

**Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования «Казанский (Приволжский) федеральный университет»
Институт вычислительной математики и информационных технологий
Кафедра системного анализа и информационных технологий**

**Направление подготовки: 10.03.01 — Информационная безопасность
Профиль: Безопасность компьютерных систем**

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

**РАЗРАБОТКА АЛГОРИТМА ВЫБОРА ВИДЕО ДЛЯ МЕТОДА
САМООБУЧЕНИЯ НЕЙРОННОЙ СЕТИ**

Обучающийся 4 курса
группы 09-841



(Вирясов В.Е.)

Руководитель
канд. физ.-мат. наук, ст. преподаватель



(Разинков Е.В.)

Заведующий кафедрой системного анализа
и информационных технологий
д-р техн. наук, профессор



(Латыпов Р.Х.)

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ	4
1. Описание предметной области	6
1.1. Методы самообучения нейронной сети	6
1.2. Последние методы	8
1.2.1. Метод «Video Noise Contrastive Estimation»	8
1.2.2. Метод «Cycle-consistency of Time»	12
1.3. Метод «Contrastive Random Walk»	14
1.3.1. Создание графа движения	14
1.3.2. Архитектура нейронной сети	17
1.3.3. Описание алгоритма обучения	20
2. Разработка алгоритма выбора видео	25
2.1. Методы оценки оптического потока	26
2.2. Параметры алгоритма оптического потока	27
2.3. Эвристический алгоритм	30
2.4. Построение набора данных	31
3. Описание экспериментов	34
3.1. Наборы данных	34
3.2. Технические детали	37
3.3. Параметры обучения	39
3.4. Использовавшиеся метрики	40
3.5. Результаты экспериментов	42
ЗАКЛЮЧЕНИЕ	47
СПИСОК ЛИТЕРАТУРЫ	55
ПРИЛОЖЕНИЯ	59
ПРИЛОЖЕНИЕ 1. Алгоритм выбора видео	59
ПРИЛОЖЕНИЕ 2. Визуализация экспериментов	61
ПРИЛОЖЕНИЕ 3. Архивы результатов экспериментов	63

ПРИЛОЖЕНИЕ 4. Эксперименты для отдельных классов	64
ПРИЛОЖЕНИЕ 5. Программный код метода самообучения	66
ПРИЛОЖЕНИЕ 6. Параметры метода самообучения	77

ВВЕДЕНИЕ

В последние времена алгоритмы машинного обучения привлекли большое внимание. Такие технологии удивляют своими результатами в области компьютерного зрения, обработки естественного языка, цифровой обработки сигналов, поиска аномалий и во многих других. Каждое научное открытие в таких направлениях сильно сказывается на жизни людей.

В сфере информационной безопасности машинное обучение играет большую роль. Благодаря обученным нейронным сетям существуют системы контроля доступа на основе распознавания лиц, системы активного аудита позволяют реагировать на угрозы своевременно, технология Face ID от Apple создана для безопасной разблокировки смартфона и подтверждения платежей, в антивирусных программах сканеры позволяют оценивать угрозы, таких примеров достаточно много. Почти каждый день люди интересующихся наукой читают научные публикации в области искусственного интеллекта и выделяют для себя какие-то новые идеи, методы и ухищрения для достижения какой-то новой цели или получения лучших результатов работ на замену предыдущим.

Каждый алгоритм с участием нейронных сетей требует достаточно большого количества данных для обучения. И чем сложнее задача, тем больше времени нужно уделить на создание и обработку набора данных для обучения такой сети. В задачи сегментации изображений, когда важен каждый пиксель, необходимо потратить много времени на разметку одного только изображения, поэтому, когда для задачи требуется как минимум несколько сотен тысяч картинок, компании трудно найти столько времени и средств, чтобы создать столько данных.

В последние годы сообщество искусственного интеллекта проводят исследования в упрощении такой задачи. Методы самообучения (англ. self-supervised learning) нейронной сети дают прирост в показателях точности

для различных задач. С помощью них такие модели могут понимать какие-то закономерности в данных без разметки, выполняемый человеком. Такого рода методы наиболее востребованы в задаче сегментации для обучения с учителем или в более сложных задачах одновременного отслеживания и сегментации объектов в режиме реального времени, когда для создания готовых ответов каждого изображения требуется несколько часов.

Например, в статье [1] 2020 года авторы предлагают с помощью видеофрагментов определять объекты на каждом из кадров путем повторения их в обратном порядке. Таким образом люди ищут способы научить нейронные сети понимать данные так, чтобы для других задач требовался небольшой набор данных. Но при обучении таких моделей изначально существует большая выборка, среди которых не все видео могут давать одинаковый эффект при вычислении градиента. Поэтому было бы полезно иметь некоторый критерий отбора элементов, способный грамотно создавать обучающую выборку, чтобы уменьшить объем данных, но при этом точность не падала.

Таким образом, целью данной выпускной квалификационной работы является разработка алгоритма выбора видео для метода самообучения нейронной сети. Для достижения данной цели были поставлены следующие задачи:

- изучение последних методов по самообучающимся нейронным системам,
- реализация эвристического алгоритма построения набора данных для обучения модели,
- реализация модели для получения векторного представления объектов с использованием последовательности кадров из отрезка видео,
- обучение модели на полученным наборе данных,
- проведение экспериментов, валидация гиперпараметров,
- анализ полученных результатов.

1. Описание предметной области

Предметная область методов самообучения нейронной сети в последние годы набирает большую популярность. Во многих задачах благодаря им были достигнуты приросты показателей точности. Каждый год люди предлагают новые подходы и идеи сделать так, чтобы максимально сократить участие человека в обработке данных и настройке при обучении нейронной сети.

На сегодняшний день очень популярна техника трансферного обучения (англ. transfer learning):

- 1) для решения своей задачи, где недостаточно данных, берут готовый результат работы, полученный на другой задачи (англ. pre-trained), например, обученные веса нейронной сети для задачи классификации объектов огромного датасета ImageNet [2];
- 2) архитектуру нейронной сети изменяют под свою задачу;
- 3) модель с весами, полученными на первой задаче, используют для обучения сети на своей задаче (англ. downstream task).

Но теперь метод использование готовых параметров перед обучением нейронной сети постепенно уходит в сторону и вместо него используют параметры полученные методами самообучения. Потому что, во-первых, необходимо большое количество вычислительных ресурсов для достижения заведомой точности. Во-вторых, собственные данные могут быть не похожи на те, что предлагает ImageNet [2], что усложняет задачу обучения.

1.1. Методы самообучения нейронной сети

На данный момент ведутся работы над созданием алгоритмов, которые позволили бы указать метки изображений автоматически (англ. pseudo labels), на которых потом обучается модель, которая позволяла бы получить векторное представление (англ. representations) объектов на изображении. Сегодня существуют различные задачи (англ. pretext task), которые позволя-

ют этого достичь. Например, задача определения положения частей картинки относительно друг друга [3], задача определения угла поворота картинки [4], задача решения головоломки из частей картинки [5] и многие другие. Общая схема методов самообучения нейронной сети показана на рисунке 1.

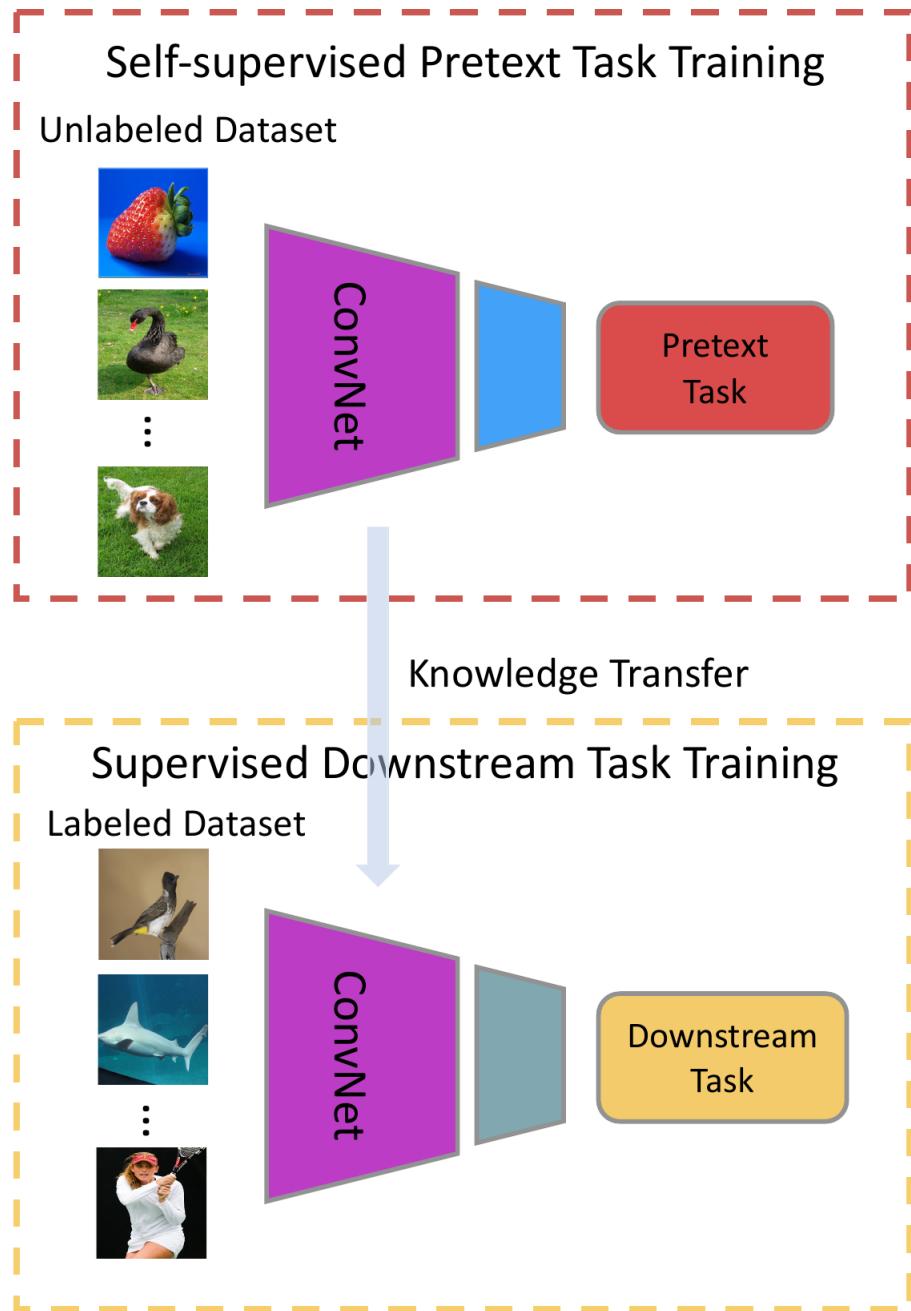


Рисунок 1 – Общая схема методов самообучения нейронной сети

Сейчас наиболее интересны методы, связанные с работой с видеофрагментами, которые представляют собой последовательности изображений. Также существуют методы, которые используют звуковые дорожки видео-

фрагментов как готовую разметку данных. Такие подходы, дают больше простора для постановки предварительных задач. Если это была бы задача обучения с учителем (англ. supervised learning), тогда человеку было бы необходимо вручную указывать метки классов для каждого видеофрагмента. Например, есть два видеофрагмента одинаковых длин, на которых показаны движения объектов. Например, первый видеофрагмент, на котором показано движение велосипеда, каждый кадр будет иметь метку класса (англ. ground truth) – велосипед. На втором видеофрагменте будет показана поведение кошки, тогда каждый кадр будет иметь метку класса – кошка.

Для подходов самообучения часто используют алгоритмы вычисления векторных представления изображения, где основная идея в том, чтобы векторные представления изображений с одинаковыми метками класса по углу между ними были близки, а векторные представления изображений с разными метками класса по углу между ними оказались далеки. Похожее можно найти в задаче классификации методом однократного обучения (англ. one-shot learning), где используют метки классов похожие или непохожие объекты, вместо понятия расстояния и угла между векторами.

1.2. Последние методы

С каждым годом появляются новые подходы получения векторного представления изображений с использованием видеофрагментов. Кто-то пытается сравнивать различные изображения, а кто-то используют хитрые алгоритмы отслеживания объектов на изображениях с одного видеофрагмента.

1.2.1. Метод «Video Noise Contrastive Estimation»

В мае 2020 года авторы статьи [6] продемонстрировали подход Video Noise Contrastive Estimation (VINCE). Как визуально отдаляют вектора от объектов с двух разных видеофрагментов статьи [6] показана на рисунке 2.

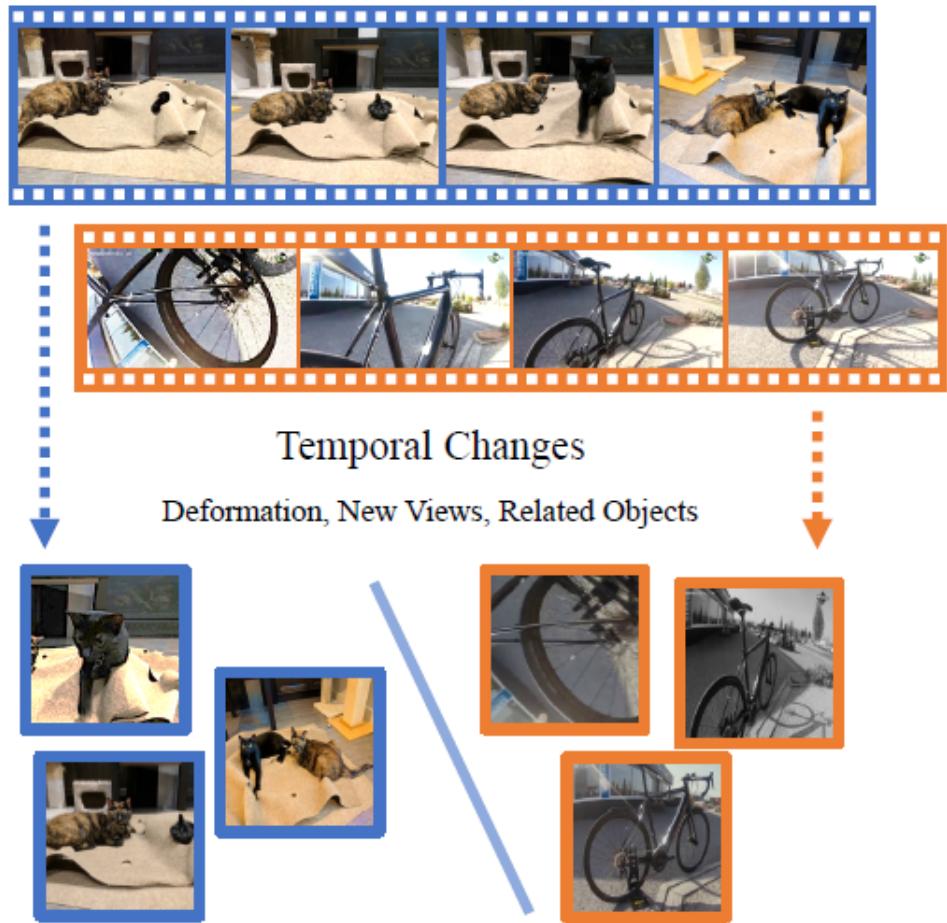


Рисунок 2 – Визуализация метода VINCE [6]

Для реализации получения векторных представлений изображений, реализуют или используют специальную функцию кодировщик (англ. encoder). В качестве такой функции может служить сверточная нейронная сеть. С помощью операции свертки и определенного количества таких слоев можно получить специальную матрицу, называющуюся картой признаков (англ. feature maps), которая хранит в себе признаки объектов. Далее с помощью операции Global Average Pooling каждую отдельную карту признаков можно усреднить к одному числу и так для каждой, где на выходе получается вектор. Этот вектор будет являться представлением изображения. Дополнительно такую модель можно усложнить, добавив в конце некоторое количество полно связанных слоев.

