Министерство науки и высшего образования РФ
ФГБОУ ВО «Тверской государственный университет»
Математический факультет
Направление 02.04.01 Математика и компьютерные науки Профиль
«Математическое и компьютерное моделирование»

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

“Объяснимые модели искусственного интеллекта для задач анализа траекторий мелких лабораторных животных в поведенческих тестах”

Автор:
Ширмер Александр Андреевич
_____
(подпись)

Научный руководитель:
к. ф.-м. н. Стрельцова О.И.
_____
(подпись)

Допущен(а) к защите:
_____2025 г.

Руководитель ООП:
д. ф.-м. н. Цветков В.П.

Тверь 2025

Оглавление

Введение

В условиях стремительного развития искусственного интеллекта (ИИ) его интеграция в критически важные сферы, такие как медицина, финансы и право, порождает необходимость обеспечения прозрачности алгоритмических решений. Непрозрачность современных моделей ИИ, часто называемая проблемой "чёрного ящика", ограничивает доверие пользователей и создаёт риски для безопасности и этики [1].

Проблема «черного ящика» возникает, когда нейросетевая модель выдает точные предсказания, но нет понимая, почему модель предсказала правильный ответ. Проблема чаще всего касается сложных моделей, состоящие из миллионов параметров, внутренние процессы которых трудно отследить, или модели со сложным математическим аппаратом. Например, если система отказывает человеку в кредите или рекомендует лечение, важно знать, на основании каких данных и правил было принято решение [2].

Для анализа траектории мелких лабораторных животных в поведенческих тест системах часто применяются сверточные нейронные сети, которые выполняют задачи: получение траектории мелких лабораторных животных из экспериментальных видеоданных , классификации траекторий. Анализ полученных результатов применяются для решения разных задач в биологии. Например: анализ влияния болезней, медицинских препаратов и практик, для выявления паттернов поведения у лабораторных животных [3]. Данные наблюдения влияют на поиск и оценку аналогичных проблем в человеческом теле, что позитивно сказывается на качество человеческих жизней.

Объяснимый ИИ (Англ. *Explainable Artificial Intelligence,* XAI) -это направление, которое стремится сделать алгоритмы машинного обучения прозрачными, интерпретируемыми и понятными для человека.

Актуальность этой темы растёт вместе с требованиями к этике, безопасности и доверию к ИИ.

Целью дипломной работы является разработка объяснимых моделей искусственного интеллекта для задач трекинга мелких лабораторных животных в поведенческой тест-системе “Водный лабиринт Морриса”.

Задачи

- Изучение основ нейросетевого подхода, различных типов нейросетевых архитектур, таких как: многослойный перцептрон, сверточные нейронные сети, методы их обучения (метод обратного распространения ошибки и его модификации)
- Изучение методов и технологий объяснимого искусственного интеллекта
- Апробация методов и технологий *XAI* для тестовых задач
- Разработка модели XAI для задачи классификации траектории мелких лабораторных животных в поведенческой тест-системе «Водный лабиринт Морриса»

1. Основы нейросетевого подхода

1.1 Отношение машинного обучения к искусственному интеллекту

ИИ — это самая широкая концепция. Это область, которая стремится создать системы или машины, способные имитировать человеческую способность к мышлению, принятию решений и решению задач. ИИ охватывает всё, от простых правил до сложных систем, которые учатся сами. Машинное обучение — это подмножество (рис. 1) ИИ, где системы улучшают свои способности к выполнению задач, анализируя данные и находя в них закономерности, не будучи явно запрограммированными для каждой задачи. Глубокое обучение — это специализированная часть машинного обучения, которая использует многослойные нейронные сети для обработки данных [4].

Рисунок 1. Взаимосвязь AI, ML и DP (Источник [5])

1.2 Нейронные сети

Нейронная сеть – это математические модели, вдохновленные устройством человеческого мозга, которые используются для решения задач машинного обучения, таких как классификация, прогнозирование или распознавание образов. Человеческий мозг состоит из множества нейронов, которые получают, обрабатывают и передают информацию из окружающей среды для совершения соответствующей реакции [5].

Нейронная сеть состоит из следующих слоев:

Входной слой: принимает входные данные, в виде тензора. Обычно входные данные достаточно разнообразны и могут быть не похожи друг на друга, перед подачей данных во входной слой, данные нормализуются до одинаково вида.

Скрытые слои: обрабатывает данные, находя сложные зависимости. В зависимости от выполняемой задачи, данные слои могут быть сверточными, полносвязными, рекуррентными и другие. Количество скрытых слоев определяет «глубину» сети.

Выходной слой: выдает результат, в зависимости от задачи результатом может быть тензор, значение или флаг.

Каждый слой состоит из нейронов, каждый нейрон принимает входные данные, применяет к ним веса (важность каждого входа), добавляет смещение (*bias*) и пропускает через функцию активации (например, ReLU или сигмоиду). Функции активации (рис. 2, 3) добавляют нелинейность модели, позволяя решать сложные задачи, например: распознавание объектов на фото требует сложных, нелинейных закономерностей, которые линейные функции (типа $y = ax + b$) не могут уловить [6]. Некоторые функции активации ограничивают значения нейронов, что упрощает обучение, например: активация ReLU обнуляет значения ниже нуля, сигмоида преобразует значения в интервал [0, 1].

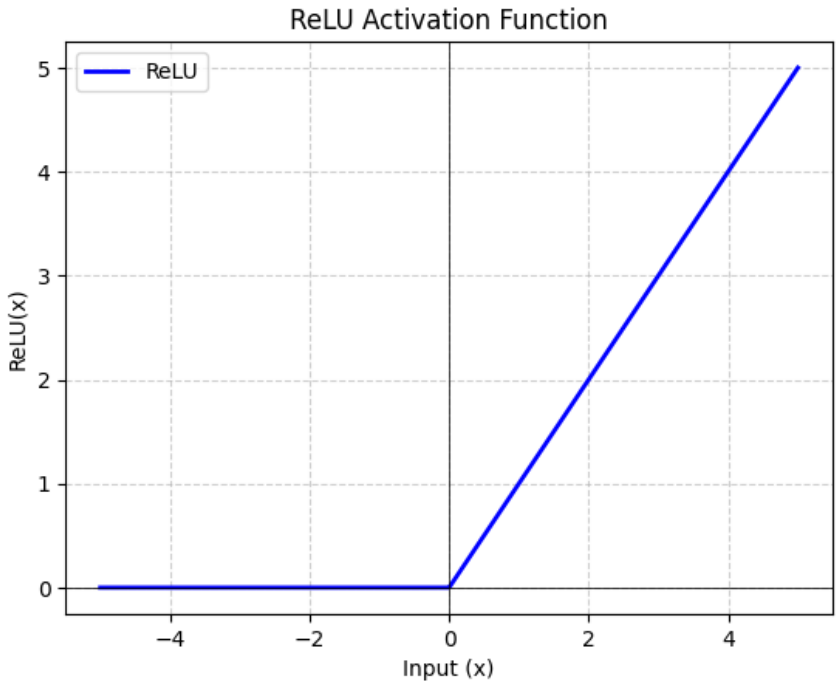

Рисунок 2. Функция активации ReLU

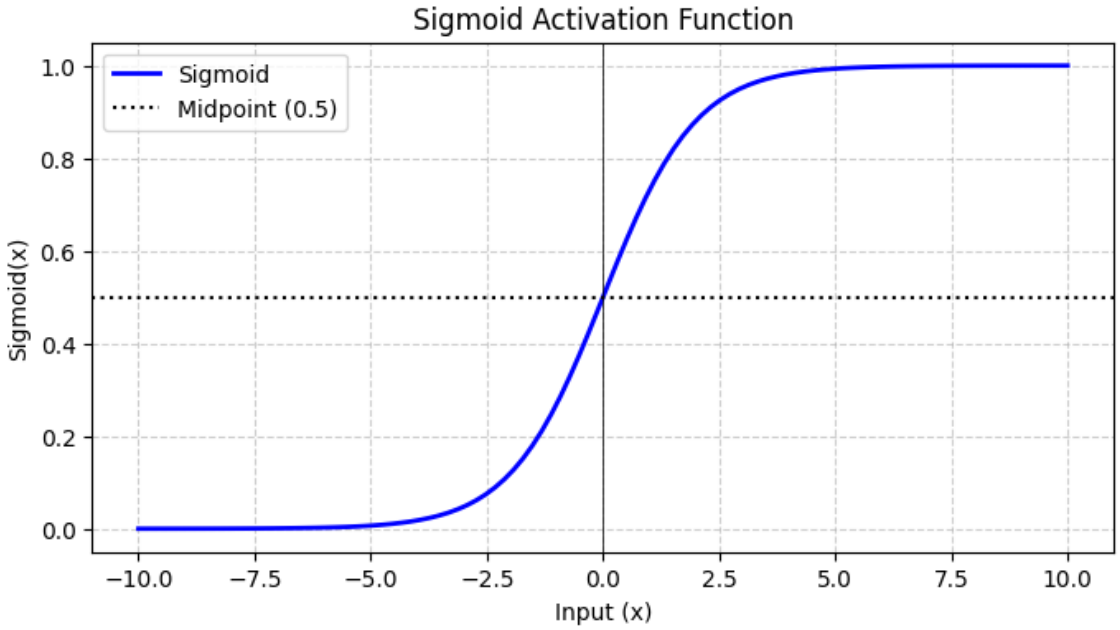

Рисунок 3. Функция активации сигмоида

1.3 Многослойный перцептрон

Многослойный перцептрон -это тип искусственной нейронной сети, который используется для решения задач классификации и регрессии. Это одна из простейших архитектур глубокого обучения, которая имитирует работу биологических нейронов, обрабатывая входные данные и выдавая

предсказания. Многослойный перцептрон подходит для задач, где нужно найти сложные зависимости в данных, например, распознавание рукописных цифр или прогнозирование цен [6]. Перцептрон состоит из входного слоя, который принимает данные, скрытых слоев, где происходит основная обработка, и выходного слоя, который выдает результат.

