonaws.com/mnist/train-images-idx3-ubyte.gz to data/MNIST/raw/train-images-idx3-ubyte.gz
100%|██████████| 9.91M/9.91M [00:00<00:00, 15.9MB/s]
Extracting data/MNIST/raw/train-images-idx3-ubyte.gz to data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz
Failed to download (trying next):
HTTP Error 403: Forbidden

Downloading https://oss-ci-datasets.s3.amazonaws.com/mnist/train-labels-idx1-ubyte.gz
Downloading https://oss-ci-datasets.s3.amazonaws.com/mnist/train-labels-idx1-ubyte.gz to data/MNIST/raw/train-labels-idx1-ubyte.gz
100%|██████████| 28.9k/28.9k [00:00<00:00, 481kB/s]
Extracting data/MNIST/raw/train-labels-idx1-ubyte.gz to data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz
Failed to download (trying next):
HTTP Error 403: Forbidden

Downloading https://oss-ci-datasets.s3.amazonaws.com/mnist/t10k-images-idx3-ubyte.gz
Downloading https://oss-ci-datasets.s3.amazonaws.com/mnist/t10k-images-idx3-ubyte.gz to data/MNIST/raw/t10k-images-idx3-ubyte.gz
100%|██████████| 1.65M/1.65M [00:00<00:00, 4.33MB/s]
Extracting data/MNIST/raw/t10k-images-idx3-ubyte.gz to data/MNIST/raw


Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz
Failed to download (trying next):
HTTP Error 403: Forbidden

Downloading https://oss-ci-datasets.s3.amazonaws.com/mnist/t10k-labels-idx1-ubyte.gz
Downloading https://oss-ci-datasets.s3.amazonaws.com/mnist/t10k-labels-idx1-ubyte.gz to data/MNIST/raw/t10k-labels-idx1-ubyte.gz
100%|██████████| 4.54k/4.54k [00:00<00:00, 3.53MB/s]
Extracting data/MNIST/raw/t10k-labels-idx1-ubyte.gz to data/MNIST/raw



Залишено 60000 прикладів після фільтрації.
Залишено 10000 прикладів після фільтрації.
Розмір батчу: torch.Size([64, 1, 28, 28]), Мітки: tensor([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
Розмір батчу: torch.Size([64, 1, 28, 28]), Мітки: tensor([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

```

```
36
37     def forward(self, x: torch.Tensor):
38         x = self.block_1(x)
39         x = self.block_2(x)
40         x = self.classifier(x)
41         return x
42
43 torch.manual_seed(42)
44 model_0 = CNNModel(input_shape=1,
45                     hidden_units=10,
46                     output_shape=len(train_data.classes))
47 model_0
```

 CNNModel(
 (block_1): Sequential(
 (0): Conv2d(1, 10, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
 (1): ReLU()
 (2): Conv2d(10, 10, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
 (3): ReLU()
 (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
)
 (block_2): Sequential(
 (0): Conv2d(10, 10, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
 (1): ReLU()
 (2): Conv2d(10, 10, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
 (3): ReLU()
 (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
)
 (classifier): Sequential(
 (0): Flatten(start_dim=1, end_dim=-1)
 (1): Linear(in_features=490, out_features=10, bias=True)
)
)