Вектора, которые будут находиться на различных расстояниях в зависимости от схожести объектов на изображениях, можно получить благодаря минимизации специальной целевой функции. Таких функций много, все претерпевали некоторые изменения. Например, целевая функция Standard NCE [7] для оптимизации векторов выглядит следующим образом:

$$L_{NCE} = -\frac{1}{n} \sum_{i=1}^n \frac{e^{sim(A_i, P_i)}}{\sum_{j=1}^n e^{sim(A_i, P_j)}}, \quad (1)$$

где sim – функция косинусного сходства для двух входных элементов, A – выбранная группа изображений (англ. batch), B – вторая группа изображений, в котором есть хотя бы один элемент похожий на один из элементов группы A . Элемент похожий с выбранным элементом (англ. anchor) называют позитивным элементом (англ. anchor-positive), в противном случае остальные элементы называют негативными (англ. anchor-negative). Сама функция косинусного сходства в формуле (1) напоминает скалярное произведение между двумя векторами:

$$sim(X, Y) = \tau \frac{f(X)}{\|X\|} \frac{f(Y)}{\|Y\|}, \quad (2)$$

где f – функция кодировщик, которая на выходе возвращает вектор, τ – коэффициент концентрации представления объектов (англ. temperature). Этот коэффициент позволяет учитывать значимость расстояний между векторами, чем ниже значение, тем больше преобладают небольшие расстояния и вектора с большими расстояниями вносят меньший вклад в целевую функцию.

В методе [6] авторы утверждают, что в целевой функции (1) используется мало негативных элементов, тем самым модели сложно отдалить такие пары векторов. Чтобы решить эту проблему, было добавлено больше негативных элементов и больше положительных пар Multi-Frame Multi-Pair

NCE:

$$L_{NCE} = -\frac{1}{n} \sum_{i=1}^n \frac{e^{sim(A_i, P_i)}}{e^{sim(A_i, P_i)} + \sum_{j=1}^m e^{sim(A_i, N_j)}}, \quad (3)$$

где N – группа изображений из негативных элементов. Для расширения количества данных, для обоих типов элементов используют процедуру аугментации, где изображения перед входом модели зеркально отображают, меняют интенсивность пикселей или случайно обрезают часть картинки и потом масштабируют. В итоге полученная обучающая выборка, при использовании функции (3) с мерой угла (2), может выглядеть так, как показана на рисунке 3.

	Current Positives							MoCo MemBank							
Anchors															
	<img alt="Person 1														

алгоритмы. Вдобавок требуется достаточно большая группа изображений с присутствием негативных элементов N .

1.2.2. Метод «Cycle-consistency of Time»

Методы, связанные с использованием положительных и отрицательных элементов, не единственные для получения векторных представлений. Существуют и сложные методы, которые предлагают использовать соседние изображения видеофрагментов для отслеживания движения объектов специальными алгоритмами. Так, например, авторы метода, под названием TimeCycle [8], предлагают сконцентрировать внимание на конкретных частях объектов на картинки и определять куда эта часть сместится после нескольких кадров одного видеофрагмента. Для этого необходимо научить модель отслеживать выбранную часть картинки на протяжение нескольких кадров.

Сначала нужно выбрать небольшой фрагмент последнего кадра (англ. patch), на котором, например, будет изображено лицо человека. Модель должна отслеживать положения объекта на протяжении всех кадров видеофрагмента. Переход фрагмента на соседний кадр определяется с помощью специального алгоритма отслеживания, показанного на рисунке 4.

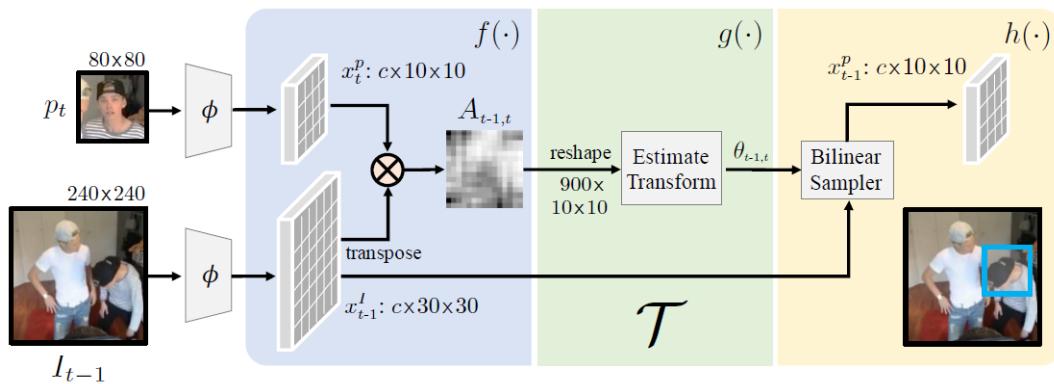


Рисунок 4 – Алгоритм отслеживания объекта на кадре [8]

Для начала с помощью сверточной нейронной сети извлекают признаки от выбранного фрагмента x_t текущего кадра размером 80×80 получают карту признаков размером $cx \times 10 \times 10$, от предыдущего кадра x_{t-1}^2 размером $240 \times$

240 получают карту признаков размером $cx \times 30 \times 30$, где cx – количество каналов на выходе модели в зависимости от архитектуры сети. Далее этапы отслеживания выглядят следующим образом:

- 1) аффинная функция $f(\cdot)$ для определения меры сходства между признаками x_t^p и x_{t-1}^I ,
- 2) функция локализации $g(\cdot)$ для оценки параметров определения позиции,
- 3) билинейная функция выбора $h(\cdot)$ для получения нового признака объекта x_{t-1}^p .

Значения аффинной функции, представляют собой меру сходства между координатами двух объектов $f(x^p, x^I)$, которые могут быть получены путем преобразования $f : R^{cx \times 30 \times 30} \times R^{cx \times 10 \times 10} \rightarrow R^{900 \times 100}$. Для каждого элемента вычисляется следующее значение:

$$A(i, j) = \frac{\exp(x^I(j)^T x^p(i))}{\sum_j \exp(x^I(j)^T x^p(i))},$$

где сходства $A(j, i)$ нормализуется с помощью функции softmax по измерению x^I для каждого $x^p(i)$. Для оценки параметров локализации используется небольшая сеть, состоящая из двух сверточных и одного полно связанных слоя. На выходе будут три параметра: перемещение в 2D-пространстве и угол поворота, полученные с помощью преобразования $g : R^{900 \times 10 \times 10} \rightarrow R^3$. Последний шаг для получения нового x_t^p , будет билинейное преобразование текущего кадра x_t^I и полученных параметров с помощью функции g .

Обучение такой модели требует попарное отслеживание объектов относительно каждого кадра и полученного фрагмента изображения из соседнего кадра, как показано на рисунке 5.

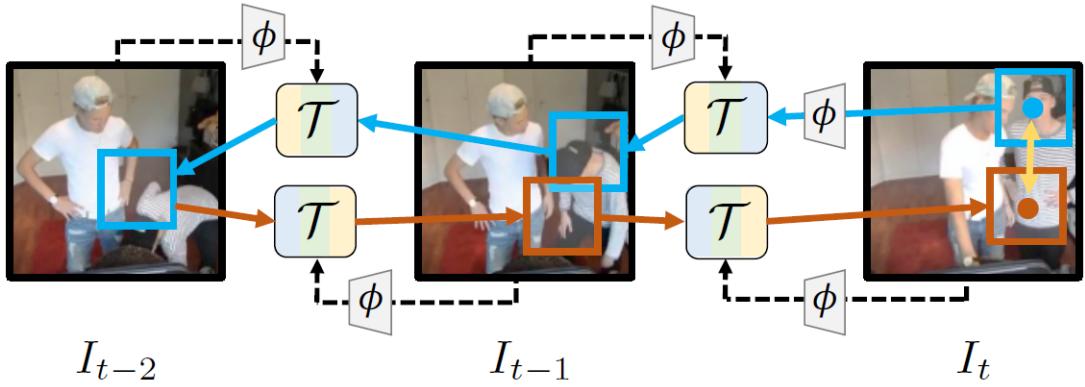


Рисунок 5 – Процесс обучения модели TimeCycle [8]

Такой алгоритм может отслеживать объект между кадрами, но имеет следующие недостатки:

- в одной итерации используется один фрагмент изображения (англ. patch),
- используются сложные алгоритмы отслеживания,
- показатели точности не улучшаются при большом количестве кадров для отслеживания (больше 6 изображений),
- показатели точности не улучшаются при увеличении размера обучающей выборки.

1.3. Метод «Contrastive Random Walk»

В качестве основного метода для самообучения нейронной сети была выбрана работа авторов [1] под названием «Contrastive Random Walk». Помимо вычисления сходства соседних участков изображений между кадрами это модель объединяет некоторые идеи из прошлых работ.

1.3.1. Создание графа движения

Пусть есть изображения I_1, \dots, I_t из одного видеофрагмента размерами $H \times W$. Каждый кадр разбивается на специальную сетку, полученную с помощью специального скользящего окна небольшого размера 64×64 с некоторым шагом смещения равным 32 пикселям. В этой сетке каждый элемент представляет собой небольшой фрагмент, который может покры-

вать определенную часть объекта на изображении. Так, например, каждый фрагмент кадра, на котором показано движение человека, каждый фрагмент отдельные его части тела: руки, ноги, торс, голову и так далее. Когда человек совершает какое-то движение в каком-то направлении на видео, его части тела двигаются от кадра к кадру, изменяя положение частей тела от одного фрагмента к другому.

Таким образом, можно построить алгоритм обучения нейронной сети для задачи классификации частей объектов на изображении для каждого фрагмента после перехода к следующему видеофрагменту спустя какое-то количество пропущенных кадров, как показано на рисунке 6.

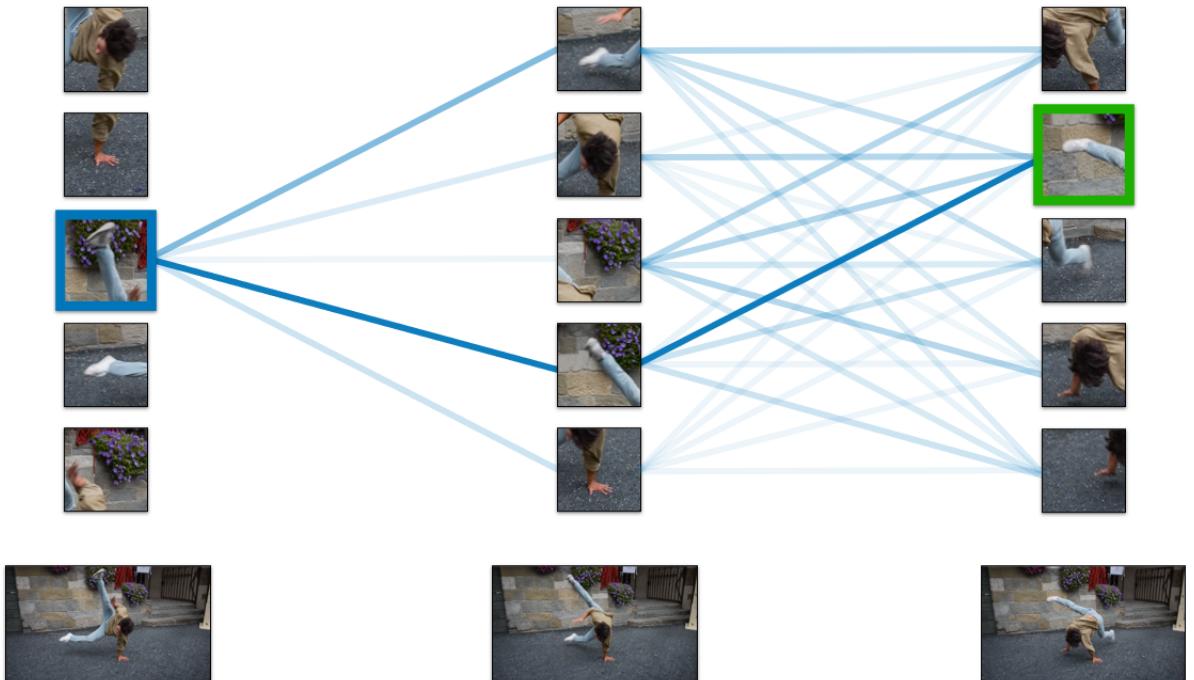


Рисунок 6 – Пример перемещение фрагментов изображения из статьи [1]

Разбиение каждого кадра можно представить в виде ориентированного графа, где его вершины являются представлениями фрагмента изображения, а взвешенные ребра будут соединять вершины соседних кадров. Движение между ними определяется с помощью неотрицательного значений, которые представляют с собой вероятности перехода от i -ой вершины кадра (момента времени) t к j -ой вершине кадра $t + 1$, полученные с помощью построчного

использования функции softmax:

$$A(i, j) = \text{softmax}(Q_t Q_{t+1}^T)_{ij} = \frac{\exp(d_\phi(q_t^i q_{t+1}^j)/\tau)}{\sum_{l=1}^N \exp(d_\phi(q_t^i q_{t+1}^l)/\tau)}, \quad (4)$$

где q_t – набор из N вершин времени t , каждая из которых размерностью R^d , d_ϕ – функция сходства векторов, которая определяет угол между ними, τ – коэффициент баланса близости векторов, $Q \in R^{N \times d}$ – матрица всех вершин q . Если учитывать, что изображения используются из одного видеофрагмента, то случайное перемещение каждого фрагмента в последующие вершины можно рассматривать как отслеживание путем сопоставления схожести соседних вершин. Визуальный пример, где каждый кадр одного видео из домена пикселей переводят в домен вершин, показан на рисунке 7.

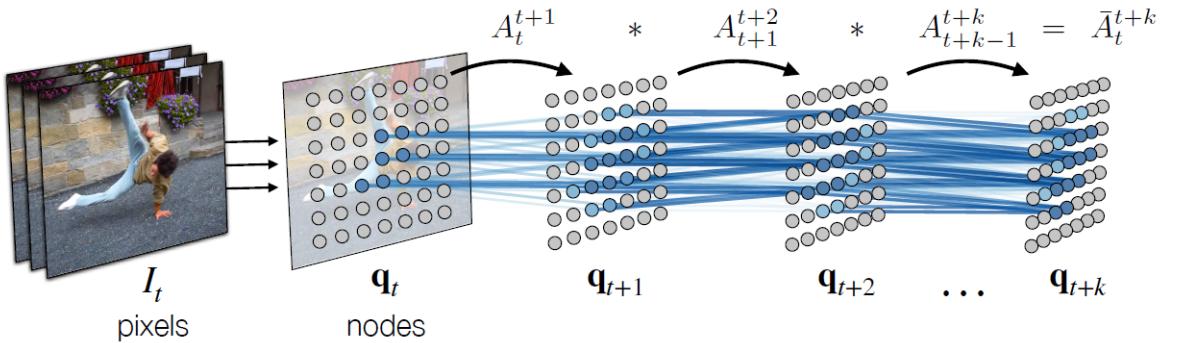


Рисунок 7 – Переход к графам из статьи [8]

В этом случае формулу (4) можно представить как условную вероятность перехода $A_t^{t+1}(i, j) = P(X_{t+1} = j | X_t = i)$, где X_t – состояние перемещения вершин в момент времени t , $P(X_t = i)$ – вероятность нахождения в вершине i в момент времени t . Тогда суперпозиция движения для последующих кадров может быть найдена по формуле:

$$\bar{A}_t^{t+k} = \prod_{i=0}^{k-1} A_{t+i}^{t+i+1} = P(X_{t+k} | X_t).$$

Возможны ситуации, когда объект на видеофрагменте перемещается так от кадра к кадру, что ни на одном из фрагментов не удается полностью его

отобразить. Например, человек на видео совершает движение танца, его нога отображена на фрагменте (в вершине) q_t^i . Спустя несколько пропущенных кадров это нога переместилась на другой фрагмент q_{t+1}^j , при этом объект покрывается не полностью, существует еще один соседний q_{t+1}^{j+1} , который частично задевает этот же объект. В такой ситуации авторы предложили обнулить или присвоить некоторым случайнм вершинам очень маленькое отрицательное значение с помощью операции Dropout [9]. Тогда вероятность перехода некоторых фрагментов изображений в последующий кадр будет минимальной. Изменение значения перехода определяется случайнм образом по распределению Бернулли с вероятностью успеха σ . После этой операции новые значения необходимо нормализовать с помощью следующей формулы:

$$B_{ij} = \frac{\tilde{A}_{ij}}{\sum_{l=1}^N \tilde{A}_{il}},$$

где \tilde{A}_{ij} – вес перехода из i -ой в j -ую вершину после операции $dropout(A, \sigma)$.