Обучение нейронной сети происходит по принципу обратного распространения, где модель сначала делает прогноз используя случайные весы. Далее вычисляется функция потерь и минимизируя ее с помощью оптимизатора.

1.4 Свертовочные нейронные сети

Сверточные нейронные сети (*Convolutional Neural Networks*, CNN) — это класс глубоких нейронных сетей, которые особенно эффективны для задач обработки изображений и видео.

Архитектура сверточной нейронной сети немного отличается от вышеупомянутых нейронных сетей. CNN можно разделить на 2 блока: сверточный блок, содержащий слои свертки, которые выделяют признаки из набора данных, и полносвязный блок, содержащий полносвязные слои, которые классифицирует выделенные признаки слоев свертки [5].

Сверточные слои: сверточный слой применяет операцию свертки (рис. 4), которая заключается в выделении таких признаков, как края, текстуры или формы. Например, один фильтр может реагировать на вертикальные линии, а другой — на углы. Фильтр или ядро свертки – матрица чисел, обычно размером 3x3 или 5x5, которая поочередно применяется к каждому значению входного тензора, получается число –

значение в выходной карте признаков.

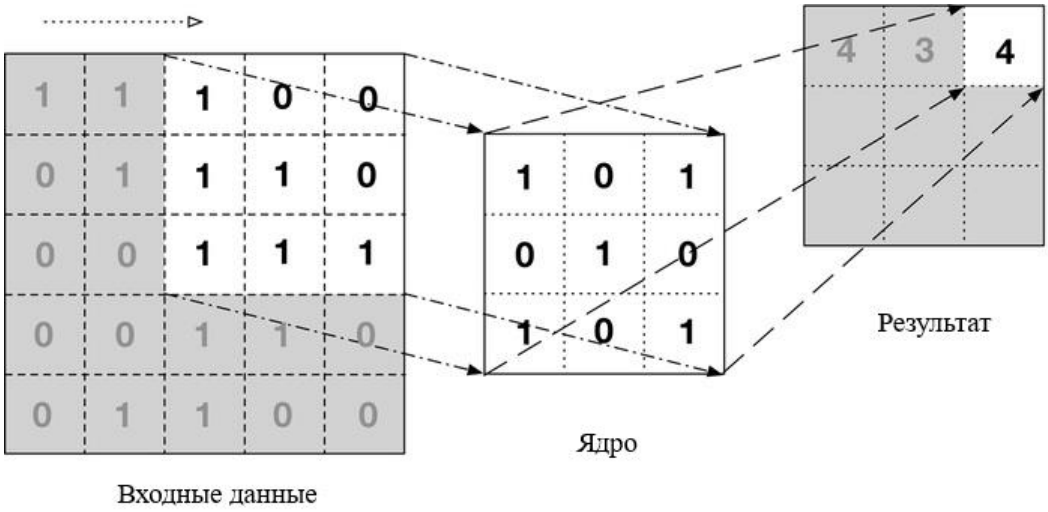

Рисунок 4. Пример операции свертки

Полносвязные слои в сверточных нейронных сетях (CNN) — это важный компонент, который обычно используется на заключительных этапах сети для объединения признаков, извлечённых сверточными слоями, и принятия окончательного решения, например, классификации.

1.4.1 Обучение нейронной сети

Количество нейронов можно достигать достаточно большого количества, у каждого из которых есть свой вес и смещение, которые необходимо подобрать и протестировать для достижения минимальной ошибки. Чаще всего обучение происходит в рамках «обучение с учителем», модель обучается на тестовом, множестве данных с метками, например: изображение с кошкой подписано как «кошка» или «класс1». Данные разделены на обучающее множество, и тестовое множество. Это сделано для проверки сети, имитируя новые данные. Если сеть показывает хорошую точность, то значит обучение прошло успешно [7]. Обучение происходит в несколько шагов, много раз проходя через обучающие данные.

Первый этап обучения – это прямое распространение, данные проходят через сеть от входного слоя к выходному, через скрытые слои. Изначально все веса и смещения случайны, если не задано иначе.

Полученный результат сравнивается с меткой с помощью функции потерь, которая измеряет ошибку. В зависимости от задачи функция потерь будет разной, например: кросс-энтропия для задачи классификации, среднеквадратичная ошибка для задач регрессии. Чем больше ошибка – тем хуже сеть справляется с задачей.

Для настройки параметров сети и минимизации функции потерь применяется алгоритм оптимизации, такой алгоритм называется оптимизатором. Градиент, или производная функции потерь, указывает направление и величину минимума функции. Если при изменении весов и смещений значение функции уменьшилось – то происходит замена весов и смещений.

Измененные весов происходит от выходного слоя к входному соответственно распределяя веса.

Данный процесс повторяется некоторое количество раз, называемое эпохами (Англ. *epoch*), за одну эпоху проходя все обучающее множество. Перед обновлением весов обрабатывает некоторое количество входных данных называемым размером батча (Англ. *batch size*).

1.5 Реализация сверточной нейронной сети

1.5.1 Задача и данные

Для реализации сверточной нейронной была взята задача классификации рукописных чисел на классы, от 0 до 9. Набор данных состоит из 70000 изображений в градациях серого, размером 28x28 пикселей (рис.5) [8].

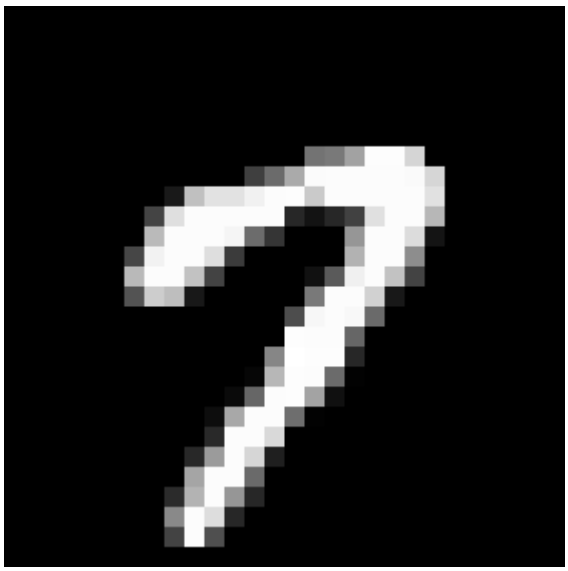

Рисунок 5. Изображение рукописной цифры 7 из набора данных MNIST

Набор поделен на 2 части: часть для обучения имеет 60000 изображений, часть для проверки 10000. Таким образом мы имитируем новые данные, к которым у нейросети не было доступа во время обучения для предотвращения переобучения, а также независимой проверки качества [5].

1.5.2 Программная реализация

Сверточная нейронная сеть была разработана с помощью фреймоврка PyTorch. Для загрузки изображений была использована функция Torch.utils.data.DataLoader, в которой train = true – используемая часть, transforms.compose([transforms.ToTensor()]) – преобразование данных из изображений в тензор PyTorch, shuffle = true – случайный порядок данных. Для обучающей части используется train_loader, для тестовой test_loader. Батч – количесво передаваемого объема данных, в данном случае количество изображений и меток MNIST.

Листинг 1.1 подготовка данных

```
train_loader = torch.utils.data.DataLoader(
    datasets.MNIST(
        "mnist_data",
        train=True,
        download=True,
        transform=transforms.Compose([transforms.ToTensor()]),
    ),
```

```
    batch_size=batch_size,
    shuffle=True,
)

test_loader = torch.utils.data.DataLoader(
    datasets.MNIST("mnist_data", train=False,
transform=transforms.Compose([transforms.ToTensor()])),
    batch_size=batch_size,
    shuffle=True,
)
```

Архитектура нейронной сети состоит из двух основных блоков: сверточного блока для извлечения признаков и полносвязного блока для классификации. Сверточный блок состоит из двух слоев свертки Conv2d, а также операции MaxPooling2d, функции активации ReLU, а также Dropout – метод случайного отключения признаков [9].

Первый сверточный слой принимает на вход изображение MNIST, к которому применяется операция свертки с размером ядра 3x3, количество ядер, применяемое к данному слою равно 10 что изменит размерность тензора с (28, 28, 1) до (26, 26, 10). *MaxPool2d* – уменьшит размерность изображения в 2 раза с (26, 26, 10) до (13, 13, 10), а активация ReLU обнулит отрицательные значения увеличит положительные. Результатом данного слоя – это тензор с размерностями (13, 13, 10).

Второй сверточный слой является копией первого, за исключением размеров входного тензора. После применения каждой из операций размер тензора измениться с (13, 13, 10) до (11, 11, 20) до (5, 5, 20) [10].

В данном случае полносвязные слои – являются многослойным перцептроном, с двумя слоями. Результирующий тензором будет вероятностное распределение, который укажет на предсказанное значение.

Первый полносвязный слой принимает на вход карту признаков, тензор размером (5*5*20, 50), где 5*5*20 – карта признаков размещенная в одномерном тензоре, а 50 - предложение надо закончить.

Второй полносвязный слой (50, 10) и после второго линейного преобразования тензор (50, 10) преобразуется в (10), где каждый из элементов – это значение, которое нейронная сеть присвоила каждому из

классов, что и является ответом нейронной сети. Чем выше значение, тем выше вероятность, что изображение, поданное на входе, принадлежит к данному классу. Логистическая функция Softmax преобразует значения тензора в виде вещественного числа в интервале [0, 1], для более легкого оценивания.

Метод forward – метод PyTorch, который определяет, в каком порядке входные данные проходят через нейронную сеть. В данном случае входные данные проходят через: сверточные слои, преобразование в тензор (батч, 500), полносвязные слои.