```
17
18 train_time_end_on_gpu = timer()
19 total_train_time_model_1 = print_train_time(start=train_time_start_on_gpu,
20                                              end=train_time_end_on_gpu)
```

 100%  5/5 [04:29<00:00, 53.61s/it]

Epoch: 0

Train loss: 1.84535
Test loss: 0.44711

Epoch: 1

Train loss: 0.31593
Test loss: 0.22731

Epoch: 2

Train loss: 0.17923
Test loss: 0.13411

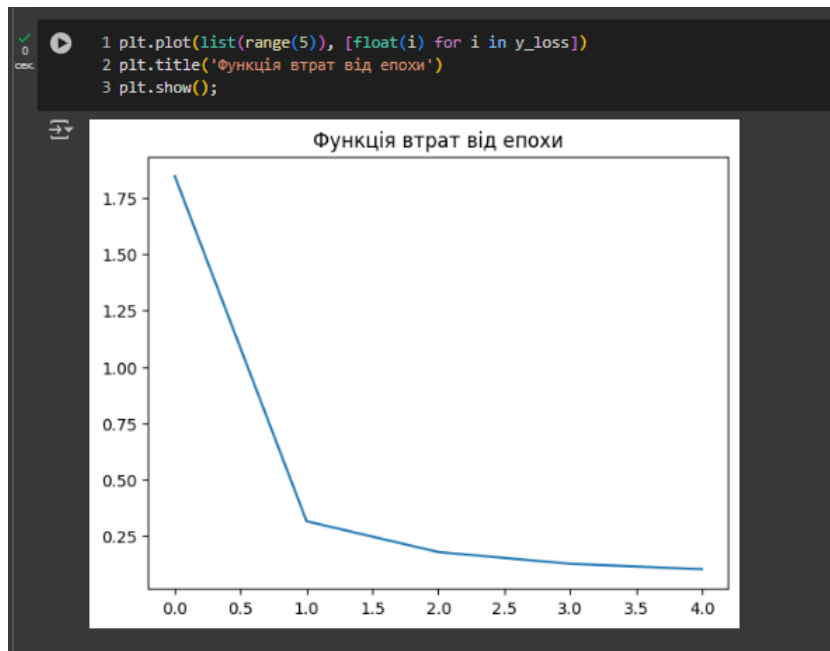
Epoch: 3

Train loss: 0.12736
Test loss: 0.08870

Epoch: 4

Train loss: 0.10337
Test loss: 0.07742

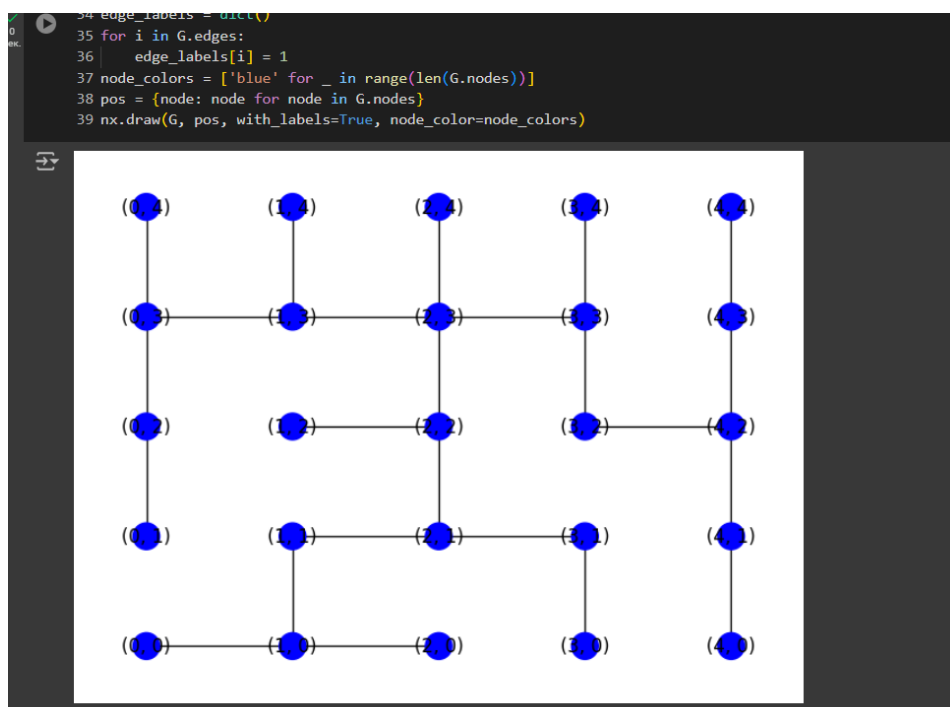
Train time: 269.203 seconds