В этом случае алгоритму дана возможность рассматривать альтернативные пути для нахождения сходства. Сам параметр σ был провалиден и установлен методом проб и ошибок. По словам авторов оказалось, что значение $\sigma \in [0.1, 0.2]$ дает больше всего прироста в точности.

1.3.2. Архитектура нейронной сети

Как было описано ранее, для получения векторных представлений обучают специальный кодировщик f , который представляет собой нейронную сеть. Обычно это известная архитектура сверточной сети, где на выходе применяется специальная операция для преобразования карт признаков в некоторый вектор размерности R^d .

В качестве глубокой сверточной нейронной сети в методе «Contrastive Random Walk» [1] используется архитектура ResNet-18 [10], которая показана в таблице 1.

Таблица 1 – Архитектура нейронной сети ResNet-18 [10] в методе «Contrastive Random Walk» [1]

Название слоя	Тип слоя	Размер, количество, шаг фильтра	Размер выхода
input	–	–	$224 \times 224 \times 3$
conv1	Convolution + Batch-Norm	$7 \times 7, 64, 2$	$112 \times 112 \times 64$
max-pooling	Max-Pooling	$3 \times 3, 1, 2$	$56 \times 56 \times 64$
conv2_x ($\times 2$)	Convolution + Batch-Norm	$3 \times 3, 64, 1$	$56 \times 56 \times 64$
	Convolution + Batch-Norm	$3 \times 3, 64, 1$	
conv3_x ($\times 2$)	Convolution + Batch-Norm	$3 \times 3, 128, 2$	$28 \times 28 \times 128$
	Convolution + Batch-Norm	$3 \times 3, 128, 2$	
conv4_x ($\times 2$)	Convolution + Batch-Norm	$3 \times 3, 256, 2$	$14 \times 14 \times 256$
	Convolution + Batch-Norm	$3 \times 3, 256, 2$	
conv5_x ($\times 2$)	Convolution + Batch-Norm	$3 \times 3, 512, 2$	$7 \times 7 \times 512$
	Convolution + Batch-Norm	$3 \times 3, 512, 2$	
average pool	Average Pooling	7×7	$1 \times 1 \times 512$
classification layer	Fully-Connected	512×1000	1000

В данной архитектуре авторами были изменены некоторые параметры нескольких последних слоев. В оригинальной статье [10] после прохождения блока conv5_x из таблицы 1 коэффициент изменения масштаба тензора равен $\frac{1}{32}$. Авторы предложили изменить шаг свертки с 2 до 1 у последних двух сверточных блоков. Такое изменение позволит сохранить выходной размер тензора в соответствующих блоках. Помимо этого, был заменен последний полносвязный слой операцией Global Average Pooling, теперь его размерность составляет 128, после которого была добавлена L2 нормализация. В итоге последние слои нейронной сети ResNet-18 [10] выглядят так, как показано в таблице 2.

Таблица 2 – Модификация в архитектуре ResNet [10] в методе «Contrastive Random Walk» [1]

Название слоя	Тип слоя	Размер, количество, шаг фильтра	Размер выхода
conv4_x ($\times 2$)	Convolution + Batch-Norm	$3 \times 3, 256, 1$	$8 \times 8 \times 256$
	Convolution + Batch-Norm	$3 \times 3, 256, 1$	
conv5_x ($\times 2$)	Convolution + Batch-Norm	$3 \times 3, 512, 1$	$8 \times 8 \times 512$
	Convolution + Batch-Norm	$3 \times 3, 512, 1$	
average pool	Average Pooling	7×7	$1 \times 1 \times 512$
embedding layer	Fully-Connected	512×128	128
L2 - norm	L2 normalization	-	128

Архитектуры нейронных сетей бывают разными, но экспериментально было выяснено, что чем больше слоев в модели, тем сложнее ее оптимизировать. До определенного количества слоев показатели точности растут, но после некоторой глубины сети показатели начинают падать (англ. degradation problem). Это связано с тем, что ранним слоям все труднее и труднее передавать информацию в более поздние слои. Особенность архитектур семейства ResNet в том, что существует возможность для некоторых слоев пропустить информацию через несколько последующих. Такие остаточные связи (англ. skip-connections) позволяют пропускать некоторые слои, которые не помогают решить данную задачу. В схеме, показанной на рисунке 8, присутствует пример типичной остаточной связи.

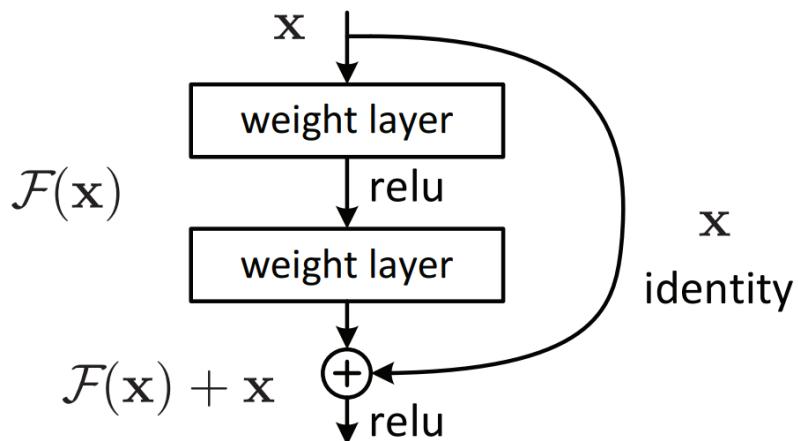


Рисунок 8 – Остаточная связь в архитектурах ResNet [10]

1.3.3. Описание алгоритма обучения

Задача обучения векторных представлений фрагментов состоит в том, чтобы случайные их перемещения совпадали по кадрам (во времени):

$$L_{sup} = L_{CE}(\bar{A}_t^{t+k}, Y_t^{t+k}) = - \sum_{i=1}^N \log P(X_{t+k} = Y_t^{t+k}(i) | X_t = i),$$

где L_{CE} – кросс-энтропия, Y_t^{t+k} – метка классов, где указаны правильные позиции фрагментов для кадра времени $t + k$. Такая целевая функция позво-

ляет максимизировать сходство между вершинами начального (англ. query) и целевого состояния (англ. target) при этом минимизировать сходства между соседними вершинами (англ. negatives).

Случайные последующие движения фрагментов из изображений I_t, \dots, I_{t+k} в различные состояния требуют для обучения от человека разметки готовых ответов Y_t^{t+k} каждого видеофрагмента. Чтобы перейти к способу самообучения, авторы предложили отслеживать фрагменты изображений от начального времени t ко времени $t+k$ и продолжать движение, но в обратном направлении к начальному состоянию (к моменту времени t), представив граф в виде палиндрома, как это показано на рисунке 9.

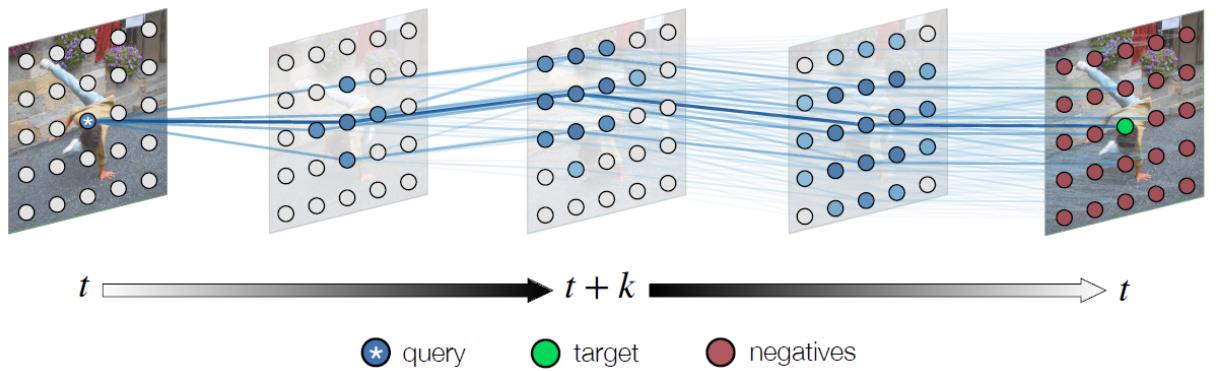


Рисунок 9 – Визуальный пример обучения одного элемента [1]

Такой способ не требует дополнительных действий по разметке данных, так как позиции фрагментов изображений в начальный момент t известны. В этом случае целевая функция выглядит следующим образом:

$$L_{sup} = L_{CE}(\bar{A}_t^{t+k} \bar{A}_{t+k}^t, I) = - \sum_{i=1}^N \log P(X_{t+2k} = i | X_t = i).$$

В случае использования техники Edge Dropout также необходимо учитывать новые значения вершин, которые получили отрицательные значения. Тогда используются новые нормализованные значения B , полученные с

помощью функции softmax:

$$L_{cyc}^k = L_{CE}(B_t^{t+k} B_{t+k}^t, I).$$

Таким образом, метод самообучения нейронной сети для вычисления векторных представлений изображений с использованием обучающей выборки видеофрагментов для одной эпохи обучения можно описать следующим псевдокодом:

Входные данные:

X – обучающая выборка,
 f – функция кодировщик,
 τ – коэффициент баланса расстояний вершин,
 $patch_size$ – размер фрагмента изображения,
 $drop_rate$ – коэффициент для Edge Dropout,
 B – размер группы изображений,
 P – число фрагментов на одном кадре.

BEGIN

```

FOR x IN X DO # Инициализация значение оценки
    x ← Unfold(x, (patch_size,patch_size)) # Разбиение
    кадров видео на фрагменты (B×C×T×H×W→B×C×T×N×h×w)
    x ← Augmentation(x) # Аугментация данных

    v ← f(x) # Выход кодировщика (B×C×T×N)
    v ← L2Norm(v) # Операция нормализации векторов с
    помощью L2 нормы

    A ← einsum("bcti,bctj → btij",v[:, :, :, -1],v[:, :, 1:])
    /τ # Соглашение Эйнштейна для перехода от t к t + 1 графу
    (B×T-1×N×N)

    AA ← cat((A,A[:, :, -1].transpose(-1,-2),1) # Создание
    движения в обратном направлении с помощью отображения графов и
    конкатенации
    AA[rand(AA) < drop_rate] = -1e10 # Edge dropout

    At ← eye(P) # Инициализация начальных позиций
    фрагментов
    FOR FROM t=0 TO 2*T-2 DO # Вычисление переходов по
    графикам
        At ← bmm(softmax(AA[:, t]), dim = -1), At)
```

```

    END FOR

    loss ← At [ [ range (P) ] * B ] ].log () # Вычисление целевой
функции
END

```

С точки зрения реализации такого метода можно выделить следующие особенности:

- в качестве правильных ответов для каждого элемента обучающей выборки используется один вектор $(0, 1, \dots, N - 1)$,
- используемая функция Соглашение Эйнштейна позволяет сократить количество циклов,
- для сокращения вычислений случайного движения в каждой группе изображений используется специальная функция умножения,
- используется только одна нормализация softmax после Edge Dropout.

Авторы с помощью экспериментов показали лучший результат по сравнению с прошлыми работами в одинаковых задачах. В большинстве случаев нельзя добиться такого результата без выбора оптимальных гиперпараметров для обучения. С помощью метода проб и ошибок авторами были подобраны следующие значения гиперпараметров, показанные в таблице 3.

Таблица 3 – Значения гиперпараметров авторов в методе [1]

Гиперпараметры при обучении	Значения
Шаг обучения	0.0001
Коэффициент τ	0.07
Размер векторного представления изображений	128
Размер кадра	256
Длина видеофрагмента	2, 4, 6, 10
Коэффициент δ	0, 0.05, 0.1, 0.2, 0.3
Количество пропущенных кадров в каждом видеофрагменте	2, 4, 8, 30

Продолжение таблицы 3

Гиперпараметры при обучении	Значения
Размер фрагмента	64
Шаг смещения каждого фрагмента	32
Диапазон обрезания изображений при аугментации	(0.7, 0.9)

2. Разработка алгоритма выбора видео

В рассмотренных методах самообучения нейронной сети человеку нет необходимости размечать каждый элемент в выборке, но для обучения требуется множество различных изображений. Такое количество данных можно получить, используя базу данных видео, например, видеохостинг YouTube [11]. Благодаря такому веб-сервису сотрудники из компании DeepMind проанализировали и собрали огромный набор данных человеческих действий или движений в различных жизненных ситуациях. Такой проект получил название Kinetics-400 [12], где отсортировано около 400 различных классов действий человека.

Обычно в глубоком обучении, когда в выборке много данных это значит, что можно обучить нейронную сеть, где метрики точности для данной задачи будут показывать как минимум приемлемый результат. В наборе данных [12], общий размер которого превышает 300 гигабайт, присутствует больше полумиллиона коротких видеоклипов. Каждый из них дает разный вклад в процессе обучения кодировщика: одни видеофрагменты с малой пользой в минимизации целевой функции стоит исключить, а другие с большей пользой необходимо оставить. Для обучения кодировщиков в методах самообучения нейронной сети такой анализ мог бы улучшить конечный результат. Предполагается, что элементы с большим "количество" движения дают больше пользы при обучении.

Идея состоит в том, чтобы создать эвристический алгоритм построения обучающей выборки, который смог бы оценить "количество" движения в каждом видео. Тогда нет необходимости использовать все элементы из набора данных, можно отобрать только те, которые предположительно дадут больший вклад и оптимальный размер группы изображений при обучении, особенно, когда вычислительных ресурсов не так много.

Один из инструментов для работы с видеофрагментами в компьютерном зонии это алгоритмы оценки оптического потока. Данные алгоритмы вычисляют сдвиги (dx, dy) , выделяют движущиеся объекты между двумя кадрами $I_1(x, y)$ и $I_2(x + dx, y + dy)$.

2.1. Методы оценки оптического потока

Существует множество методов оптического потока, каждый предполагает, что интенсивность при переходе к следующему кадру в каждой точке сохраняется $I_1 = I_2$. С помощью разложение в ряд Тейлора до первого члена интенсивности точек получаем уравнение с двумя неизвестными:

$$I_x dx + I_y dy + I_t dt = 0$$

Согласно теореме Кронекера – Капелли [13] для такого уравнения ранг основной матрицы (из коэффициентов I_x и I_y) не равен рангу расширенной матрицы, поэтому однозначного решения получить нельзя. Существуют различные математические методы для оптического потока способные решить данную проблему.

Метод Lucas-Kanade [14]. Стандартный подход, где предполагают, что вокруг пикселя будут примерно одинаковые сдвиги. Поэтому используется весовое окно вокруг этой точки с коэффициентами, распределенными по Гауссу.

Метод Farneback [15]. Использует квадратичную форму с симметричной матрицей для аппроксимирования значений изменения интенсивности пикселей, позволяет вычислять dense-поток благодаря заранее вычисленным параметрам.

Метод Horn-Schunck [16]. Предполагает, что переходы между кадрами будут гладкими, для этого в уравнении добавляется требование: отсутствие резкости изменения сдвигов с весовым коэффициентом.

Метод SimpleFlow [17]. Для нахождения сдвига в окне находят наиболее похожую точку на изображении, тогда вычислять производные, для окна которого не позволяет находить сдвиги больше, чем его размер, нет необходимости.

Каждый метод имеет свои достоинства и недостатки. Некоторые из способов быстрее всего вычисляют значения оптического потока, какие-то вычисляют результат точнее, но медленнее. Выбор метода зависит от поставленной задачи и компромиссов между точностью и скоростью работы.

2.2. Параметры алгоритма оптического потока

В качестве основного метода для вычисления оптического потока был выбран Farneback [15], так как этот метод позволяет вычислять dense-поток за достаточно короткое время. Квадратичная форма, которая описывает значение интенсивности в данном способе, выглядит следующим образом:

$$I(x) = x^T A x + b^T x + c, \quad (5)$$

где A, b, c – параметры аппроксимации функции интенсивности пикселя. На некоторых фрагментах изображения могут возникать резкие изменения интенсивности. В этом случае используется окно с весовыми гауссовскими коэффициентами w . Также остается проблема с вычислениями больших сдвигов в 2–3 пикселя. Для решения такой проблемы вычисляют оптический поток на разных масштабах изображения. На самом маленьком масштабе вычисляются сдвиги в порядке 1–2 пикселей, потом изображение, например, увеличивают в 2 раза и уточняют значение сдвига с предыдущего шага и так далее (англ. multi-scaling). Визуально это можно представить, как построение “пирамиды” разного масштаба.