Листинг 1.2 архитектура сверточной нейронной сети

```
Class Net(nn.Module):
    def __init__(self):
        super().__init__()

        self.conv_layers = nn.Sequential(
            nn.Conv2d(1, 10, kernel_size=3),
            nn.MaxPool2d(2),
            nn.ReLU(),
            nn.Conv2d(10, 20, kernel_size=3),
            nn.Dropout(),
            nn.MaxPool2d(2),
            nn.ReLU(),
        )
        self.fc_layers = nn.Sequential(
            nn.Linear(500, 50),
            nn.ReLU(),
            nn.Dropout(),
            nn.Linear(50, 10),
            nn.Softmax(dim=1),
        )

    def forward(self, x):
        x = self.conv_layers(x)
        x = x.view(-1, 500)
        x = self.fc_layers(x)
        return x
```

Для того, чтобы построенная нейронная сеть вывела правильный ответ, после того как на вход будет подано изображение, сеть необходимо обучить. Для обучения был задействован процессор, размер батча = 128, количество эпох = 8. Эпохи – это количество циклов обучения. Оптимизатор

SDG – оптимизатор, используемый для минимизации функции потерь. Первый шаг в обучении модели, output = model(data) – это классификация изображений, со случайными весами. После получении ответа, функция loss = F.nll.loss(output.log(), target) вычислит потерю между предсказанием и меткой изображения. Обратное распространение loss.backward() – функция PyTorch для вычисление градиентов. Optimizer.step() – обновляет веса модели на основе полученных градиентов.

Для минимизации функции потерь был использован оптимизатор SGD - это итеративный метод оптимизации, используемый для минимизации функции потерь. В отличие от обычного градиентного спуска (GD), который вычисляет градиент по всему набору данных, SGD оценивает градиент на отдельном подвыборке данных [10].

После обучения в первой эпохе данные тестируются на тестовом множестве. Output = model.data() – предсказание, которое делает модель. Test_loss += (output.log(), target) – накапливает ошибки. Pred = output.max() – функция выводит индекс с максимальной вероятностью. Correct += pred.eq(target.view as(pred)).sum.item() – количество верных предсказаний.

Листинг 1.3 реализация метода обучения нейронной сети

```
batch_size = 128
num_epochs = 8
device = torch.device("cpu")


def train(model, device, train_loader, optimizer, epoch):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()
        output = model(data)
        loss = F.nll_loss(output.log(), target)
        loss.backward()
        optimizer.step()
        if batch_idx % 100 == 0:
            print(
                f"Train Epoch: {epoch} [{batch_idx *
len(data)}/{len(train_loader.dataset)}"
                f" ({100.0 * batch_idx / len(train_loader):.0f}%)]"
```

```
                f"\tLoss: {loss.item():.6f}"
            )

def test(model, device, test_loader):
    model.eval()
    test_loss = 0
    correct = 0
    with torch.no_grad():
        for data, target in test_loader:
            data, target = data.to(device), target.to(device)
            output = model(data)
            test_loss += F.nll_loss(output.log(), target).item()  # sum up
batch loss
            pred = output.max(1, keepdim=True)[1]  # get the index of the
max log-probability
            correct += pred.eq(target.view_as(pred)).sum().item()

    test_loss /= len(test_loader.dataset)
    print(
        f"\nTest set: Average loss: {test_loss:.4f},"
        f" Accuracy: {correct}/{len(test_loader.dataset)}"
        f" ({100.0 * correct / len(test_loader.dataset):.0f}%)\n"
    )

model = Net().to(device)
optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.5)
```

Обучение и тестирование происходит каждую эпоху, тем самым накапливая информацию о верно предсказанных ответов в каждой эпохе.

Для оценки обучения нейронной сети были выведены следующие графики. График потерь (рис. 6), на котором изображено количество потерь в каждой эпохе. Первоначально процент ошибок большой, но в последующем обучении среднее количество ошибок падает и достигает 0.0011. Потери в наборе данных test начали расти ближе к концу обучения, что указывает на небольшое переобучение модели.

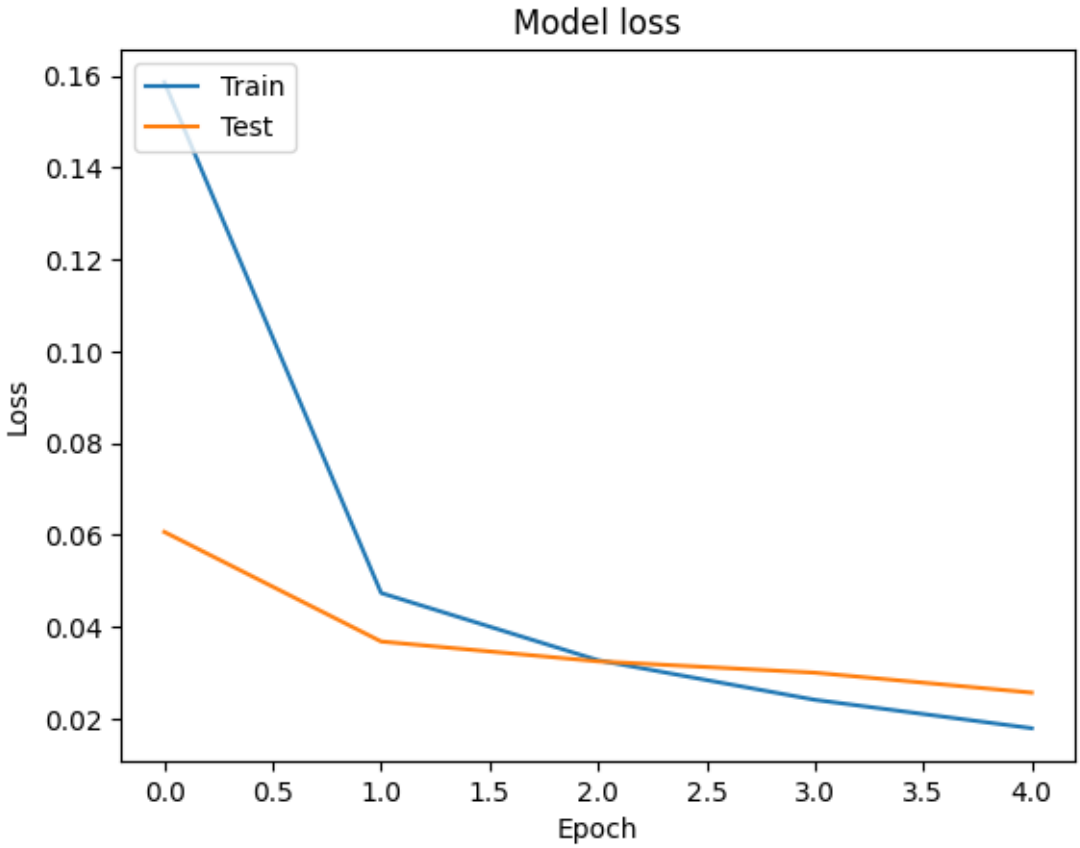

Рисунок 6. Потери на этапах обучения

Метрика точности (рис. 7) показывает отношение верных предсказаний от всех предсказаний, которая достигает 98,7%. И также как и метрика loss указывает на незначительное переобучение.

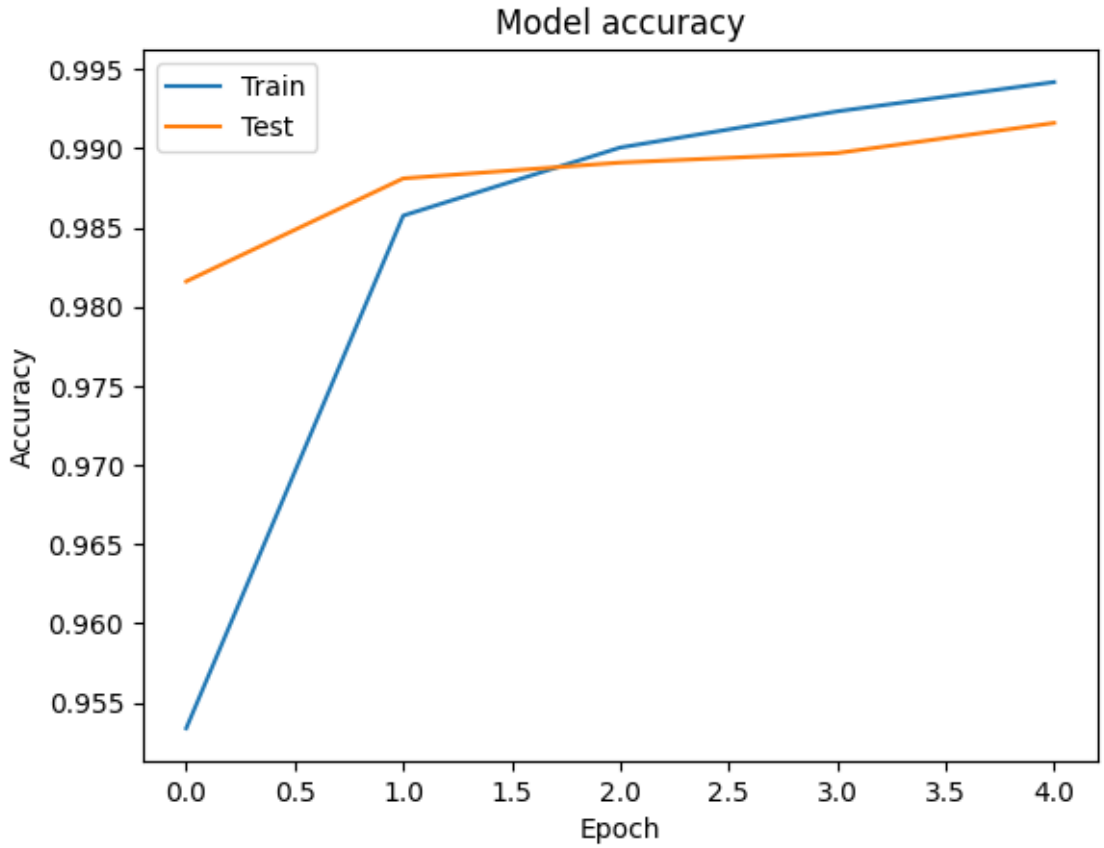

Рисунок 7. Точность на этапах обучения и проверки

После всех проделанных шагов финальная проверка – вывод результатов классификации (рис. 8), как видно на рисунке все изображения были успешно предсказаны.