```
Train time: 269.283 seconds

[6] 1 # seed(42)
2 '''Перевірка на правильність роботи моделі, якщо потрібен фіксований результат, то можна розкоментувати команду seed(42)
3 Якщо ж необхідно розглянути сам семпл(картинку), що розібрана на тензори, то треба розкоментувати та запустити код нижче'''
4
5 test_samples = [choice(test_data) for _ in range(3)]
6 unsqueezed_samples = [sample[0].unsqueeze(dim=0) for sample in test_samples]
7 res = []
8 for i in unsqueezed_samples:
9     res.append(model_0(i))
10 res_prob = torch.stack([torch.softmax(i.squeeze(), dim=0) for i in res])
11 # image, label = test_samples[2]
12 # print(f"Image shape: {image.shape}")
13 # plt.imshow(image.squeeze()) # image shape is [1, 28, 28] (colour channels, height, width)
14 # print(test_samples[2][1])
15 # print(res)
16 # plt.title(label);
17 pred_classes = res_prob.argmax(dim=1)
18 pred_classes, [i[1] for i in test_samples]
```

(tensor([2, 5, 7]), [2, 5, 7])

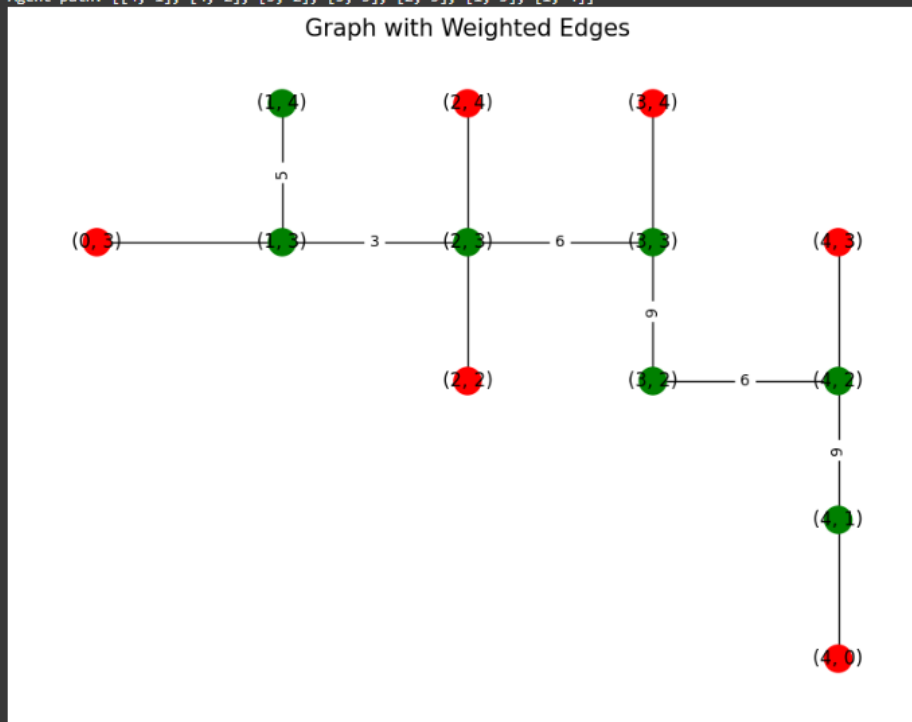


```

22
23 edge_labels = dict()
24 for key, value in agent.new_edges_speed.items():
25     if type(value) is int:
26         edge_labels.update({key: value})
27
28 nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels, font_size=10)
29 plt.title("Graph with Weighted Edges", fontsize=15)
30 plt.show()

```

Program has finished it's work with success at the point [1, 4]
Agent path: [[4, 1], [4, 2], [3, 2], [3, 3], [2, 3], [1, 3], [1, 4]]

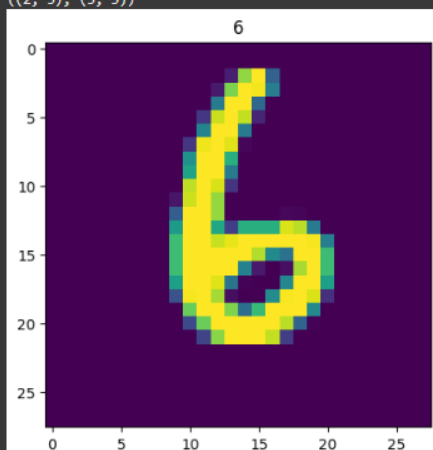


```

[11] 1 '''Щоб подивитися як виглядає картинка швидкості для певного ребра оберіть номер проходження ребра у функції та запустіть її'''
2
3 def check_image(order):
4     if (tuple(agent.path[order]), tuple(agent.path[order+1])) in agent.new_edges_speed_copy.keys():
5         image, label = agent.new_edges_speed_copy[(tuple(agent.path[order]), tuple(agent.path[order+1]))]
6     if (tuple(agent.path[order+1]), tuple(agent.path[order])) in agent.new_edges_speed_copy.keys():
7         image, label = agent.new_edges_speed_copy[(tuple(agent.path[order+1]), tuple(agent.path[order]))]
8     print(f"Image shape: {image.shape}")
9     plt.imshow(image.squeeze())
10    print((tuple(agent.path[order+1]), tuple(agent.path[order])))
11    plt.title(label)
12
13 check_image(3)

```

Image shape: torch.Size([1, 28, 28])
((2, 3), (3, 3))



Висновок

На даній лабораторній роботі я удосконалив інтелектуального агента-автомобіля з попередньої лабораторної роботи, а саме додав йому систему контролю швидкості на основі згорткової нейронної мережі. Отримав практичні навички роботи з нейронними мережами для вирішення задач класифікації зображень та вдало імплементував модель нейронної мережі для аналізу зображень з цифрами для переформування у зручний для висвітлення на графі фортам. Всі необхідні пояснення записані вище для кожного пункту лабораторної роботи + в самому коді є документація до кожної частини для спрощення читабельності коду.

Для отримання доступу до коду напряму, будь ласка, перейдіть за цим посиланням на colab.research.google:

<https://colab.research.google.com/drive/1WAtXelx7NMuRH5BbXVMQTN08pROPxmdM?usp=sharing>