Существует готовая библиотека на языке программирования Python, которая содержит уже реализованные и оптимизированные функции для работы с оптическим потоком. OpenCV [18] – библиотека компьютерного

зрения, которая обрабатывает изображения и имеет готовые инструменты для работы с ними. Для использования метода Farneback [15] существует функция calcOpticalFlowFarneback. Процесс вычисления оптического потока в этой функции можно контролировать с помощью следующих аргументов:

- отношение соседних изображений, которое уменьшает размер каждого последующего в указанное количество;
- количество рассматриваемых изображений разных масштабов, включая оригинальное изображение;
- размер весового окна для устойчивости к проблеме шума, чем больше его размер, тем больше шансов на обнаружения смещенного участка на изображении, но более размытое поле движения;
- число итераций, выполняемых алгоритмом, для каждого изображения разного масштаба;
- размер рассматриваемой окрестности пикселя, чем больше его значение, тем лучше аппроксимируются параметры A, b, c в формуле (5);
- значение стандартного отклонения σ весов гауссовского распределения $w \sim N(\mu, \sigma^2)$.

В качестве параметров для функции оптического потока в алгоритме выбора видео использовались следующие значения (указаны в таблице 4):

Таблица 4 – Используемые параметры для функции Farneback [15]

Название параметра	Значение
Отношение масштабов соседних изображений	0.5
Количество рассматриваемых изображений разных масштабов	5
Размер весового окна	11
Число итераций алгоритма	5
Размер окрестности пикселя	5

Продолжение таблицы 4

Название параметра	Значение
Стандартное отклонение $N(\mu, \sigma^2)$	1.1

Выходные данные функции можно визуализировать двумя способами. Первый способ — использование изображения формата HSV (Hue, Saturation, Value), где угол поворота векторов будет использоваться как цветовой тон, а нормализованные значения магнитуды как яркость пикселя. Второй способ — отображение значения оптического потока как поле векторов. Пример работы метода Farneback [15] для двух кадров с отображением результата в изображение формата HSV и поле векторов показан на рисунке 10.



Рисунок 10 – Пример работы алгоритма Farneback [15]

2.3. Эвристический алгоритм

Для оценки “количество” движения вычисляются компоненты векторов оптического потока соседних кадров в видеофрагментах. Каждый вектор имеет две составляющие: магнитуду и угол поворота. Так как для оценки важны значения смещений, а не направлений, поэтому в алгоритме используются только магнитуды векторов. Чтобы оценить один переход между двумя кадрами, значения магнитуды нормируются от 0 до 1. С помощью некоторого порога, выбранного экспериментально, отсекаются пиксели с небольшими значениями (меньше 3 пикселей). Далее вычисляются: маска покрытия движения на изображении, отношение между площадью маски и площадью всего изображения. Данная процедура повторяется для всех соседних кадров одного видеофрагмента. В итоге получаем эвристическую оценку от 0 до 1, которая показывает, какая часть изображения покрывается движением объектов на протяжении видеоклипа.

Таким образом, алгоритм выбора видео выглядит следующим образом:

Входные данные:

V – видеофрагмент,
 θ – параметры для функции Optical Flow,
 τ – порог для магнитуды.

Выходные данные: значение от 0 до 1

BEGIN

```
    eval ← 0 # Инициализация значения оценки
    S ← |V| - 1 # Количество переходов между кадрами
    FOR FROM i=0 TO S DO # Вычисление по соседним кадрам
        I_1 ← V[i] # Предыдущий кадр
        I_2 ← V[i+1] # Следующий кадр
        magnitude ← OpticalFlow(I_1, I_2, θ) # Вычисление
        магнитуды смещений
        magnitude ← Normalization(magnitude) # Нормализация
        значения магнитуд
        mask ← Binarization(magnitude, τ) # Бинаризация
        изображения магнитуд
        eval ← eval + mask / N # Вычисление оценки для одного
        перехода
```

```
    RETURN eval/S  
END
```

2.4. Построение набора данных

Алгоритм выбора видео позволяет оценивать “количество” движения для создания обучающей выборки перед обучением методом самообучения нейронной сети. Для того чтобы определить какие элементы стоит выбирать для обучения, был проведен анализ различных видеофрагментов.

Существуют видеоклипы с различными типами ситуаций, где:

- объекты движутся на неподвижном фоне,
- объекты движутся на подвижном фоне,
- объекты движутся вместе с фоном.

Для оценки движения в каждом из трех описанных ситуации были вычислены значения алгоритма выбора видео. Результаты показали, что видеофрагменты, которые получили низкие значения метрики (ниже 10%), имеют близкую схожесть соседних кадров, такие элементы в выборке будут иметь похожие фрагменты почти во всех частях изображения, что плохо отражается при обучении методом [1]. В случае если результат алгоритма оказался достаточно большим (больше 90%), то, как правило, в таких ситуациях на видеофрагменте присутствуют резкие сдвиги, быстрые переходы частей картинки между кадрами. В остальных случаях каждый кадр будет изменяться с небольшими сдвигами фона и объектов. Примеры различных ситуаций со значением алгоритма показаны на рисунке 11.

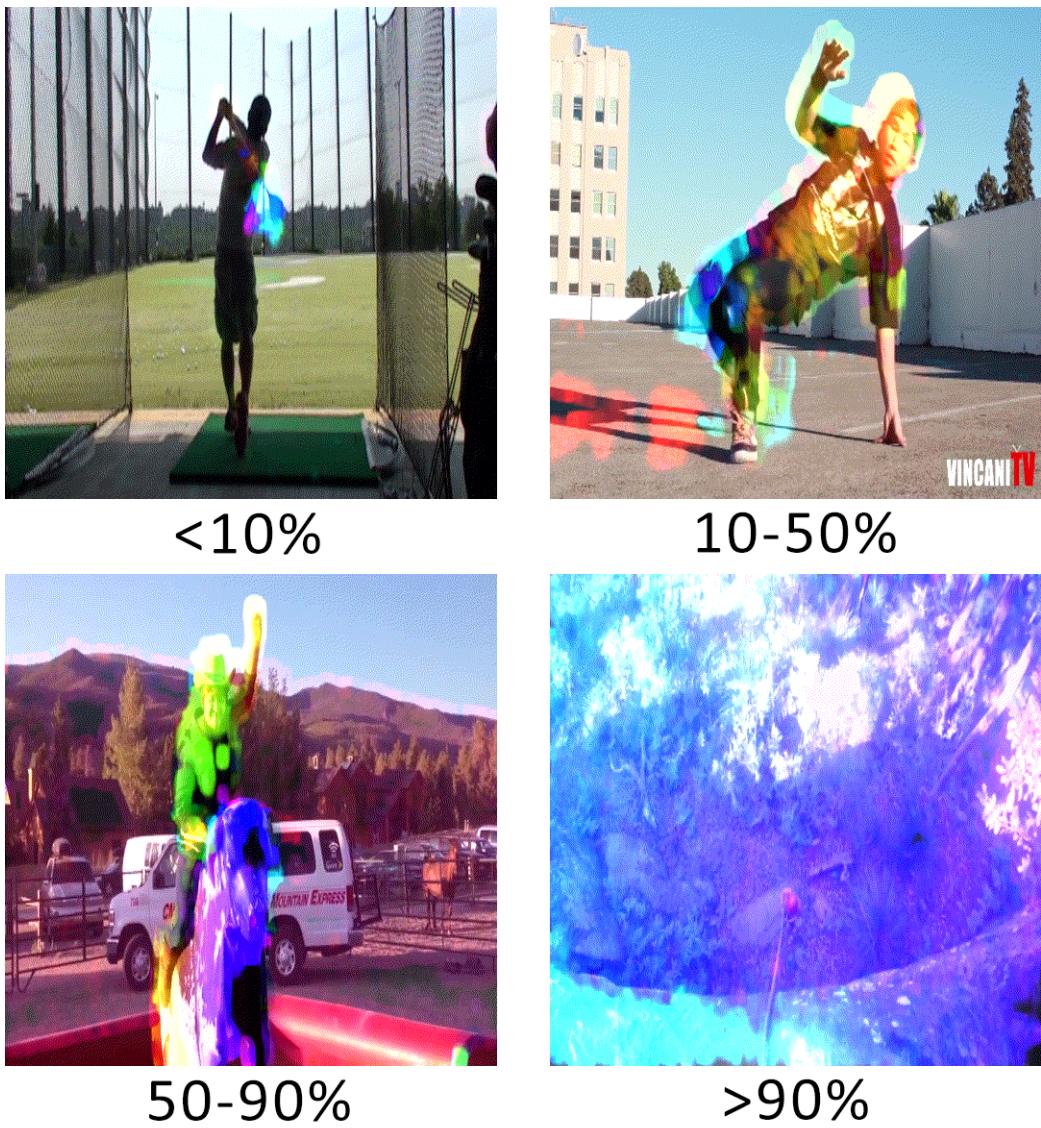


Рисунок 11 – Примеры различных ситуаций на видеофрагментах

Предварительно было изучено влияние использования элементов разного “количество” движения на процесс обучения. График обучения, где в обучающей выборке 1626 и 1698 видеофрагментов с разными диапазонами значений, показан на рисунке 12. Такой результат показал, что для элементов, у которых в среднем небольшие сдвиги между кадрами, значение точности оказалось меньше, чем у элементов с большими. В качестве элементов для основного набора данных были взяты те, которые принимают значения от 50 до 90 и большая часть от 10 до 50, так как такие элементы труднообучаемые. Итоговый набор данных для обучения составил около 4-5% данных от основного.

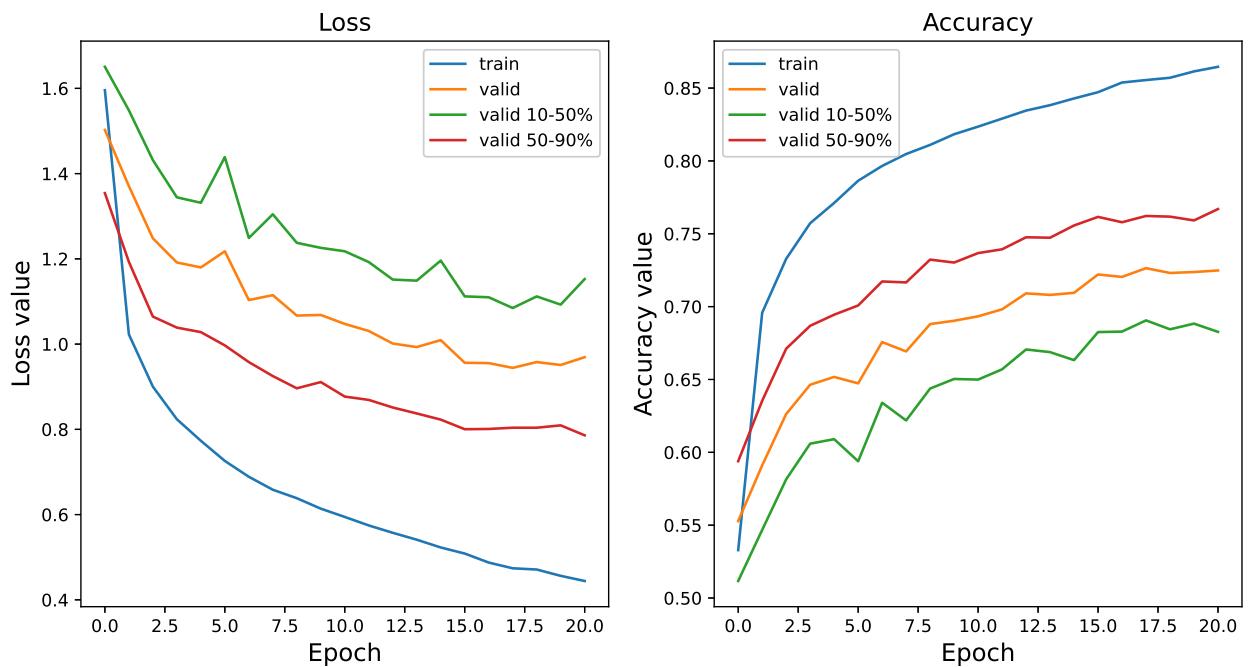


Рисунок 12 – График обучения элементов с разным “количество” движение

3. Описание экспериментов

В данной главе будет описана информация об используемых наборах данных для обучения и тестирования метода самообучения, о технических деталях в реализации метода «Contrastive Random Walk» [1], о параметрах обучения моделей и информация о результатах работы метода и алгоритма выбора видео для создания обучающей выборки, оцененные с помощью метрик качества.

3.1. Наборы данных

В основной работе для обучения методом самообучения «Contrastive Random Walk» [1] использовался набор данных Kinetics-400 [2]. Это один из крупномасштабных наборов данных для распознавания действий человека на видео. В нем содержится 400 классов действий человека, в каждом из которых 400-1150 клипов. Все элементы в выборке представляют собой видеофрагменты длительностью около 10 секунд, взятые по отдельности из уникальных видеороликов. Видеоклипы были взяты с видеохостинга YouTube [11] и имеют различные разрешения и частоты кадров. Набор данных состоит из 306245 видеоклипов и разделен на три части:

- видеоклипы для обучения,
- видеоклипы для валидации гиперпараметров и контролирования процесса обучения,
- видеоклипы для тестирования модели.

На обучающую выборку выделено от 62.5% до 87% от общего количества данных в каждом классе. Статистика разбиения на обучающую, валидационную и тестовую выборки приведена в таблице 5.

Таблица 5 – Статистика разбиения элементов в каждом классе набора данных Kinetics-400 [12]

Обучающая	Валидационная	Тестовая
250-1000	100	50

Как видно из таблицы 5, если учитывать такое количество данных для всех 400 классов, то обучающая выборка состоит из большого количества элементов. При таком раскладе в процессе обработки данных перед обучением может возникнуть проблема с нехваткой или переполнением памяти, поэтому из этого набора были выбраны только те элементы, которые отобрал алгоритм выбора видео. Структура организации данных состоит из следующих элементов:

- метка класса (действие человека),
- URL адрес видео на странице YouTube [11],
- время начала клипа для обрезания,
- время конца клипа для обрезания,
- метка типа элемента (train, valid, test).

После обучения модели самообучения нейронной сети необходимы данные, с помощью которых можно оценить качество работы. Для этого был использован набор данных DAVIS-2017 (Densely Annotated Video Segmentation) [19]. Это общедоступный набор данных, созданный для тестирования алгоритмов (англ. benchmark), специально разработанный для задачи сегментации объектов видеоизображений. В этой работе так же, как и в работе [1], этот набор данных использовался для предсказания маски видеообъектов. Статистика по размеру самого DAVIS-2017 [19] представлена в таблице 6.

Таблица 6 – Статистика по набору данных DAVIS-2017 [19]

	Обучающая	Валидацион-ная	Тестовая	Общее количество
Количество клипов	60	30	30	120
Количество кадров	4219	2023	2037	8279

Продолжение таблицы 6

	Обучающая	Валидацион- ная	Тестовая	Общее количество
Среднее количество кадров в клипах	70.3	67.4	67.9	69
Общее количество объектов	138	59	89	286
Среднее количество объектов в клипах	2.30	1.97	2.97	2.38

Авторы этого набора данных предлагают решить задачу сегментации двумя способами. Один из способов — это обучение с частично размеченными данными (англ. semi-supervised learning). В этом случае для каждого видеоизображения из обучающей и валидационной выборок есть размечена маска, но в тестовой присутствует лишь маска первого кадра видео. Идея в том, чтобы обучить нейронную сеть, например, методом самообучения для того, чтобы получить модель, которая способна по первому кадру определять маску для последующих с помощью векторных представлений изображений.