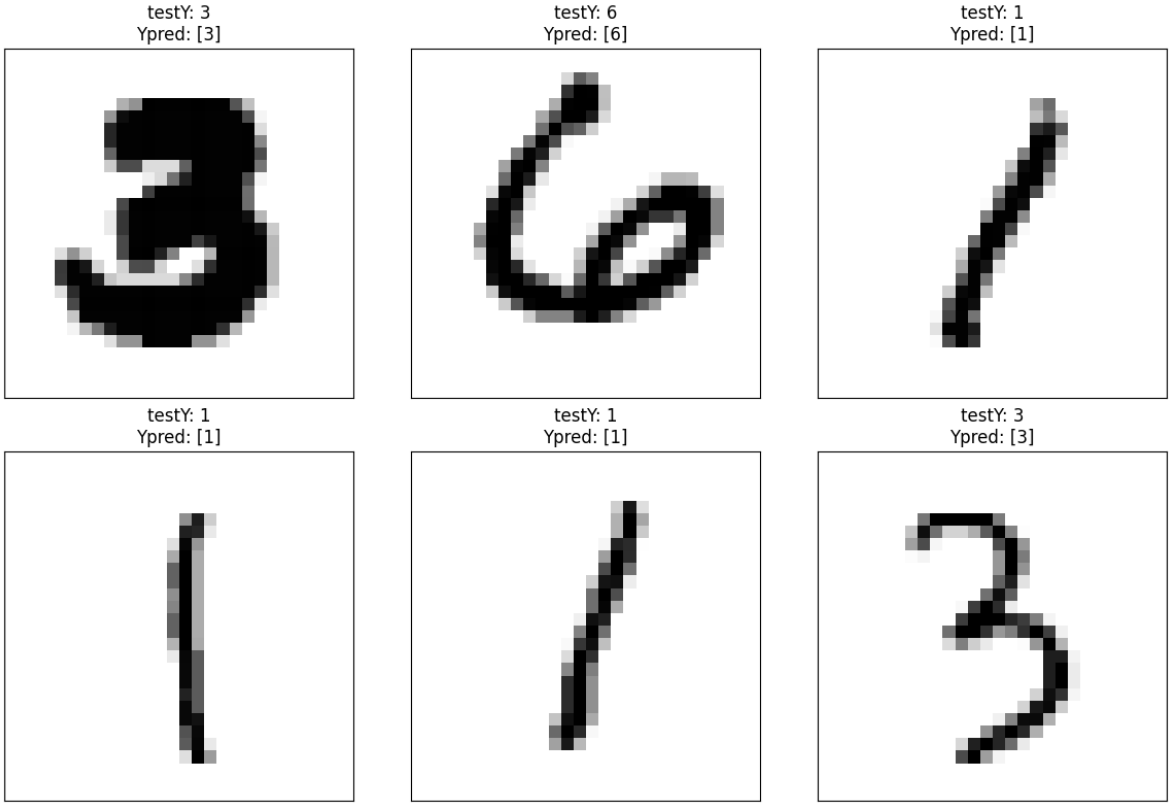

Рисунок 8. Пример классификации изображений MNIST

2 Объяснимые модели искусственного интеллекта

Объяснимый ИИ — это подходы и методы, которые помогают понять, как и почему модели искусственного интеллекта (особенно сложные, вроде нейронных сетей) принимают те или иные решения. Если обычный ИИ — это "чёрный ящик", который выдаёт ответ, но не объясняет как он к нему пришёл, то XAI стремится сделать этот процесс прозрачным [2].

В зависимости от поставленной задачи и используемой модели объяснимый искусственный интеллект можно категоризировать по следующим признакам.

По агностичность: применяемые инструменты и технологии объяснимого ИИ можно разделить на специфические модели, которые созданы для малого круга задач и агностические, применимы к разнообразным задачам.

По масштабу: в зависимости от выбора технологии и решаемой задачи объяснения могут быть: глобальными, объясняющие целиком модель, или локальными, объясняющие конкретный экземпляр модели.

Локальные результаты модели объяснимого ИИ укажут на данные, которые приблизили результат нейросетевой модели к предсказанному экземпляру. Совокупность объяснений локальных результатов, в случае если модель XAI выводит только локальные результаты, поможет объяснить глобальные параметры нейросетевой модели, такие как: сходимость обучения, влияние шума на результат, выявление в смещении данных, в случаях если модель игнорирует один из параметров.

2.1 Подходы объяснимого искусственного интеллекта

2.1.1 Значения Шепли

Одна из технологий объяснимого ИИ – SHAP основанная на значениях Шепли. Значение Шепли — это метод, предложенный Ллойдом Шепли в 1953 году для решения задачи распределения вклада между параметрами стремящимся к общей цели [4]. Значение Шепли определяет,

как справедливо распределить общий выигрыш (или вклад) между признаками коалиции совместной деятельности, учитывая их индивидуальный вклад в результат. Оно основано на идее, что каждый признак коалиции должен получить долю, пропорциональную его вкладу в общий результат [17]. Ценность Шепли оценивается следующим образом:

Есть группа признаков. Мы смотрим на все возможные способы, которыми признаки могли бы объединиться.

Предположим, что признаки присоединяются к коалиции один за другим в разном порядке. Для каждого порядка необходимо посчитать, насколько увеличивается общий результат, когда добавляется новый признак.

Для каждого признак мы считаем, насколько в среднем он увеличивает результат, учитывая все возможные порядки присоединения.

Если у нас есть n признаков, ценность Шепли для признака i определяется так:

$$\varphi_i(\mathrm{v}) = \sum_{S \subset F \setminus \{i\}} \left(\frac{|S|!\,(|N| - |S| - 1)!}{|N|!} \right) \left(v\big((S\ U\ \{i\}) - v(S)\big) \right)$$

N - множество всех признаков.

S - подмножество признаков без признака i.

$|S|$ - размер подмножества S.

$v(s)$ - ценность (результат), которую создаёт коалиция S.

|S| - размер подмножества

Формулу можно разделить на 2 части: $\left(\frac{|S|!(|N|-|S|-1)!}{|N|!}\right)$ – вес, который учитывает количество возможных перестановок, $\left(v\big((S\ U\ \{i\}) - v(S)\big)\right)$ – вклад признака для коалиции S.

2.1.2 Библиотека SHAP

Для задач, связанных с машинным обучением, часто используется приближения, которые оптимизируют расчеты, так как при наличии n

признаков нужно рассмотреть 2n коалиций, что становиться вычислительно дорого при большом n.

Для оптимизации времени работы программы библиотека SHAP аппроксимирует значения Шепли.

Пользователь предоставляет модель глубокого обучения и фоновый набор данных (*background dataset*). Фоновый набор — это выборка данных, которая представляет "типичное" распределение входных данных. Обычно это небольшой поднабор обучающих данных. Фоновый набор используется для определения "базового" предсказания модели, когда признаки отсутствуют или заменены "нейтральными" значениями, чем больше количество данных в фоновом наборе – тем лучше результат [11].

DeepExplainer в SHAP вычисляет значения Шепли, аппроксимируя их через интегрирование градиентов модели от фонового значения признаков к текущему. Вместо явного перебора всех подмножеств, как в оригинальной формуле Шепли, DeepExplainer аппроксимирует вклад, вычисляя, как изменение признака влияет на выход модели относительно фонового значения. Это позволяет эффективно интерпретировать сложные модели глубокого обучения, такие как нейронные сети, сохраняя при этом теоретическую обоснованность значений Шепли.

2.2 Реализация метода объяснимого ИИ

Для апробации модели объяснимого искусственного интеллекта была взята ранее реализованная сверточная нейронная сеть, классифицирующая рукописные цифры MNIST. Для анализа изображения была использована библиотека объяснимого ИИ SHAP. В данном примере анализируется какие части изображения имеют наибольший эффект на предсказании модели. Архитектура сверточной нейронной сети является копией архитектуры рассмотрено выше.

Перед объяснением необходимо задать изображения для background, необходимые для вычисления значений Шепли и test_images, изображения,

которые будут объяснены. Чем больше background изображений, тем лучше будет результат, но вместе с увеличением, падает скорость выполнения программы. Для набора данных MNIST, где изображения и формы относительно просты, достаточно 1000 экземпляров [11].

Путем интегрирования по множеству фоновых примеров, shap.DeepExplainer(model, background) аппроксимируют значения SHAP таким образом, что их сумма равна разнице между ожидаемым выходом модели на переданных фоновых образцах и текущим выходом модели.

Листинг 2.1 подсчет и вывод значений Шепли

```
batch = next(iter(test_loader))
images, _ = batch

background = images[:1000]
test_images = images[100:105]

e = shap.DeepExplainer(model, background)
shap_values = e.shap_values(test_images)

shap_numpy = list(np.transpose(shap_values, (4, 0, 2, 3, 1)))
test_numpy = np.swapaxes(np.swapaxes(test_images.numpy(), 1, -1), 1, 2)

# plot the feature attributions
shap.image_plot(shap_numpy, -test_numpy)
shap.image_plot(shap_numpy, -test_numpy*0, vmax= 0.002)
```

В результате вычисление значений (рис. 5) заметны два разных значения, негативные значения (синие) и позитивные значения (красные). Области позитивных значений указывают на места, которые увеличивают вероятность предсказания данного числа для данного класса. Области негативных значений указывают на уменьшение вероятности предсказания данного класса для данного класса [15]. Например: цифра 0 (рис. 9) окрашена в красный цвет, указывая на то, что данные пиксели изображения больше всего повлияли на результат классификации модели. Так же для нуля характерена окрашенная область в центре, так как отсутствие пикселей в центре характерно для нуля. В некоторых примерах можно заметить, что

модель фокусировалась не на тех классах, которые соответствуют примеру. Например: цифра 6 определена как класс, соответствующий шестерки, но также мы можем заметить, что модель нашла общие черты с классом соответствующий единице, обычно это указывает на переобучение модели, однако для рукописных цифр это ожидаемо, так как они не имеют единую структуру и могут быть сложны в интерпритации.