Следующая задача, где используются векторные представления изображений с использованием видеопоследовательностей, это отслеживание человеческих поз (англ. pose tracking) в различных ситуациях. Для такой задачи существует набор данных JHMDB (Joint-annotated Human Motion Data Base) [20], который имеет разметку 15 ключевых точек для отслеживания частей тела. Такая база данных содержит 5100 клипов, 51 уникальных человеческих

действий на видео из интернета или кинофильмов. Особенность такого набора данных в том, что была произведена процедура очистки элементов, которые описывали неоднозначные ситуации. В итоге конечная выборка содержит 36-55 клипов на каждый класс, каждый клип содержит 15-40 кадров, всего 31838 размеченных видеоизображений. Для разбиения на обучающую, валидационную и тестовую выборки авторы [20] предложили три различных способа, показанные в таблице 7. Для выполнения протокола оценивания использовался способ 1.

Таблица 7 – Способы разбиения данных в наборе JHMDB [20]

Типы разбиения	Обучающая + Валидационная	Тестовая
Способ 1	660	268
Способ 2	658	270
Способ 3	663	265

Следующий способ оценки работы полученной модели была рассмотрена задача сегментации частей тела человека. Для этого использовался набор данных VIP (Video Instance Parsing) [21]. В данном случае требуется подробно определить маску для 20 классов. Сам набор данных содержит 404 видеопоследовательностей, где на обучающую выборку отведено 304, а остальные разделены на валидационную и тестовую. Каждый видеофрагмент дискретизирован с низкой частотой шага по сравнению с остальными наборами данных. Всего присутствует 10100 размеченных кадров, где для каждого видео отведено 25 последовательностей.

3.2. Технические детали

Основной язык программирования для реализации алгоритмов и функций в данной работе это Python. Данный язык программирования по сравнению с другими популярными на сегодняшний день является очень простым, удобным, выразительным и лаконичным. Благодаря ему можно с минималь-

ными затратами по времени и сил создавать сложные алгоритмы. К тому же, Python очень хорошо сочетается с быстрыми высокоуровневыми языками программирования C/C++. Это означает, что все инструменты, которые были реализованы на них, можно использовать для вычисления сложных операций, но при этом пользоваться ими с помощью простого программного кода. Поэтому данный язык является самым популярным в машинном обучении.

Для работы с моделями глубокого обучения была выбрана библиотека PyTorch [22]. Благодаря такому инструменту можно создавать архитектуры нейронных сетей с уже готовыми реализованными операциями слоев. Также присутствует возможность вычислять градиенты автоматически с помощью динамических вычислительных графов. Тогда при реализации модели нет необходимости описывать алгоритм обратного распространения ошибки, все вычисления будут происходить автоматически.

В настоящее время без графических процессоров нельзя за приемлемое время обучать глубокие архитектуры нейронных сетей. Один из возможных сервисов для использования видеопамяти с технологией параллельных вычислений CUDA это Google Colaboratory [23]. Данный сервис позволяет упрощенным способом пользоваться технологиями для исследований в области машинного обучения. Здесь можно получить различные графические процессоры, сами характеристики представлены в таблице 8.

Таблица 8 – Характеристики графических процессоров в облачном сервисе Google Colaboratory [23]

	Tesla K80	Tesla T4	Tesla P4	Tesla P100
Дата выпуска видеокарты	17 ноября 2014	13 сентября 2018	13 сентября 2016	20 июня 2016
Тактовая частота ядра видеокарты	562 MHz	585 MHz	886 MHz	1190 MHz

Продолжение таблицы 8

	Tesla K80	Tesla T4	Tesla P4	Tesla P100
Количество операций с плавающей точкой	4.113 flops/s	8.1 flops/s	5.704 flops/s	4.763 flops/s
Размер видеопамяти видеокарты	12 GB	16 GB	8 GB	16 GB
Пропускная способность памяти	240.6 GB/s	320.0 GB/s	192.3 GB/s	732.2 GB/s

Скорость обучения нейронных сетей при вычислении математических операций зависит от параметра числа операций с плавающей точкой. Так же для эффективного вычисления параметров нейронной сети необходим большой размер видеопамяти, чтобы вычислять градиент сразу по нескольким элементам. В данной работе по характеристикам и результатам тестирования предпочтительнее оказалась видеокарта Tesla T4.

3.3. Параметры обучения

Видеофрагменты в наборе данных Kinetics-400 [12] использовались для получения небольших видеоклипов. Каждые отдельные отрезки длиной 4 кадра получались благодаря обрезанию одного видео (с длительностью 10 секунд), расстояния между которыми 8 пропущенных изображений. Все изображения были изменены до размера 256×256 , где каждый из них разбили на 49 фрагментов размера 64×64 с помощью шага окна в 32 пикселя. Для нормализации данных использовались средние значения цветовых каналов [0.4914, 0.4822, 0.4465] со значениями дисперсии [0.2023, 0.1994, 0.2010].

Для обучения нейронной сети использовались следующие значения гиперпараметров:

- шаг обучения = $1e-4$,
- инициализация весов Не [24],
- размер группы изображений при обучении = 6,
- количество эпох = 20,
- регуляризация Weight Decay = $1e-5$,
- метод оптимизации Adam [25],
- количество эпох Warm-Up = 1,
- размерность векторных представлений = 128.

Процесс обучения контролировался с помощью валидационной выборки. График обучения с использованием алгоритма и при стандартной схеме семплирования показан на рисунке 13.

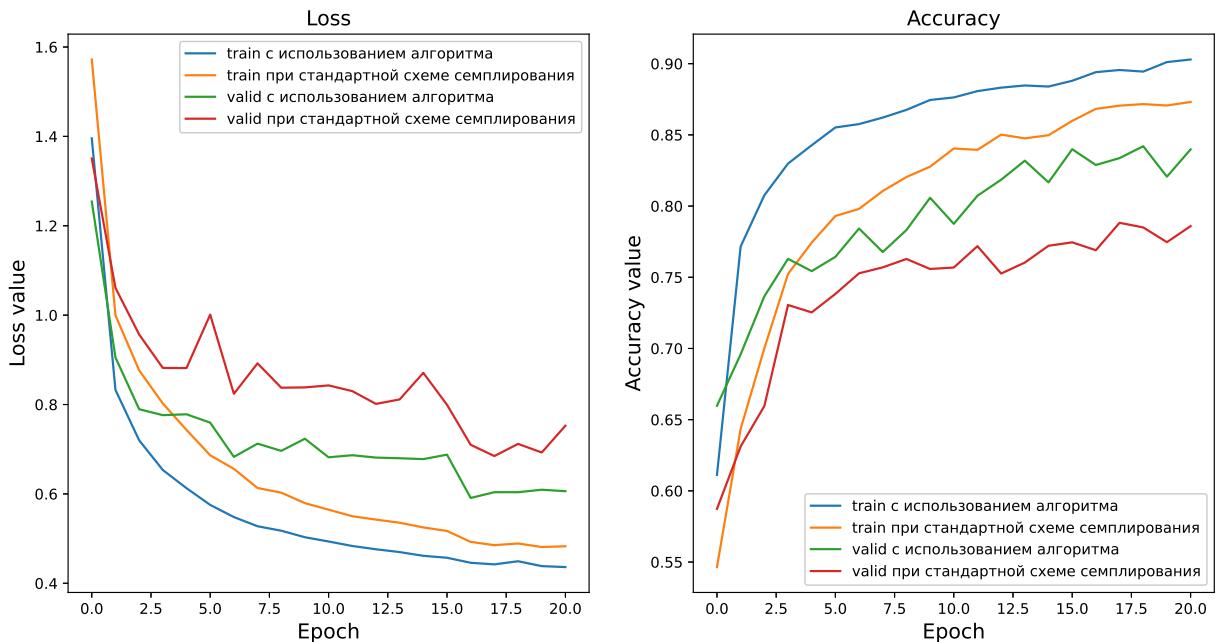


Рисунок 13 – График обучения метода «Contrastive Random Walk» [1]

3.4. Использовавшиеся метрики

Для оценки работы обученной модели использовался протокол оценивания как в статье [1] и [19]. С помощью алгоритма распространения меток

(англ. Label Propagation) с использованием k-ближайших соседей (англ. k-nearest neighbors) вычислялись маски последующих кадров элементов набора данных DAVIS-2017 [19]. Полученные маски объектов на изображении оценивались с помощью метрик \mathcal{J} и \mathcal{F} .

Метрика \mathcal{J} (мера Жаккара) или IoU (Intersection-Over-Union) показывает, насколько точно алгоритм классифицирует каждый пиксель. Для вычисления используется значение пересечения пикселей между результатом сегментации модели и размеченной маской и их объединения:

$$\mathcal{J} = \frac{|M \cap G|}{|M \cup G|},$$

где M – результат сегментации модели и G – размеченная маска.

Метрика \mathcal{J} позволяет оценить точность сегментированных границ. Авторы [19] предложили область сегментации M рассматривать, как набор замкнутых контуров и вычислять *Precision* и *Recall* значения для итоговой метрики:

$$\mathcal{F} = \frac{2P_cR_c}{P_c + R_c}.$$

В наборе данных присутствуют различные классы, поэтому итоговые значения метрик будут вычислять как среднее значение по классам $\mathcal{J} \& \mathcal{F}_m, \mathcal{J}_m, \mathcal{J}_r, \mathcal{J}_d, \mathcal{F}_m, \mathcal{F}_r, \mathcal{F}_d$.

В задаче отслеживания позиции разных частей тела человека JHMDB [20] используется метрика процента корректности ключевых точек $PCK@k$, где k – значение порога нормализованного расстояния от размеченной ключевой точки до предсказанной. Сама метрика вычисляется с помощью расстояния L2, где используются при подсчете только те вершины, расстояния которых не превосходят заданный порог k :

$$PCK@k = \frac{\sum_{i=1}^n I(d_i < kD)}{n},$$

где d_i – L2 расстояние между i -ой предсказанной и размеченной ключевой точкой и D – диаметр части ключевой точки.

Для оценки модели в задачи сегментация частей человека VIP [21] используется метрика $mIoU$.

3.5. Результаты экспериментов

После реализации метода «Contrastive Random Walk» [1], создания обучающей выборки с использованием алгоритма выбора видео и обучения самой модели была произведена оценка полученной модели на различных задачах с помощью метрик качества. Предсказания для каждого набора данных получалось с помощью кодировщика и алгоритма распространения метки, который с помощью первого размеченного элемента определяет смещение разметки последующих кадров.

Предсказания масок сегментации набора данных DAVIS-2017 [19], используя векторные представления изображений, были получены благодаря алгоритму распространения меток. Разрешение каждого кадра не изменялось, использовался размер 480×480 . Параметры алгоритма распространения меток, с модификацией авторов [1], применялись следующие: $m = 5$, $k = 10$ и $r = 12$, где m – количество контекстных кадров, k – количество ближайших соседей и r – радиус каждой точки. Для оценки полученной модели использовались метрики $\mathcal{J} \& \mathcal{F}_m$, \mathcal{J}_m , \mathcal{J}_r , \mathcal{J}_d , \mathcal{F}_m , \mathcal{F}_r , \mathcal{F}_d , показанные в таблице 9.

Таблица 9 – Результаты экспериментов в сегментации изображений для набора данных DAVIS-2017 [19]

	$\mathcal{J} \& \mathcal{F}_m$	\mathcal{J}_m	\mathcal{J}_r	\mathcal{J}_d	\mathcal{F}_m	\mathcal{F}_r	\mathcal{F}_d
Обучающая (стандартная)	55.8	53.1	59.4	19.6	58.5	66.3	28.2
Валидационная (стандартная)	56.1	54.1	63.7	29.1	58.0	66.7	36.0

Продолжение таблицы 9

	$\mathcal{J} \& \mathcal{F}_m$	\mathcal{J}_m	\mathcal{J}_r	\mathcal{J}_d	\mathcal{F}_m	\mathcal{F}_r	\mathcal{F}_d
Обучающая с использованием алгоритма	58.9	56.1	65.4	18.5	61.6	69.5	27.9
Валидационная с использованием алгоритма	59.6	57.7	68.2	26.2	61.3	70.2	33.0

Оценить работу, полученной модели, можно с помощью сравнения результатов с предыдущими работами (показано в таблице 10). Так как тестовая выборка является закрытой в соревновании, авторы работ [1, 6, 8, 26] используют для оценки валидационную выборку.

Таблица 10 – Сравнение экспериментов в сегментации изображений для набора данных DAVIS-2017 [19] на валидационной выборке

	Набор данных	$\mathcal{J} \& \mathcal{F}_m$	\mathcal{J}_m	\mathcal{J}_r	\mathcal{F}_m	\mathcal{F}_r
VINCE [6]	Kinetics [12]	60.4	57.9	66.2	62.8	71.5
UVC [26]	Kinetics [12]	60.9	59.3	68.8	62.7	70.9
TimeCycle [8]	VLOG [27]	48.7	46.4	50.0	50.0	48.0
Оригинальные результаты [1]	Kinetics [12]	67.6	64.8	76.1	70.2	82.1
Реализации метода [1]	Kinetics [12]	56.1	54.1	63.7	58.0	66.7

Продолжение таблицы 10

	Набор данных	$\mathcal{J} \& \mathcal{F}_m$	\mathcal{J}_m	\mathcal{J}_r	\mathcal{F}_m	\mathcal{F}_r
Реализации метода [1] с ис- пользованием алгоритма	Kinetics [12]	59.6	57.7	68.2	61.3	70.2

В задаче отслеживания ключевых точек используется набор данных JHMDB [20], где каждый кадр изменяется до размера 320×320 . Для получения предсказания использовались параметры распространения метки: $m = 20, k = 10$ и $r = 12$. Чтобы получить оценку предсказанных точек, использовалась метрика $PCK@k$ для значений $k = 0.1, 0.2, 0.3, 0.4, 0.5$, которая описана в таблице 11.

Таблица 11 – Результаты экспериментов в отслеживании ключевых точек поз человека для набора данных JHMDB [20] при $r = 12$ с использованием алгоритма

	$PCK@0.1$	$PCK@0.2$	$PCK@0.3$	$PCK@0.4$	$PCK@0.5$
Обучающая	43.4	65.2	79.1	86.4	90.7
Валидационная	46.9	67.8	80.4	88.0	92.7

Радиус ключевых точек был изменен значением $r = 5$ так, как входной размер изображений был уменьшен почти в три раза, от 900×400 к 320×320 . Новые результаты с обновленными параметрами показаны в таблице 12.

Таблица 12 – Результаты экспериментов в отслеживании ключевых точек поз человека для набора данных JHMDB [20] при $r = 5$ с использованием алгоритма

	$PCK@0.1$	$PCK@0.2$	$PCK@0.3$	$PCK@0.4$	$PCK@0.5$
Обучающая	44.3	66.7	79.9	86.8	91.3
Валидационная	47.1	68.4	81.1	89.2	93.3

В предыдущих работах вычислялась метрика $PCK@k$ для значений $k = 0.1, 0.2$. Результаты сравнения, полученные в этой работе с результатами [1, 8, 26], показаны в таблице 13.

Таблица 13 – Результаты экспериментов в отслеживание ключевых точек поз человека для набора данных JHMDB [20]

	$PCK@0.1$	$PCK@0.2$
TimeCycle [8]	57.3	78.1
UVC [26]	58.6	79.6
Оригинальная работа [1]	59.3	84.9
Реализации метода [1], при $r = 5$	46.7	68.2
Реализации метода [1], при $r = 5$ с использованием алгоритма	47.1	68.4

В задаче сегментации 20 частей человека для оценки с помощью набора данных VIP [21] использовались кадры размером 560×560 . Как и в работе [1] использовались такие же параметры для алгоритма распространение меток: $m = 4$, $k = 10$ и $r = 12$. Дополнительно было посчитано значение метрики $mIoU$ при значениях $r = 3, 4, 5, 7, 12, 15$, результаты показаны в таблице 14.

Таблица 14 – Результаты экспериментов в сегментации частей человека для набора данных VIP [21] при различных значениях r , с использованием алгоритма

	$mIoU$
Результат при $r = 3$	31.9
Результат при $r = 4$	32.4
Результат при $r = 5$	32.6
Результат при $r = 7$	33.0
Результат при $r = 12$	33.2
Результат при $r = 15$	33.1

Оптимальное значение, при котором метрика $mIoU$ перестает падать $r = 12$. Сравнение результатов экспериментов работ [1, 8, 26] с результатами в этой работе с оптимальным значением r показаны в таблице 15. Данный эксперимент является самым сложным так, как в задаче требуется более тщательно предсказание маски по сравнению с набором данных DAVIS-2017 [19]. Итоговое сравнение показало, что разница полученных результатов с работой [26] оказалась невелика. При этом, благодаря алгоритму выбора видео, удалось получить прирост точности по сравнению с методом [8] и со стандартной схемой семплирования.

Таблица 15 – Сравнение экспериментов в сегментации частей человека для набора данных VIP [21]

	$mIoU$
TimeCycle [8]	28.9
UVC [26]	34.1
Оригинальные результаты [1]	38.6
Реализации метода [1] при $r = 12$	31.0
Реализации метода [1] при $r = 12$ с использованием алгоритма	33.2

ЗАКЛЮЧЕНИЕ

В ходе данной выпускной квалификационной работы был разработан алгоритм выбора видео для метода самообучения нейронной сети. Для достижения поставленной цели были выполнены следующие задачи:

- изучение последних методов по самообучающимся нейронным сетям,
- реализация эвристического алгоритма построения набора данных для обучения модели,
- реализация модели для получения векторного представления объектов с использованием последовательности кадров из отрезка видео,
- обучение модели на полученном наборе данных,
- проведение экспериментов, валидация гиперпараметров,
- анализ полученных результатов.

Разработанный алгоритм выбора видео позволил отобрать из набора данных Kinetics-400 [12] только те видеопоследовательности, которые показали оптимальное значение “количество” движения. Реализованная модель «Contrastive Random Walk» [1] с помощью библиотеки глубокого обучения PyTorch [22], обученная на наборе данных [12], позволила получить векторные представления изображений, которые были оценены с помощью метрик качества. Для задачи сегментации видеоизображений на наборе данных DAVIS-2017 [19] результаты статьи [8] были улучшены на 10.9%, 11.3%, 18.2%, 11.3%, 22.2% для метрик \mathcal{J} & \mathcal{F}_m , \mathcal{J}_m , \mathcal{J}_r , \mathcal{F}_m , \mathcal{F}_r соответственно. В определение ключевых точек набора данных JHMDB [20] разница полученных результатов $PCK@k$ для значений $k = 0.1, 0.2$ с оригинальной работой [1] и с прошлыми [8] и [26] 12.2%, 16.7%, 10.2%, 9.9% и 11.5%, 11.4% соответственно. Для одной из трудных задач сегментации частей человека на наборе данных VIP [21], по метрики $mIoU$ удалось получить прирост точности по сравнению с методом [8] на 1.9%. Для оригинальной

работы [1] и методом [26] разница полученных результатов составила 7.8% и 3.3% соответственно.

Таким образом, с помощью алгоритма выбора видео при небольшом размере обучающей выборке удалось добиться прироста показателей точности для некоторых работ и для сравнения со стандартной схемой семплирования, но полученная модель уступает результатам из работы 2020 года [1] при условии, что авторы использовали весь набор данных Kinetics-400 [12]. В дальнейшем алгоритм выбора видео может быть модифицирован, для того чтобы создать качественную обучающую выборку. Тогда можно получить прирост значений метрик качества при проведения экспериментов, демонстрирующие влияние разработанного семплирования на точность обученной нейронной сети.

За период выполнения выпускной квалификационной работы были приобретены компетенции по направлению 10.03.01 — «Информационная безопасность», представленные в таблице 16.

Таблица 16 – Приобретенные компетенции

Шифр компетенции	Расшифровка проверяемой компетенции	Расшифровка освоения компетенции
OK-1	способность использовать основы философских знаний для формирования мировоззренческой позиции	способность формулировать цель и задачи для выпускной квалификационной работы
OK-2	способность использовать основы экономических знаний в различных сферах деятельности	умение применять экономические знания в научных работах
OK-3	способность анализировать основные этапы и закономерности исторического развития России, ее место и роль в современном мире для формирования	навыки анализа современных подходов и информации в предметной области

Продолжение таблицы 16

Шифр компетенции	Расшифровка проверяемой компетенции	Расшифровка освоения компетенции
	гражданской позиции и развития патриотизма	
ОК-4	способность использовать основы правовых знаний в различных сферах деятельности	умение находить достоверную информацию в предметной области
ОК-5	способность понимать социальную значимость своей будущей профессии, обладать высокой мотивацией к выполнению профессиональной деятельности в области обеспечения информационной безопасности и защиты интересов личности, общества и государства, соблюдать нормы профессиональной этики	знания в области машинного обучения и ее роль в развитие методов защиты информации
ОК-6	способность работать в коллективе, толерантно воспринимая социальные, культурные и иные различия	умение в написании научных работ совместно с научным руководителем
ОК-7	способность к коммуникации в устной и письменной формах на русском и иностранном языках для решения задач межличностного и межкультурного взаимодействия, в том числе в сфере профессиональной деятельности	способность поддержки коммуникации в устной и письменной формах в процессе написание научной работы
ОК-8	способность к самоорганизации и	способность самостоятельно искать

Продолжение таблицы 16

Шифр компетенции	Расшифровка проверяемой компетенции	Расшифровка освоения компетенции
	самообразованию	нужную информацию для решения поставленных проблем
ОК-9	способность использовать методы и средства физической культуры для обеспечения полноценной социальной и профессиональной деятельности	способность в использование методов физических культур для поддержания процесса написания научной работы
ОПК-1	способность анализировать физические явления и процессы для решения профессиональных задач	навыки использование средств вычислительной техники
ОПК-2	способность применять соответствующий математический аппарат для решения профессиональных задач	применение алгоритмов глубокого обучения и методов компьютерного зрения
ОПК-3	способность применять положения электротехники, электроники и схемотехники для решения профессиональных задач	навыки работы с библиотекой PyTorch для решения профессиональных задач глубокого обучения
ОПК-4	способность понимать значение информации в развитии современного общества, применять информационные технологии для поиска и обработки информации	умение находить нужные, научные статьи для написания выпускной квалификационной работы
ОПК-5	способность использовать нормативные правовые акты в профессиональной деятельности	изучение нормативных актов РФ, регулирующие сферу информационной безопасности
ОПК-6	способность применять приемы оказания первой	умение работать с техническим

Продолжение таблицы 16

Шифр компетенции	Расшифровка проверяемой компетенции	Расшифровка освоения компетенции
	помощи, методы и средства защиты персонала предприятия и населения в условиях чрезвычайных ситуаций, организовать мероприятия по охране труда и технике безопасности	оборудованием, соблюдая технику безопасности
ОПК-7	способность определять информационные ресурсы, подлежащие защите, угрозы безопасности информации и возможные пути их реализации на основе анализа структуры и содержания информационных процессов и особенностей функционирования объекта защиты	знания в использование алгоритмов машинного обучения для решения задач информационной безопасности
ПК-1	способность выполнять работы по установке, настройке и обслуживанию программных, программно-аппаратных (в том числе криптографических) и технических средств защиты информации	была произведена валидация гиперпараметров в процессе обучения нейронной сети
ПК-2	способность применять программные средства системного, прикладного и специального назначения, инструментальные средства, языки и системы программирования для	были получены и проанализированы результаты работы итоговой нейронной сети

Продолжение таблицы 16

Шифр компетенции	Расшифровка проверяемой компетенции решения профессиональных задач	Расшифровка освоения компетенции
ПК-3	способность администрировать подсистемы информационной безопасности объекта защиты	навыки в написание программного кода на языке программирования Python
ПК-4	способность участвовать в работах по реализации политики информационной безопасности, применять комплексный подход к обеспечению информационной безопасности объекта защиты	способность реализовывать протоколы оценивания результатов выпускной квалификационной работы
ПК-5	способность принимать участие в организации и сопровождении аттестации объекта информатизации по требованиям безопасности информации	умение соблюдать протоколы оценивания результатов выпускной квалификационной работы
ПК-6	способность принимать участие в организации и проведении контрольных проверок работоспособности и эффективности применяемых программных, программно-аппаратных и технических средств защиты информации	навыки оценивания работы алгоритмов глубокого обучения для проверки качества
ПК-7	способность проводить анализ исходных данных для проектирования	навык в обоснование выводов о проделанной работе и соответствующих

Продолжение таблицы 16

Шифр компетенции	Расшифровка проверяемой компетенции	Расшифровка освоения компетенции
	подсистем и средств обеспечения информационной безопасности и участвовать в проведении технико-экономического обоснования соответствующих проектных решений	принятых проектных решений
ПК-8	способность оформлять рабочую техническую документацию с учетом действующих нормативных и методических документов	умение соблюдать требования ГОСТ-а при оформлении документов
ПК-9	способность осуществлять подбор, изучение и обобщение научно-технической литературы, нормативных и методических материалов, составлять обзор по вопросам обеспечения информационной безопасности по профилю своей профессиональной деятельности	способность выбирать подобные статьи для реализации собственного метода решения задачи выбора видео для метода самообучения нейронной сети
ПК-10	способность проводить анализ информационной безопасности объектов и систем на соответствие требованиям стандартов в области информационной безопасности	умения анализировать научные работы в предметной области машинного обучения для решения задач защиты информации
ПК-11	способность проводить эксперименты по заданной методике, обработку, оценку погрешности и	были проанализированы эксперименты выбора видео с использованием метода самообучения

Продолжение таблицы 16

Шифр компетенции	Расшифровка проверяемой компетенции	Расшифровка освоения компетенции
	достоверности их результатов	нейронной сети
ПК-12	способность принимать участие в проведении экспериментальных исследований системы защиты информации	умение понимать результаты выпускной квалификационной работы
ПК-13	способность принимать участие в формировании, организовывать и поддерживать выполнение комплекса мер по обеспечению информационной безопасности, управлять процессом их реализации	навыки написания программного кода на языке программирования Python с использованием библиотеки PyTorch
ПК-14	способность организовывать работу малого коллектива исполнителей в профессиональной деятельности	умение обсуждать процесс написания выпускной квалификационной работы с научным руководителем
ПК-15	способность организовать технологический процесс защиты информации ограниченного доступа в соответствии с нормативными правовыми актами и нормативными методическими документами Федеральной службы безопасности Российской Федерации, Федеральной службы по техническому и экспортному контролю	умение организовывать процесс защиты информации связанные с выпускной квалификационной работой

СПИСОК ЛИТЕРАТУРЫ

- 1) Space-Time Correspondence as a Contrastive Random Walk / Allan Jabri, Andrew Owens, Alexei A. Efros // In Advances in Neural Information Processing Systems. – 2020. C. 19545–19560.
- 2) The ImageNet Large Scale Visual Recognition Challenge / Li Fei-Fei, Jia Deng, Olga Russakovsky, Alex Berg, Kai Li // 2009 IEEE Conference on Computer Vision and Pattern Recognition – 2009. C. 248–255.
- 3) Unsupervised visual representation learning by context prediction / Doersch, A. Gupta, and A. A. Efros // In Proceedings of the IEEE International Conference on Computer Vision – 2015. C. 1422–1430.
- 4) Unsupervised representation learning by predicting image rotations / Spyros Gidaris, Praveer Singh, Nikos Komodakis // arXiv preprint arXiv:1803.07728: – 2018. – URL: <https://arxiv.org/abs/1803.07728> (дата обращения: 11.09.2021).
- 5) Unsupervised Learning of Visual Representations by Solving Jigsaw Puzzles / Mehdi Noroozi, Paolo Favaro // In European Conference on Computer Vision – 2016. C. 69–84.
- 6) Watching the World Go By: Representation Learning from Unlabeled Videos / Daniel Gordon, Kiana Ehsani, Dieter Fox, Ali Farhadi // arXiv preprint arXiv: 2003.07990: – 2020. – URL: <https://arxiv.org/abs/2003.07990> (дата обращения: 20.09.2021).
- 7) Noise-contrastive estimation: A new estimation principle for unnormalized statistical models / Gutmann, M., Hyvärinen, A. // Thirteenth International Conference on Artificial Intelligence and Statistics. – 2010. C. 297-304.
- 8) Learning correspondence from the cycleconsistency of time / Xiaolong Wang, Allan Jabri, and Alexei A Efros // In: The IEEE Computer Vision and Pattern Recognition Conference (CVPR). – 2019. C. 2566- 2576.

9) Dropout: A simple way to prevent neural networks from overfitting / Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov // The Journal of Machine Learning Research. – 2014. C. 1929–1958.

10) Deep residual learning for image recognition. / Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun // arXiv preprint arXiv:1512.03385: – 2015. – URL: <https://arxiv.org/abs/1512.03385> (дата обращения 20.09.2021).

11) Видеохостинг YouTube. / Steve Chen, Chad Hurley, Jawed Karim – 2005. – URL: <https://www.youtube.com> (дата обращения 28.10.2021).

12) Набор данных Kinetics-400. / Will Kay, Joao Carreira, Karen Simonyan, Brian Zhang, Chloe Hillier, Sudheendra Vijayanarasimhan, Fabio Viola, Tim Green, Trevor Back, Paul Natsev, Mustafa Suleyman, Andrew Zisserman // arXiv preprint arXiv:1705.06950: – 2017. – URL: <https://www.deepmind.com/open-source/kinetics> (дата обращения 10.11.2021).

13) Линейная алгебра / В. А. Ильин, Э. Г. Позняк. – Москва // Наука Физматлит, 1999. – 297 с. – ISBN 5-02-015238-8.

14) Pyramidal Implementation of the Affine Lucas Kanade Feature Tracker Description of the algorithm / Jean-Yves Bouguet // In Intel Corporation – 1999. – URL: <https://api.semanticscholar.org/CorpusID:9350588> (дата обращения 15.11.2021).

15) Two-Frame Motion Estimation Based on Polynomial Expansion / Gunnar Farnebäck // In: Proceedings of 15th International Conference on Pattern Recognition – 2003. – URL: <https://api.semanticscholar.org/CorpusID:15601477> (дата обращения 17.11.2021).

16) Determining Optical Flow / Berthold K. P. Horn, B. G. Schunck // Artificial Intelligence. – 1981. – URL: <https://api.semanticscholar.org/CorpusID:1371968> (дата обращения 19.11.2021).

- 17) SimpleFlow: A Non-Iterative, Sublinear Optical Flow Algorithm / Michael W. Tao, Jiamin Bai, Pushmeet Kohli, Sylvain Paris // Eurographics 2012 – 2012. – URL: <https://api.semanticscholar.org/CorpusID:1671978> (дата обращения 21.11.2021).
- 18) Библиотека OpenCV. – URL: <https://opencv.org> (дата обращения 23.01.2022).
- 19) DAVIS Challenge on Video Object Segmentation 2017 / Prof. F. Li, Prof. C. Xu, R. O'Reilly, Dr. M. Gygli // arXiv preprint arXiv:1704.00675 – 2017. – URL: <https://davischallenge.org/challenge2017/index.html> (дата обращения 05.02.2022).
- 20) Towards understanding action recognition / H. Jhuang and J. Gall and S. Zuffi and C. Schmid and M. J. Black // International Conf. on Computer Vision (ICCV). – 2013. C. 3192- 3199.
- 21) Adaptive Temporal Encoding Network for Video Instance-level Human Parsing / Qixian Zhou, Xiaodan Liang, Ke Gong, Liang Lin // Proc. of ACM International Conference on Multimedia (ACM MM) – 2018.
- 22) Библиотека для глубокого обучения. – URL: <https://pytorch.org> (дата обращения 10.03.2022).
- 23) Облачная среда Google Research. – URL: <https://colab.research.google.com> (дата обращения 02.04.2022).
- 24) Delving deep into rectifiers: surpassing humanlevel performance on imagenet classification / He, K., Zhang, X., Ren, S., Sun, J. // Proceedings of the IEEE International Conference on Computer Vision. - 2015. C. 1026–1034.
- 25) Kingma, D. Adam: A Method for Stochastic Optimization / D. Kingma, J. Ba. // arXiv preprint arXiv:1412.6980: – 2014. – URL: <https://arxiv.org/abs/1412.6980> (дата обращения: 18.05.2022).
- 26) Joint-task self-supervised learning for temporal correspondence / Xuetong Li, Sifei Liu, Shalini De Mello, Xiaolong Wang, Jan Kautz, and Ming-

Hsuan Yang // In Advances in Neural Information Processing Systems. – 2019.
C. 317–327.

27) From lifestyle vlogs to everyday interactions / Fouhey, David F and Kuo,
Wei-cheng and Efros, Alexei A and Malik, Jitendra // Proceedings of the IEEE
Conference on Computer Vision and Pattern Recognition. – 2018. C. 4991–5000.