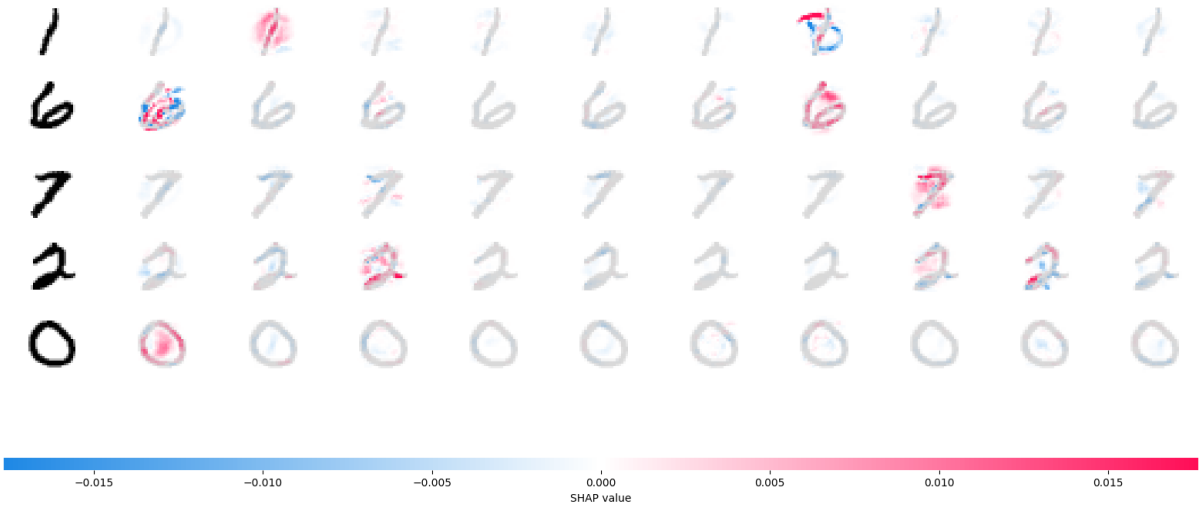

Рисунок 9. Пример объяснения предсказания сети

3. Задача Классификации траекторий мелких лабораторных животных

3.1 Водный лабиринт Морриса

Поведенческий тест «Водный Лабиринт Морриса» разработанный Ричардом Моррисом в 1981 году, применяется для изучения состояния центральной нервной системы мелких лабораторных животных, а так же для оценки их когнитивных способностей (рис. 10). Тест-установка «Водный лабиринт Морриса» - это круглый бассейн, заполненный обычно окрашенной водой, в котором скрыта небольшая платформа чуть ниже поверхности воды [12]. Во время тестирования лабораторное животное должно найти и подняться на скрытую платформу. Поскольку платформа не видна, животное полагается на свою память и ориентиры вокруг бассейна, чтобы понять, где она находится. В качестве лабораторных животных чаще всего используют крыс, так как они являются хорошими пловцами [14].

Тест состоит из нескольких этапов:

Обучение: животное помещают в бассейн в случайной точке в один из квадрантов бассейна. Животное плавает пока не найдет платформу, которую не видит, по истечению 60секунд животное достают отдохнуть 10 – 20 минут. Обычно в течении 3-5 дней животное сделает 4-6 заплывов в день. Со времен животное запоминает, где находиться платформа.

Пробное испытание: животное помещают в бассейн без платформы или перемещают в другую часть установки, для проверки памяти. Так же существуют модификации теста, с подогревом определенных квадрантов установки.

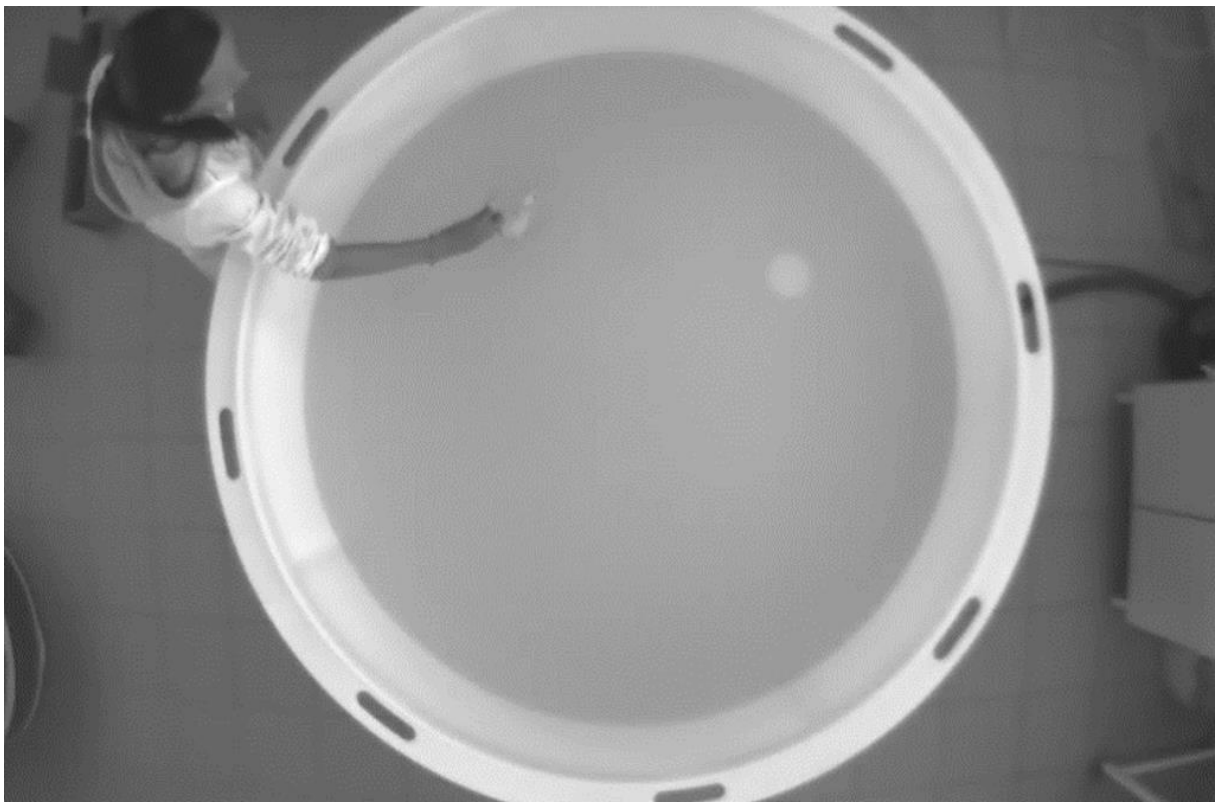

Рисунок 10. Тест система "Водный лабиринт Морриса"

3.2 Классификация траекторий

Анализ стратегии поиска. Стратегии поиска во время приобретения, тренировки и пробного тестирования были проанализированы с помощью программы Pathfinder (Лаборатория Джейсона Снайдера, Ванкувер, Канада) [15]. Было выделено 8 возможных стратегий плавания (рис. 11):

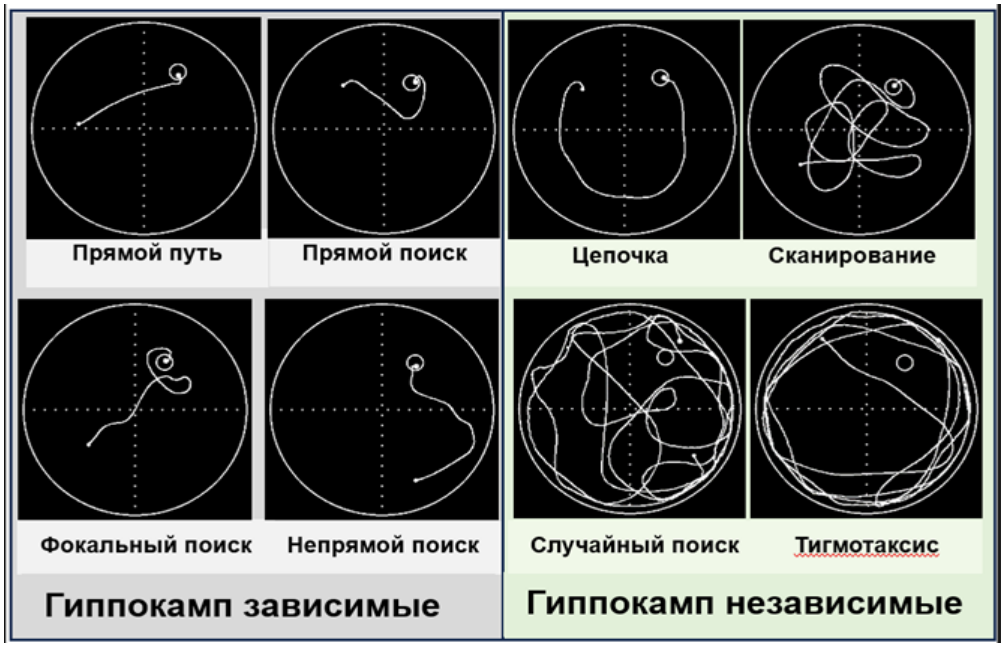

Рисунок 11. Приобретенные стратегии поиска

Различные пространственные параметры были настроены в соответствии с экспериментальной установкой (положение цели [x/y]: 275, 775; диаметр цели: 200; диаметр лабиринта: 1100; центр лабиринта [x/y]: 550, 550; ширина углового коридора: 40; ширина кольцевой цепи: 200; размер зоны тигмотаксиса: 50).