ПРИЛОЖЕНИЯ

В этом разделе описан программный код на языке программирования Python, используя библиотеку PyTorch [22], алгоритма выбора видео, реализованного метода «Contrastive Random Walk» [1], дополнительные результаты экспериментов и ссылок на источники.

ПРИЛОЖЕНИЕ 1. Алгоритм выбора видео

Функция `get_opticalflow` возвращает оптический поток в виде изображения и значения магнитуды векторов смещения. Для вычисления среднего значения оптического потока по всем соседним кадрам используется `mean_opticalflow`. Основная функция алгоритма выбора видео это `eval_movement`. С помощью `eval_folder` вычисляется оценка для всех видеоклипов из одной директории.

```
import os
import numpy as np
import cv2
import matplotlib.pyplot as plt

def get_opticalflow(frame1, frame2) -> [np.array, np.array]:
    hsv = np.zeros_like(frame1)
    hsv[:, :, 1] = 255

    before_frame = cv2.cvtColor(frame1, cv2.COLOR_BGR2GRAY)
    next_frame = cv2.cvtColor(frame2, cv2.COLOR_BGR2GRAY)

    flow = cv2.calcOpticalFlowFarneback(before_frame, next_frame,
                                         None, 0.5, 3, 15, 3, 5, 1.2, 0)
    mag, ang = cv2.cartToPolar(flow[:, :, 0], flow[:, :, 1])

    hsv[:, :, 0] = ang * 180 / np.pi / 2
    hsv[:, :, 2] = cv2.normalize(mag, None, 0, 255, cv2.NORM_MINMAX)
    bgr = cv2.cvtColor(hsv, cv2.COLOR_HSV2BGR)

    return bgr, hsv[:, :, 2]

def mean_opticalflow(path: str, skip: int=8) -> float:
    cap = cv2.VideoCapture(path)
    count = cap.get(cv2.CAP_PROP_FRAME_COUNT)

    ret, frame1 = cap.read()
```

```

n = 0
movement_mean = 0
while(cap.isOpened()):
    cap.set(cv2.CAP_PROP_POS_FRAMES, (n + 1) * skip)
    ret, frame2 = cap.read()

    if ret:
        optical, magnit = get_opticalflow(frame1, frame2)
        movement_mean += eval_movement(magnitude=magnit, size=(256,
                                                               256))
        n += 1

    frame1 = frame2
else:
    break

cap.release()
return movement_mean / n

def eval_movement(magnitude: np.array, thresh_pixel: float=0.025,
                  size=None) -> float:
    if size is not None:
        magnitude = cv2.resize(magnitude, size)
    N = magnitude.shape[0] * magnitude.shape[1]
    magnitude = magnitude / 255.
    mask = magnitude > thresh_pixel
    return np.sum(mask) / N

def eval_folder(path: str, thresh_show: float=0.5, up: bool=False):
    files = os.listdir(path)
    for f in files:

        if f.split('.')[ -1] == 'mp4':
            full_path = path + '\\\\' + f
            eval_move = mean_opticalflow(path=full_path, skip=8)

            if eval_move > thresh_show and up:
                print(f, eval_move)
                continue
            if eval_move < thresh_show and not up:
                print(f, eval_move)
                continue

```

ПРИЛОЖЕНИЕ 2. Визуализация экспериментов

Сравнения полученных масок набора данных DAVIS-2017 [19], где в левом столбце результаты оригинальной статьи [1], а в правом данной работы, показаны на рисунке 14.

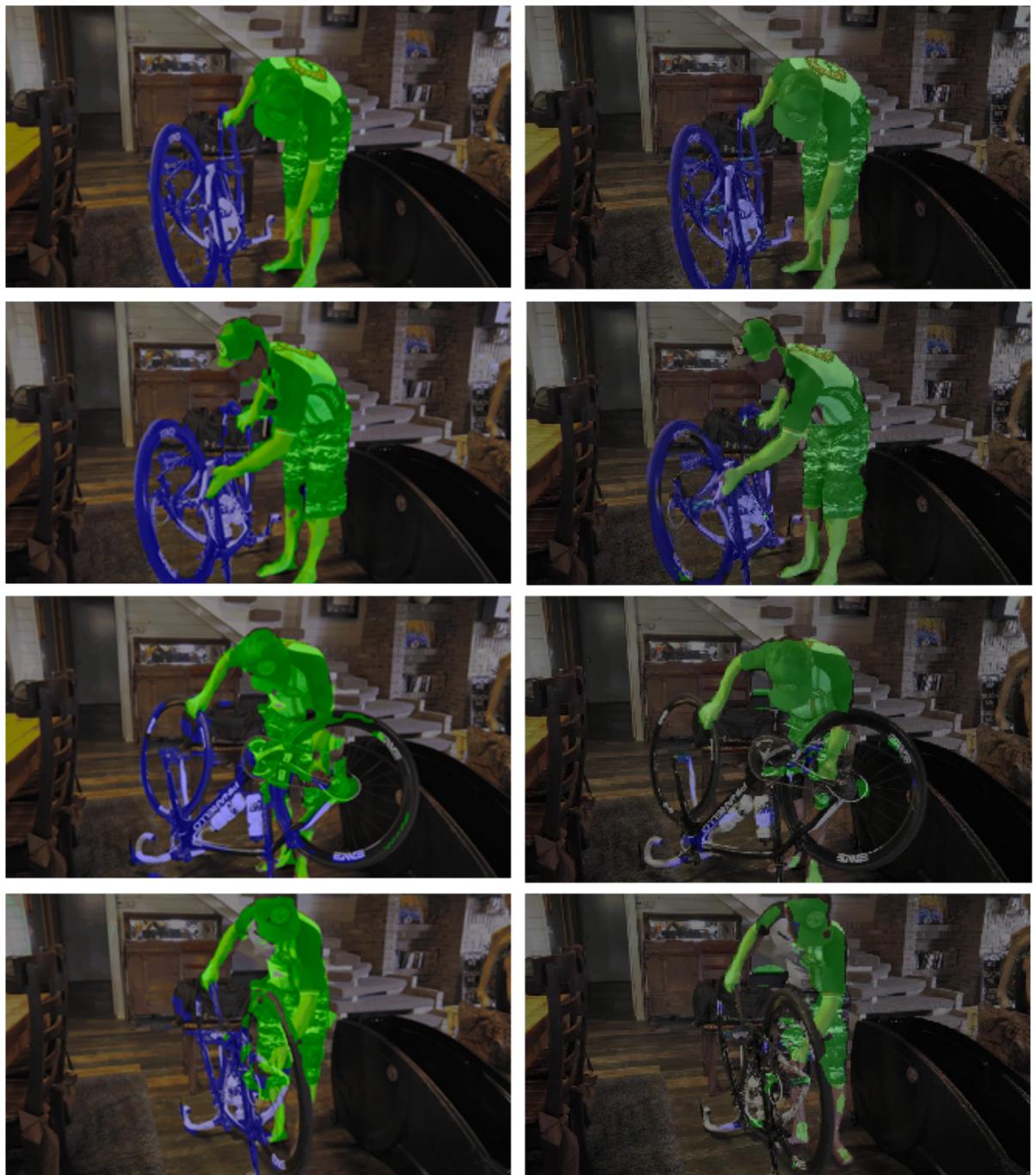


Рисунок 14 – Визуальное сравнение результатов сегментации набора данных DAVIS-2017 [19] с оригинальной статьи [1]

Сравнения полученных масок набора данных JHMDB [20], где в верхней строке результаты оригинальной статьи [1], а в нижней данной работы, показаны на рисунке 15.

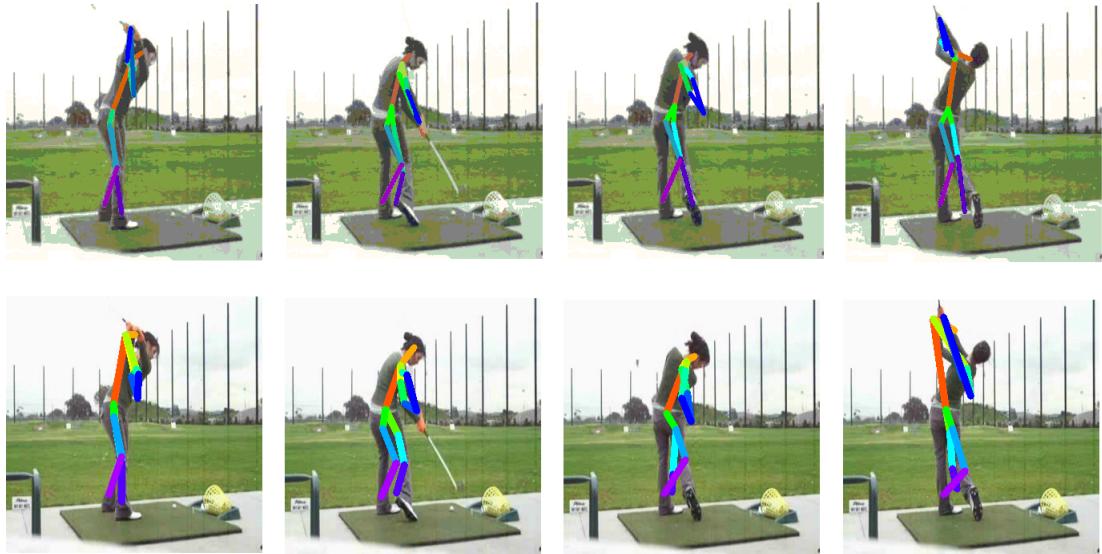


Рисунок 15 – Визуальное сравнение результатов сегментации набора данных JHMDB [20] с оригинальной статьи [1]

Сравнения полученных масок набора данных VIP [21], где в верхней строке результаты оригинальной статьи [1], а в нижней данной работы, показаны на рисунке 16.



Рисунок 16 – Визуальное сравнение результатов сегментации набора данных VIP [21] с оригинальной статьи [1]

ПРИЛОЖЕНИЕ 3. Архивы результатов экспериментов

Дополнительные результаты экспериментов для задачи сегментации объектов на видеоизображениях набора данных DAVIS-2017 [19] можно посмотреть в свободном доступе в онлайн облаке по ссылке <https://disk.yandex.ru/d/xwQzOeKc7S0kvA> или QR кодом на рисунке 17.



Рисунок 17 – QR-код на изображения с предсказанием маски для набора данных DAVIS-2017 [19]

Визуализация предсказанных ключевых точек из набора данных JHMDB [20] можно ознакомиться по ссылке <https://disk.yandex.ru/d/AN-Smcdy4mWDLg> или QR-кодом на рисунке 18.



Рисунок 18 – QR-код на изображения с предсказанием ключевых точек для набора данных JHMDB [20]

Сегментация 20 классов, где каждый класс — это определенная часть тела человека из набора данных VIP [21] можно посмотреть по ссылке <https://disk.yandex.ru/d/1komuLvUur4gRg> или QR-кодом на рисунке 19.



Рисунок 19 – QR-код на изображения с сегментацией частей человека для набора данных VIP [21]

ПРИЛОЖЕНИЕ 4. Эксперименты для отдельных классов

Значения метрик \mathcal{J}_m и \mathcal{F}_m отдельных классов из набора данных на валидационной выборке представлены в таблице 17.

Таблица 17 – Значения метрик \mathcal{J}_m и \mathcal{F}_m для отдельных классов из набора данных DAVIS 2017 [19] на валидационной выборке

Название класса	\mathcal{J}_m	\mathcal{F}_m	Название класса	\mathcal{J}_m	\mathcal{F}_m
bike-packing_1	36.9	54.9	kite-surf_2	25.2	26.9
bike-packing_2	67.9	72.9	kite-surf_3	53.5	73.1
blackswan_1	91.5	95.2	lab-coat_1	0.0	0.0
bmx-trees_1	10.2	31.2	lab-coat_2	0.0	0.0
bmx-trees_2	53.3	71.6	lab-coat_3	88.7	74.6
breakdance_1	64.2	64.5	lab-coat_4	87.1	75.3
camel_1	66.9	75.0	lab-coat_5	85.9	82.3
car-roundabout_1	80.2	71.1	libby_1	70.5	83.4
car-shadow_1	86.3	82.3	loading_1	85.8	73.5
cows_1	88.7	89.0	loading_2	36.0	42.0
dance-twirl_1	42.3	49.6	loading_3	70.7	71.8
dog_1	81.8	81.7	mbike-trick_1	56.2	69.8
dogs-jump_1	21.9	31.9	mbike-trick_2	52.8	56.6
dogs-jump_2	44.3	44.5	motocross-jump_1	34.9	40.3
dogs-jump_3	73.8	80.6	motocross-jump_2	22.6	21.6

Продолжение таблицы 17

Название класса	\mathcal{J}_m	\mathcal{F}_m	Название класса	\mathcal{J}_m	\mathcal{F}_m
drift-chicane_1	69.3	75.5	paragliding-launch_1	76.2	83.2
drift-straight_1	46.3	46.2	paragliding-launch_2	53.9	77.5
goat_1	78.7	70.0	paragliding-launch_3	2.1	8.7
gold-fish_1	78.0	73.1	parkour_1	58.4	61.8
gold-fish_2	67.2	73.6	pigs_1	71.1	72.8
gold-fish_3	80.9	83.6	pigs_2	51.1	63.1
gold-fish_4	75.6	77.9	pigs_3	87.3	75.6
gold-fish_5	83.8	77.9	scooter-black_1	19.1	50.8
horsejump-high_1	70.5	76.8	scooter-black_2	67.0	62.7
horsejump-high_2	64.9	84.2	shooting_1	26.5	36.1
india_1	60.6	51.5	shooting_2	69.7	57.5
india_2	47.9	44.2	shooting_3	67.3	82.6
india_3	55.5	52.5	soapbox_1	72.3	68.6
judo_1	76.7	78.9	soapbox_2	55.1	55.2
judo_2	54.8	59.5	soapbox_3	32.4	48.5
kite-surf_1	21.2	22.4			

Результаты, полученные в наборе данных VIP [21] на валидационной выборке представлены в таблице 18.

Таблица 18 – Значение метрики IoU для отдельных классов из набора данных VIP [21] на валидационной выборке

Название класса	IoU	Название класса	IoU
bike-packing_1	36.9	kite-surf_2	25.2
background	91.6	scarf	23.9
hat	26.5	skirt	50.8
hair	36.2	torso-skin	44.2
sun-glasses	19.5	face	44.1
upper-glasses	1.5	right-arm	18.2
dress	61.5	left-arm	19.8
coat	20.4	right-leg	15.2
socks	59.7	left-leg	11.1

Продолжение таблицы 18

Название класса	IoU	Название класса	IoU
pants	4.7	right-shoe	8.4
gloves	48.6	left-shoe	9.8

ПРИЛОЖЕНИЕ 5. Программный код метода самообучения

С полным репозиторием GitHub можно ознакомиться, перейдя по ссылке <https://github.com/Vlad15lav/video-understanding-crw.git> или QR-кодом на рисунке 20.