Пространственные стратегии включали "прямой путь", "направленный поиск", "фокальный поиск", "непрямой поиск". В качестве непространственных стратегий рассматривались "цепочка", "сканирование", "случайный поиск" и "тигмотаксис". Таким образом, "прямой путь" определяется как почти идеальная траектория движения к платформе с минимальным отклонением от прямого пути. “Направленный поиск” описывает стратегию плавания с небольшим отклонением от прямого пути. “Фокальный поиск” означает пространственно ограниченный поиск (в центральной части бассейна), в то время как "непрямой поиск" означает пространственно направленный поиск, содержащий большую ошибку направления. Кроме того, "цепочка" описывает пространственно неспецифическую стратегию, при которой мыши ищут на фиксированном расстоянии от стенок бассейна.

“Сканирование” относится к случайной стратегии, которая избегает стенки. “Случайный поиск” не включает пространственную схему поиска, а "Тигмотаксис" указывает на траекторию плавания, ограниченную стенкой бассейна.

3.3 Реализация модели объяснимого ИИ для задачи классификации

3.3.1 Классификация траекторий

Для классификации траекторий мелких лабораторных животных в поведенческой системе «Водный лабиринт Морриса» была разработана сверточная нейронная сеть на языке Python с использованием фреймворка PyTorch. Для объяснения результатов классификации использовалась библиотека SHAP.

Данные траекторий лабораторных животных включают в себя набор траекторий, разделенные на 8 классов, описанные раннее. В данном примере рассмотрены 3 траектории, визуально различимые между собой.

Перед классифицированием данных изображения были нормализированы до тензора размером (430, 430, 1), где 430 – ширина и высота изображения, а 1 – канал градации серого. Размер обучающего множества – 83, размер тестового множества – 9. Данные изображения были нормализированы по размеру и каналам.

Изначально была разработана сверточная нейронная сеть с тремя свертычными слоями для более глубокого извлечения признаков, так как чем больше набор данных и чем сложнее его интерпретация тем глубже и шире нейронная сеть.

Листинг 3.1. Архитектура сети №1

```
class Net(nn.Module):
    def __init__(self):
        super().__init__()

        self.conv_layers = nn.Sequential(
            nn.Conv2d(1, 32, kernel_size=3, padding=1),  # 32x430x430
```

```
            nn.MaxPool2d(2),                                # 32x215x215
            nn.ReLU(),
            nn.Conv2d(32, 64, kernel_size=3, padding=1), # 64x215x215
            nn.MaxPool2d(2),                                 # 64x107x107
            nn.ReLU(),
            nn.Conv2d(64, 128, kernel_size=3, padding=1), # 128x107x107
            nn.MaxPool2d(2),                                 # 128x53x53
            nn.ReLU(),
            nn.Dropout(0.25),
        )
        self.fc_layers = nn.Sequential(
            nn.Linear(128*53*53, 256),
            nn.ReLU(),
            nn.Dropout(0.5),
            nn.Linear(256, 3),
            nn.Softmax(dim=1),
        )

    def forward(self, x):
        x = self.conv_layers(x)
        #print(x.shape)
        x = x.view(x.size(0), -1)  # Flatten
        x = self.fc_layers(x)
        #print(x.shape)
        return x
```

Однако данная модель показала низкую точность в 67% (рис. 12), данная архитектура справилась, с классификацией классов «1» и «7», но класс «4» в большинстве случаев был классифицирован неправильно, что не

является удовлетворительным результатом.

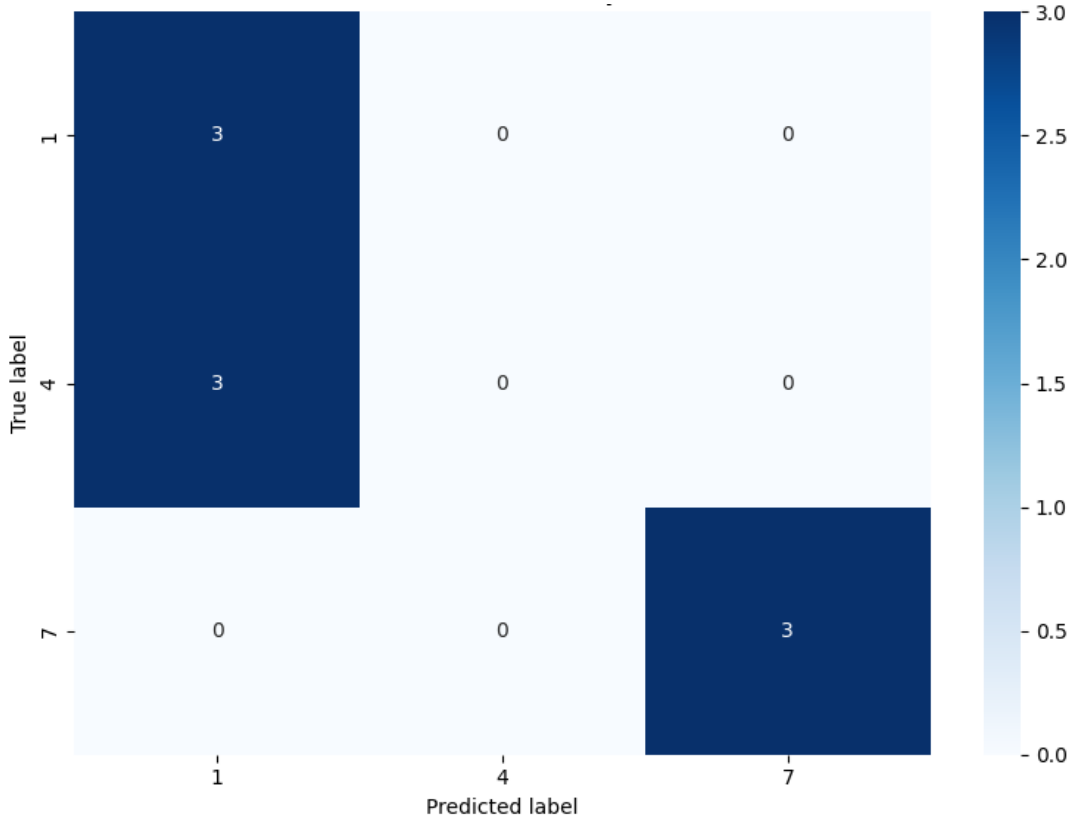

Рисунок 12. Матрица ошибок для первой модели

Причина таких результатов – переобучение (рис. 13) и чрезмерная глубина сверточной нейронной сети, что и привело к неправильной

классификации.

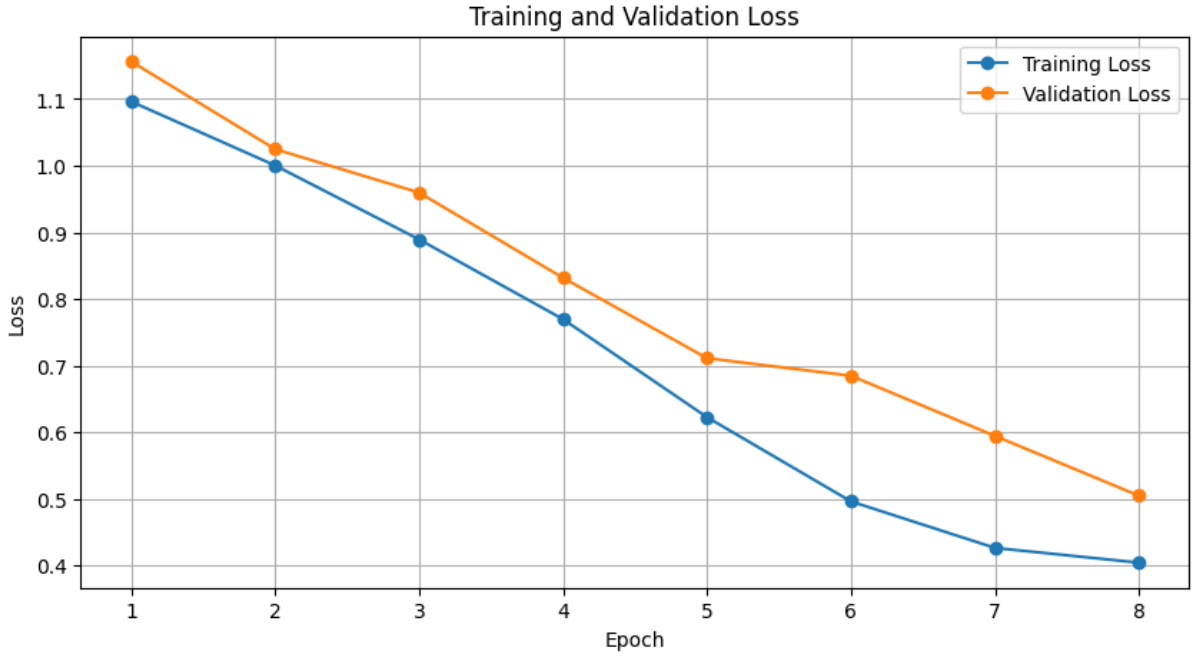

Рисунок 13. Потери на этапах обучения первой модели

Листинг 3.2. Нормализация данных

```
transform = transforms.Compose([
    transforms.Grayscale(),
    transforms.Resize((430, 430)),  # Resize
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))  # Normalize for single channel
])
```

Данная Модель схожа с описанной ранее, за исключением padding=1 в сверточных слоях, который позволяет не уменьшать тензор при применении свертки, так же задан dropout(0.25) так как размер изображения значительно выше, чем у изображений MNIST.