Рисунок 20 – QR-код на репозиторий GitHub, где реализована модель «Contrastive Random Walk» [1]

Модуль «train.py», в котором определена процедура оптимизации модели.

```
import argparse
import os
import pickle
import math
import numpy as np
import torch

from torch.utils.data.sampler import RandomSampler
from torch.utils.data import DataLoader
from torch.autograd import Variable

from tqdm import tqdm
from IPython import display

from data.datasets import Kinetics400
from data.augments import get_train_augmentation
```

```

from models.crw import CRW
from utils.util import get_cache_path, collate_fn, get_scheduler,
    adjust_learning_rate

def get_args():
    parser = argparse.ArgumentParser('Training-CRW')
    parser.add_argument('--data-path', type=str, help='path to dataset')
    parser.add_argument('--weight-path', type=str, default='state/crw', help='path for logs, weights training')
    parser.add_argument('--name-checkpoint', type=str, default='checkpoint', help='name checkpoint file pth')
    parser.add_argument('--cache-path', type=str, help='cache file dataset')

    parser.add_argument('--depth', type=int, default=18, help='depth resnet model')
    parser.add_argument('--head-depth', type=int, default=0, help='depth head')
    parser.add_argument('--pretrained', help='load imagenet weights', action="store_true")
    parser.add_argument('--cont-train', help='use last weights', action="store_true")
    parser.add_argument('--temperature', type=float, default=0.05, help='(temperature) shaping')
    parser.add_argument('--featdrop', type=float, default=0.1, help='dropout rate on maps')
    parser.add_argument('--edgedrop', type=float, default=0.0, help='dropout rate on A')

    parser.add_argument('--img-size', nargs='+', type=int, default=[256, 256], help='image size')
    parser.add_argument('--clip-len', default=4, type=int, metavar='N',
        help='number of frames per clip')
    parser.add_argument('--frame-skip', default=8, type=int, help='kinetics : fps | others : skip between frames')
    parser.add_argument('--augs', type=str, default='crop', help='select augmentation (crop, jitter, flip, grid)')
    parser.add_argument('--patch-size', nargs='+', type=int, default=[64, 64], help='patch size')
    parser.add_argument('--mean-norm', nargs='+', type=int, default=[0.4914, 0.4822, 0.4465], help='mean pixel')
    parser.add_argument('--std-norm', nargs='+', type=int, default=[0.2023, 0.1994, 0.2010], help='std pixel')
    parser.add_argument('--bs', type=int, default=8, help='batch size')
    parser.add_argument('--epoches', type=int, default=40, help='number of epoches')

    parser.add_argument('--lr', type=float, default=0.0002, help='init learning rate')

```

```

parser.add_argument(' -- final-lr', type=float, default
                   =0.00005, help='init□learning□rate')
parser.add_argument(' -- wup-lr', type=float, default=0.000001,
                   help='start□learning□rate□warm-up')
parser.add_argument(' -- warm-up', type=int, default=1, help='
                   number□of□epoches□with□warm-up')
parser.add_argument(' -- wd', ' -- weight-decay', type=float,
                   default=0.0005, help='weight□decay')
parser.add_argument(' -- adam', help='adam□optimizer', action="
                   store_true")
parser.add_argument(' -- n_work', type=int, default=2, help='
                   number□of□gpu')
parser.add_argument(' -- seed', type=int, default=841, help='
                   number□of□seed')
parser.add_argument(' -- device', type=str, default='cuda',
                   help='use□cpu□or□cuda')

args = parser.parse_args()
return args

def train(model, train_loader, valid_loader, optimizer,
          lr_schedule, opt):
    train_loss, train_acc = [], []
    valid_loss, valid_acc = [], []

    # load checkpoing weights
    if os.path.exists(opt.weight_path) and opt.cont_train:
        checkpoint = torch.load(f'{opt.weight_path}/{opt.
                           name_checkpoint}.pth')
        model.load_state_dict(checkpoint['model'])
        optimizer.load_state_dict(checkpoint['optimizer'])
        print('weights□loaded!')

    if os.path.exists(f'{opt.weight_path}/log_training.pickle'):
        f_log = open(f'{opt.weight_path}/log_training.pickle', 'rb')
        obj = pickle.load(f_log)
        train_loss, train_acc, valid_loss, valid_acc, valid_loss,
        valid_acc = obj
        f_log.close()
        print('training□data□loaded!')

    print("Training□start . . .")
    train_size = len(train_loader)
    for epoch in range(len(train_loss), opt.epoches):
        display.clear_output(wait=True)

        # training
        model.train()
        loss_batch, acc_batch = [], []
        for i, clip in enumerate(train_loader):

```



```

# save last and best weights
checkpoint = {
    'model': model.state_dict(),
    'optimizer': optimizer.state_dict(),
    'epoch': epoch}
torch.save(
    checkpoint,
    os.path.join(opt.weight_path, 'best_checkpoint.pth'))

# save last and best weights
checkpoint = {
    'model': model.state_dict(),
    'optimizer': optimizer.state_dict(),
    'epoch': epoch}
torch.save(
    checkpoint,
    os.path.join(opt.weight_path, 'checkpoint.pth'))

# log history
lists = (train_loss, train_acc, valid_loss, valid_acc)
f_log = open(f'{opt.weight_path}/log_training.pickle', 'wb')
pickle.dump(lists, f_log)
f_log.close()

if __name__ == '__main__':
    opt = get_args()

    np.random.seed(opt.seed)
    torch.manual_seed(opt.seed)

    if not os.path.exists(opt.weight_path):
        os.makedirs(opt.weight_path)

    # get transforms for dataloader
    transform_train = get_train_augmentation(opt)

    # load cache of dataset
    cached = None
    if opt.cache_path:
        if not os.path.exists(opt.cache_path):
            os.makedirs(opt.cache_path)

        cache_path = get_cache_path(opt.cache_path)
        if os.path.exists(cache_path):
            trainset, _ = torch.load(cache_path)
            cached = dict(video_paths=trainset.video_clips.
                          video_paths,
                          video_fps=trainset.video_clips.video_fps,
                          video_pts=trainset.video_clips.video_pts)

```

```

# load dataset
trainset = Kinetics400(root=opt.data_path + '/train',
                       frames_per_clip=opt.clip_len,
                       step_between_clips=1,
                       transform=transform_train,
                       extensions=( 'mp4' ),
                       frame_rate=opt.frame_skip,
                       _precomputed_metadata=cached)
validset = Kinetics400(root=opt.data_path + '/valid',
                       frames_per_clip=opt.clip_len,
                       step_between_clips=1,
                       transform=transform_train,
                       extensions=( 'mp4' ),
                       frame_rate=opt.frame_skip,
                       _precomputed_metadata=None)

# save cache dataset
if cached is None and cache_path:
    trainset.transform = None
    torch.save((trainset, opt.data_path), cache_path)
    trainset.transform = transform_train

# create dataloader
train_sampler = RandomSampler(trainset)
if os.path.exists(opt.weight_path) and opt.cont_train:
    checkpoint = torch.load(f'{opt.weight_path}/{opt.name_checkpoint}.pth')
    train_sampler = checkpoint['state_sampler']

train_loader = DataLoader(trainset, batch_size=opt.bs,
                          sampler=train_sampler,
                          num_workers=opt.n_work, pin_memory=True,
                          collate_fn=collate_fn)
valid_loader = DataLoader(validset, batch_size=opt.bs,
                         num_workers=opt.n_work, pin_memory=True,
                         collate_fn=collate_fn)

# create crw model
model = CRW(opt).to(opt.device)

# optimizer and sheduler
lr_schedule = get_sheduler(opt.lr, opt.final_lr, len(
    train_loader), opt.epoches, opt.warm_up, opt.wup_lr)

if opt.adam:
    optimizer = torch.optim.Adam(model.parameters(), lr=opt.lr,
                                 weight_decay=opt.wd)
else:
    optimizer = torch.optim.SGD(model.parameters(), lr=opt.lr,
                               momentum=opt.momentum, weight_decay=opt.wd)

```

```
train(model, train_loader, valid_loader, optimizer,
      lr_schedule, opt)
```

Модуль «crw.py», в котором реализован метод самообучения «Contrastive Random Walk» [1].

```
import torch
import torch.nn as nn
import torch.nn.functional as F

from models.resnet import get_resnet

EPS = 1e-20

class CRW(nn.Module):
    def __init__(self, opt):
        super(CRW, self).__init__()
        self.temperature = opt.temperature
        self.featdrop_rate = opt.featdrop
        self.edgedrop_rate = opt.edgedrop

        self.encoder = get_resnet(opt.depth)
        self.find_vector_dim()
        self.head = self.add_head(opt.head_depth)

        self.featdrop = nn.Dropout(p=self.featdrop_rate, inplace=False)

        self.criterion = nn.CrossEntropyLoss()

    def find_vector_dim(self):
        out = self.encoder(torch.zeros(1, 3, 256, 256).to(
            next(self.encoder.parameters()).device))
        self.encoder_out_dim = out.shape[1]
        self.map_scale = 256 // out.shape[-1]

    def add_head(self, depth_head=0):
        layers = []
        if depth_head >= 0:
            for _ in range(depth_head - 1):
                layers.append(nn.Linear(self.encoder_out_dim,
                                       self.encoder_out_dim))
                layers.append(nn.ReLU())
            layers.append(nn.Linear(self.encoder_out_dim, 128))
        return nn.Sequential(*layers)

    def forward(self, x):
        """
        x is (B, T, C*N, H, W)
        T - number of frames
        N - number of patches
        """
        x = self.featdrop(x)
        x = self.encoder(x)
        x = self.featdrop(x)
        x = self.head(x)
        x = self.featdrop(x)
        x = self.criterion(x, target)
```

```

"""
B, T, CN, H, W = x.shape
C, N = 3, CN // 3
x = x.transpose(1, 2).view(B, N, 3, T, H, W) # (B, N, C, T, H, W)

x = x.flatten(0, 1) # (BN, C, T, H, W)
x = x.permute(0, 2, 1, 3, 4).contiguous(). \
    view(-1, C, H, W) # (BNT, C, H, W)

# to nodes
maps = self.encoder(x) # (BNT, c, h, w)
c, h, w = maps.shape[-3:]
maps = maps.view(B*N, T, c, h, w).transpose(1, 2) # (BN, c, T, h, w)

if self.featdrop_rate > 0:
    maps = self.featdrop(maps)

q = maps.sum(-1).sum(-1) / (h * w) # (BN, c, T)
q = self.head(q.transpose(-1, -2)).transpose(-1, -2) # (BN, T, c)
q = F.normalize(q, p=2, dim=1) # l2 norm vector (BN, c, T)

# to embed patches (B x c x T x N) and maps (B, N, c, T, h, w)
q = q.view(B, N, q.shape[1], T).permute(0, 2, 3, 1)
maps = maps.view(B, N, *maps.shape[1:])

# transitions from t to t+1 (B x T-1 x N x N)
A = torch.einsum('bctn,bctm->btm', q[:, :, :-1], q[:, :, 1:]) / self.temperature

## Transition energies for palindrome graph
AA = torch.cat((A, torch.flip(A, dims=[1]).transpose(-1, -2)), dim=1)
AA[torch.rand_like(AA) < self.edgedrop_rate] = -1e10
At = torch.diag_embed(torch.ones((B, N))).to(A.device)

## Compute walks
for t in range(2 * T - 2):
    At = torch.bmm(F.softmax(AA[:, t]), dim=-1), At)

## Walk Loss
target = torch.arange(At.shape[-1])[None]. \
    repeat(At.shape[0], 1).view(-1).to(At.device)
logits = torch.log(At + EPS).flatten(0, -2)
loss = self.criterion(logits, target)
acc = (torch.argmax(logits, dim=-1) == target).float().mean()

```

```
    return q, loss, acc
```

Модуль «resnet.py», в котором реализован кодировщик.

```
import torch
import torch.nn as nn
import torchvision.models.resnet as torch_resnet

from torchvision.models.resnet import BasicBlock, Bottleneck
from torch.hub import load_state_dict_from_url

model_urls = { 'resnet18': 'https://download.pytorch.org/models/resnet18-5c106cde.pth',
               'resnet34': 'https://download.pytorch.org/models/resnet34-333f7ec4.pth',
               'resnet50': 'https://download.pytorch.org/models/resnet50-19c8e357.pth',
               'resnet101': 'https://download.pytorch.org/models/resnet101-5d3b4d8f.pth'
}

class ResNet(torch_resnet.ResNet):
    def __init__(self, *args, **kwargs):
        super(ResNet, self).__init__(*args, **kwargs)

    def modify(self, remove_layers=[], padding=''):
        # Set stride of layer3 and layer 4 to 1 (from 2)
        filter_layers = lambda x: [l for l in x if getattr(self, l) is not None]
        for layer in filter_layers(['layer3', 'layer4']):
            for m in getattr(self, layer).modules():
                if isinstance(m, torch.nn.Conv2d):
                    m.stride = tuple(1 for _ in m.stride)
        # Set padding (zeros or reflect, doesn't change much;
        # zeros requires lower temperature)
        if padding != '':
            for m in self.modules():
                if isinstance(m, torch.nn.Conv2d) and sum(m.padding) > 0:
                    m.padding_mode = padding

        # Remove extraneous layers
        remove_layers += ['fc', 'avgpool']
        for layer in filter_layers(remove_layers):
            setattr(self, layer, None)

    def forward(self, x):
        x = self.conv1(x)
        x = self.bn1(x)
        x = self.relu(x)
        x = x if self.maxpool is None else self.maxpool(x)
```

```

        x = self.layer1(x)
        x = self.layer2(x)
        x = x if self.layer3 is None else self.layer3(x)
        x = x if self.layer4 is None else self.layer4(x)

    return x

def resnet18():
    return ResNet(BasicBlock, [2, 2, 2, 2])

def resnet34():
    return ResNet(BasicBlock, [3, 4, 6, 3])

def resnet50():
    return ResNet(Bottleneck, [3, 4, 6, 3])

def resnet101():
    return ResNet(Bottleneck, [3, 4, 23, 3])

def get_resnet(depth, pretrained=False):
    if depth == 18:
        model = resnet18()
        model.modify(padding='reflect')
    elif depth == 34:
        model = resnet34()
    elif depth == 50:
        model = resnet50()
    elif depth == 101:
        model = resnet101()

    if pretrained:
        state_dict = load_state_dict_from_url(model_urls[f'resnet_{depth}'],
                                                progress=True)
        model.load_state_dict(state_dict)
    return model

```

Модуль «tools.py», в котором реализованы операции для работы с изображениями.

```

import numpy as np
import torch
import cv2

def color_normalize(x, mean, std):
    if x.size(0) == 1:
        x = x.repeat(3, 1, 1)
    for t, m, s in zip(x, mean, std):
        t.sub_(m)
        t.div_(s)
    return x

```

```

def to_numpy(tensor):
    if torch.is_tensor(tensor):
        return tensor.cpu().numpy()
    elif type(tensor).__module__ != 'numpy':
        raise ValueError("Cannot convert {} to numpy array"
                          .format(type(tensor)))
    return tensor

def to_torch(ndarray):
    if type(ndarray).__module__ == 'numpy':
        return torch.from_numpy(ndarray)
    elif not torch.is_tensor(ndarray):
        raise ValueError("Cannot convert {} to torch tensor"
                          .format(type(ndarray)))
    return ndarray

def im_to_numpy(img):
    img = to_numpy(img)
    img = np.transpose(img, (1, 2, 0)) # H*W*C
    return img

def im_to_torch(img):
    img = np.transpose(img, (2, 0, 1)) # C*H*W
    img = to_torch(img).float()
    return img

def resize(img, owidth, oheight):
    img = im_to_numpy(img)
    img = cv2.resize(img, (owidth, oheight) )
    img = im_to_torch(img)
    return

```

Модуль «util.py», в котором реализованы функции для процесса оптимизации, создания пакетов обучения и преобразования кэша.

```

import hashlib
import os
import math
import numpy as np

from torch.utils.data.dataloader import default_collate

def collate_fn(batch):
    """
    batch creator
    """
    batch = [d[0] for d in batch]
    return default_collate(batch)

def get_cache_path(filepath):
    """
    catch path creator
    """

```

```

"""
h = hashlib.sha1(filepath.encode()).hexdigest()
cache_path = os.path.join(filepath, h[:10] + ".pt")
cache_path = os.path.expanduser(cache_path)
return cache_path

def get_scheduler(lr, final_lr, batches, epoches, warmup_epochs
=10, warmup_lr=1e-5):
"""
create scheduler learning rate from iteration
"""
warmup_lr_schedule = np.linspace(warmup_lr, lr, batches *
warmup_epochs)
iters = np.arange(batches * (epoches - warmup_epochs))
cosine_lr_schedule = np.array([final_lr + 0.5 * (lr -
final_lr) *
(1 + math.cos(math.pi * t / (
batches * (epoches -
warmup_epochs))))])
for t in iters]:
lr_schedule = np.concatenate((warmup_lr_schedule,
cosine_lr_schedule))
return lr_schedule

def adjust_learning_rate(optimizer, lr_schedule, iteration):
"""
get learning rate from iteration
"""
for param_group in optimizer.param_groups:
param_group['lr'] = lr_schedule[iteration]

```

ПРИЛОЖЕНИЕ 6. Параметры метода самообучения

Параметры обученной модели при стаднартной схеме сэмплирования и с использованием алгоритма можно получить по ссылке <https://disk.yandex.ru/d/0UZX624xpNtdFg> или QR-кодом на рисунке 21.



Рисунок 21 – QR-код на параметры полученной модели ResNet-18 [10] и
полносвязанного слоя