Листинг 3.3. Архитектура сети #2

```
class Net(nn.Module):
    def __init__(self):
        super().__init__()

        self.conv_layers = nn.Sequential(
            nn.Conv2d(1, 10, kernel_size=3, padding=1),  # 16x430x430
            nn.MaxPool2d(2),                             # 16x215x215
            nn.ReLU(),
            nn.Conv2d(10, 20, kernel_size=3, padding=1), # 32x215x215
            nn.MaxPool2d(2),                             # 32x107x107
            nn.ReLU(),
```

```
            nn.Dropout(0.25),
        )
        self.fc_layers = nn.Sequential(
            nn.Linear(20*107*107, 256),  # Изменился входной размер!
            nn.ReLU(),
            nn.Dropout(0.5),
            nn.Linear(256, 3),
            nn.Softmax(dim=1),
        )
```

В результате обучения данной модели была выведена метрика потерь от эпохи обучения (рис. 14), конечное значение потерь на обучающем множестве - 0.138, на тестовом – 0,123. Точность на тестовом множестве равна 89%.

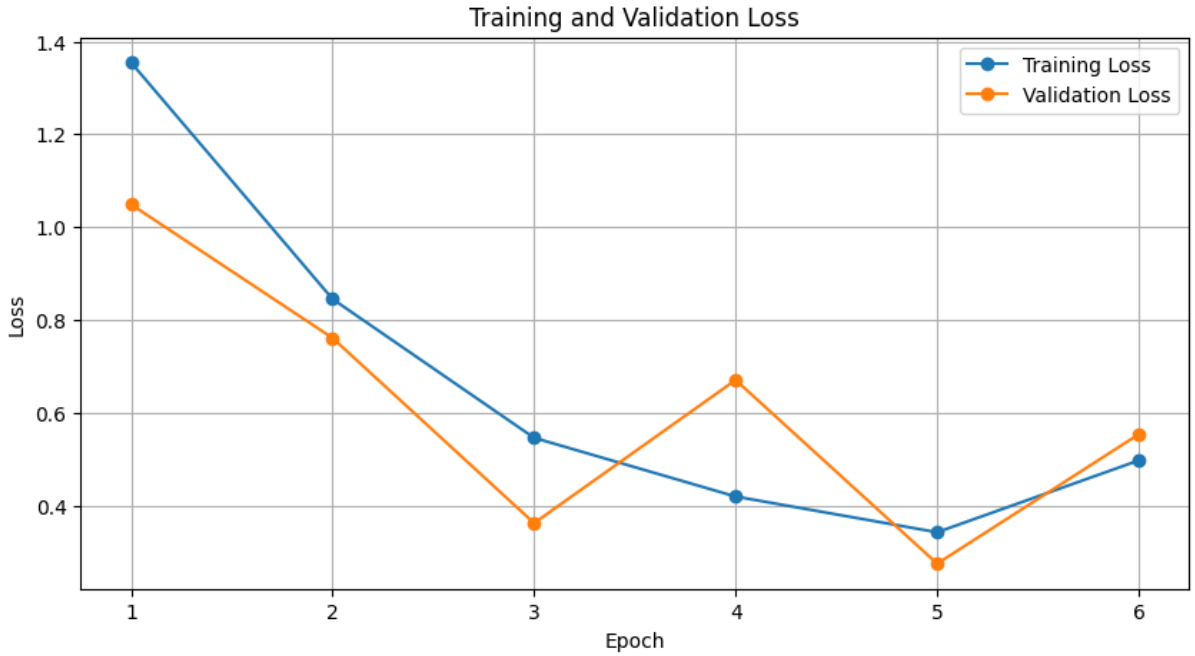

Рисунок 14. Потери на этапах обучения второй модели

Обычно допустимая точность для большинства моделей находиться в пределах 90% - 100%, однако из-за небольшого размера тестового

множества 8 из 9 правильных предсказания - допустимое значение (рис 15).

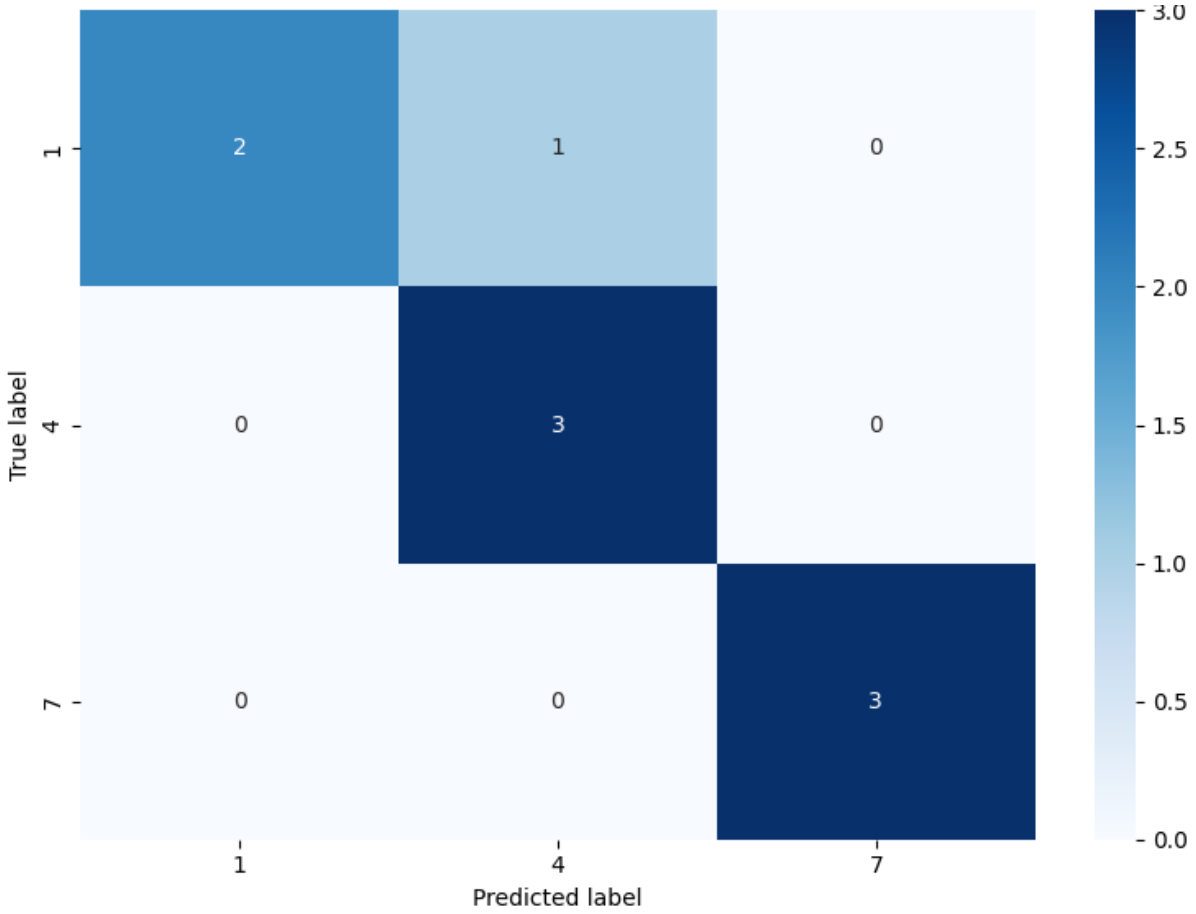

Рисунок 15. Матрица ошибок второй модели

3.3.2 Применение объяснимого ИИ к задаче классификации

Для подсчета значений Шепли с помощью shap.DeepExplainer, в качестве фонового множества было использовано все множество для обучения, и объяснены все экземпляры тестового множества.

Листинг 3.4. Подсчет и вывод значений Шепли

```
train_features, _ = next(iter(train_loader))
test_features, _ = next(iter(test_loader))


background = train_features[:80].to(device)
test_images = test_features[0:9].to(device)

e = shap.DeepExplainer(model, background)
shap_values = e.shap_values(test_images)
```

Траектории классифицированы на 3 класса (рис 16), для 8 изображений, каждое из которых классифицировано правильно, за исключением предпоследнего. Если мы сравним с объяснением предсказаний MNIST (рис. 9) можно заметить, что траектории классов «1» и «4» отмечены как негативное влияние на выбор данного класса, что может быть интерпретировано как неверный результат объяснения. Однако рассмотрев результаты подробнее можно заметить несколько признаков, которые указывают на положительный результат объяснения. Траектории класса «7» имеют намного больше информации о траектории, нежели траектории классов «1» и «4», что можно заметить на первом и третьем экземплярах, где вся траектория отмечена как положительное влияние на результат классификации, в то время как в остальных экземплярах она отмечена как положительное влияние в «1» и «4» классах. Во втором экземпляре заметно, что «петля», которая присуща классу траекторий «4» оказывает положительное влияние, на выбор класса, в то время как в четвертом экземпляре ее отсутствие указывает на положительное влияние в определении.

Таким образом, из-за хаотичности и относительно большего пути, модель больше фокусируется на отсутствии большого пути для определения классов «1» и «4» и его наличии для класса «7». Для разделения между «1» и «4» классами модель фокусируется на отсутствии «петли» для «1» класса и ее наличии для «4».

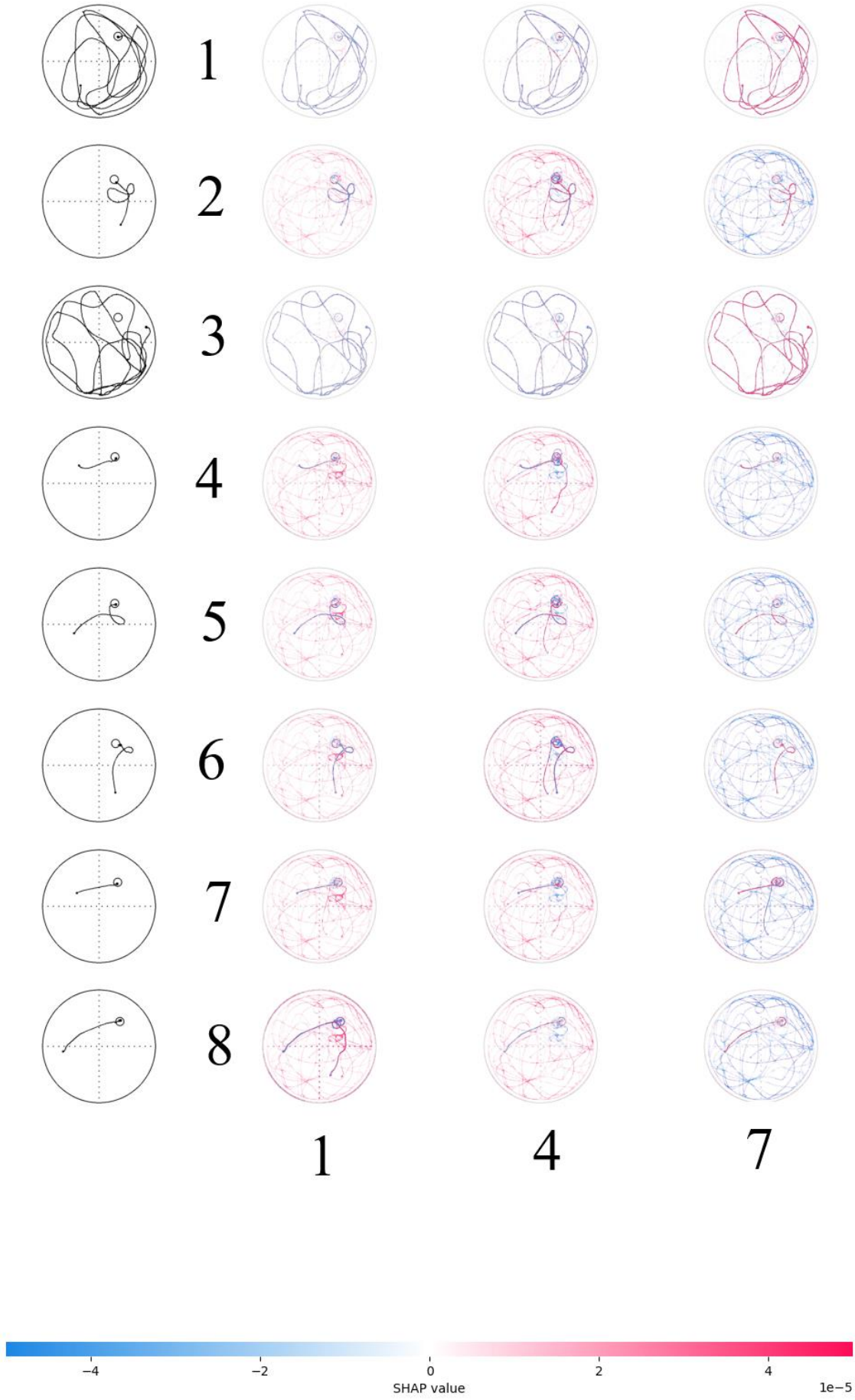

Рисунок 16. Результат объяснения классификации

Заключение

В ходе выполнения магистерской диссертации были изучены основы нейросетевого подхода, архитектуры нейронных сетей, таких как: многослойный перцептрон, сверточные нейронные сети, методы их обучения (метод обратного распространения ошибки и его модификации). На данных основах реализована сверточная нейронная сеть классифицирующая рукописные цифры MNIST, с помощью языка программирования Python, фреймворка PyTorch, а так же библиотек numpy, matplotlib и OpenCV.

Изучены методы и технологии XAI для задач классификации траекторий мелких лабораторных животных в тест системе «Водный лабиринт Морриса». Апробирована модель XAI для нейронной сети, классифицирующей рукописные числа MNIST.

Построена сверточная нейронная сеть для задачи классификации траекторий мелких лабораторных животных в тест системе «Водный лабиринт Морриса». Разработана модель объяснимого ИИ для анализа результатов. Данные для обучения модели классификации траекторий мелких лабораторных животных были предоставлены ЛРБ ОИЯИ.

Отдельная благодарность Татевик Жораевне Бежанян за помощь в изучении материала, связанного с тест системами, лабораторными животными и паттернами их поведения.

Список литературы

1. Palgrave Macmillan. Trustworthy AI. Springer Nature Switzerland AG, 2025.
2. Francois Chollet. On the Measure of Intelligence [Электронный ресурс] // URL: https://arxiv.org/abs/1911.01547v2 (дата обращения: 10.12.2024)
3. Рамсундар Б., Истман П., Уолтерс П., Панде В. Глубокое обучение в биологии и медицине. – Москва, 2020
4. Прадипта Мишра. Объяснимые модели искусственного интеллекта на Python. - Москва, 2022.
5. Франсуа Ш. Глубокое обучение на Python. – Санкт-Петербург, 2018.
6. М.Тим Джонс Программирование искусственного интеллекта в приложениях. Москва 2011.
7. Дж. Вандер Плас. Объяснимые модели искусственного интеллекта на Python для сложных задач наука о данных и машинное обучение. - Санкт-Петербург, 2018
8. База данных рукописных чисел MNIST: [Страница загрузки:] // URL: https://www.kaggle.com/datasets/hojjatk/mnist-dataset (дата обращения: 26.01.2025)
9. Учебник по машинному обучению [Электронный ресурс:] // URL: https://education.yandex.ru/handbook/ml/article/pervie-shagi (дата обращения 01.02.2025)
10. Библиотека torch: [API documentation page] // URL: https://docs.pytorch.org/docs/stable/index.html (дата обращения 04.03.2025)
11. Библиотека SHAP: [API documentation page] // URL: https://shap.readthedocs.io/en/latest/ (дата обращения: 04.02.2025)
12. Установка «Водный лабиринт Морриса», круглая [Электронный ресурс] // OpenScience. – URL:

https://www.openscience.ru/index.php?page=ts&item=021 (дата обращения: 12.02.2025).

13. Д.П. Чернюк, А.Г. Зорин, К.З. Деревцова, Е.В. Ефимова, В.А Приходько, Ю.И. Сысоев, О.Л. Власова, М.В. Болсуновская, И.Б. Безпрозванный. Автоматический анализ данных поведенческого теста «Водный лабиринт Морриса». Журнал высшей нервной деятельности, 2021, том 71 №1, с. 126-135
14. Д.П. Чернюк, А.В. Большакова, О.Л. Власова, И.Б. Безпрозванный. Возможности и перспективы поведенческого теста «Водный лабиринт Морриса». Российский физиологический журнал им. И.М. Сеченова 107(3): 267 - 287. 2021.
15. Multi-class ResNet50 on ImageNet [Электронный ресурс] // URL: https://shap.readthedocs.io/en/latest/example_notebooks/image_examples/image_classification/Multi-class%20ResNet50%20on%20ImageNet%20%28TensorFlow%29-checkpoint.html (дата обращения 04.02.2025)
16. Анализ стратегии поиска у мышей Tg4-42 с болезнью Альцгеймера в водном лабиринте Морриса выявляет ранние нарушения пространственной навигации [Электронный ресурс] // URL: https://www.nature.com/articles/s41598-022-09270-1 (дата обращения: 10.03.2025)
17. Chandan S., W James M., Bin Yu. Hierarchal interpritation for neural network prediction. [Электронный ресурс] // URL: https://arxiv.org/abs/1806.05337 (дата обращения: 12.12.2024)
18. Ляхова К. Н., Утина Д. М., Северюхин Ю. С. Использование алгоритмов компьютерного зрения для анализа экспериментальных данных, полученных при исследовании поведенческих реакций лабораторных животных. – 2020.
19. Беляков В. И. и др. Современные методы изучения поведения грызунов в модельных биомедицинских исследованиях (обзор

проблемы) //Современные вопросы биомедицины. – 2022. – Т. 6. – №. 4 (21). – С. 13-22.

20. Воронцов К. В. Математические методы обучения по прецедентам (теория обучения машин) // Москва. – С. 119-121. – 2011
21. Свёрточные нейронные сети: от основ до современных технологий [Электронный ресурс] // Habr. – 2023. – URL: https://habr.com/ru/articles/887268/ (дата обращения: 17.02.2025).
22. Марк Саммерфилд Программирование на Python. - Санкт-Петербург – Москва, 2009.
23. Библиотека numpy: [API documentation page] // URL: https://numpy.org/doc/ (дата обращения 06.02.2025)
24. Библиотека TensorFlow: [API documentation page] // URL: https://www.tensorflow.org/api_docs/python/tf (дата обращения: 12.03.2025)
25. Среда разработки Jupiter Notebook: [API documentation page] // URL: https://docs.jupyter.org/en/latest/ (дата обращения 08.03.2025).
26. Библиотека Keras: [API documentation page] // URL: https://keras.io/api/ (дата обращения 19.02.2025)
27. Библиотека Grad-CAM: [API documentation page] // URL: https://shap.readthedocs.io/en/latest/ (дата обращения: 15.03.2025)
28. Язык программирования python [API documentation page] // URL: https://www.python.org/doc/ (дата обращения 27.01.2025)
29. Библиотека Lime: [API documentation page] // URL: https://github.com/jacobgil/pytorch-grad-cam (дата обращения: 21.03.2025)
30. Библиотека Matplotlib: [API documentation page] // URL: https://matplotlib.org/ (дата обращения: 01.02.2025) https://opencv.org/
31. Библиотека OpenCV: [API documentation page] // URL: https://opencv.org/ (дата обращения: 02.04.2025)