

СОДЕРЖАНИЕ

СОДЕРЖАНИЕ	2
ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ	3
ВВЕДЕНИЕ	4
1 АНАЛИТИЧЕСКИЙ ОБЗОР	6
1.1 Классификация изображений	6
1.1.1 Виды классификации изображений	6
1.1.2 Методы классификации изображений	7
1.2 Сверточные нейронные сети	10
1.2.1 Принцип работы	10
1.2.2 Структура CNN	11
1.2.3 Параметры сверточной нейронной сети	15
1.2.3 Популярные архитектуры	18
1.2.4 Преимущества и недостатки	20
1.3 Гибридные квантовые нейросети	21
1.3.1 Основные определения	21
1.3.2 Квантовый слой	22
1.3.3 Интеграция	23
1.4 Квантовые схемы	23
1.4.1 Основные одно-кубитные операции	24
1.4.2 Основные многокубитные операции	26
1.4.3 Примеры реализации	27
1.5 Острый лимфобластный лейкоз	28
2 ОСНОВНАЯ ЧАСТЬ	30
2.1 Содержательная постановка задачи	30
2.2 Математическая постановка задачи	31
2.3 Средства разработки	31
2.4 Сведения о проведенных исследованиях и полученных результатах	32
ЗАКЛЮЧЕНИЕ	39
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	41
ПРИМЕЧАНИЕ	42

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ

CNN – сверточные нейронные сети

SVM – метод опорных векторов (Support Vector Machines)

MNIST - Modified National Institute of Standards and Technology

QKNN – квантово-классические нейронные сети

ИИ – искусственный интеллект

ВВЕДЕНИЕ

В последние годы искусственный интеллект достиг огромных успехов, и его возможности оказывают значительное влияние на различные сферы жизни. Нейросети активно используются не только в индустрии и искусстве, но и в таких областях, как медицина. Уже сегодня они помогают в диагностике заболеваний, например, в распознавании раковых клеток или обнаружении злокачественных лейкоцитов, которые могут быть причиной лейкемии.

С каждым днем нейросетевые модели демонстрируют все более высокие результаты за счет добавления новых слоев, увеличения датасетов, тестирования новых архитектур. Однако усложнение моделей приводит к увеличению времени их обучения, оно может достигать нескольких недель. Применение квантовых вычислений позволит уменьшить время обучения моделей за счет квантового параллелизма, а также повысит их эффективность.

Целью данного исследования является оценка эффективности квантово-классических нейронных сетей в задаче классификации изображений по сравнению с классическими моделями.

В ходе работы будут решены следующие задачи:

- Анализ возможностей применения квантово-классических нейронных сетей для задачи классификации изображений
- Изучение возможной архитектуры гибридной модели, сочетающей классические нейронные сети и квантовые вычисления.
- Анализ составляющих квантового слоя.

Научно-технический аппарат, используемый в работе, включает: методы линейной алгебры для работы с матрицами и векторами, теорию вероятностей для анализа результатов квантовых вычислений, а также основы квантовых вычислений (квантовые схемы и операторы). Для реализации модели были использованы библиотеки TensorFlow (для классической части нейронной

сети), PennyLane (для квантовых вычислений). Для обработки данных и визуализации результатов будут применены библиотеки NumPy и Matplotlib.

Решение поставленных задач позволит не только улучшить качество классификации изображений, но и расширить возможности квантового машинного обучения, открывая перспективы для дальнейших исследований в этой области.

1 АНАЛИТИЧЕСКИЙ ОБЗОР

1.1 Классификация изображений

Классификация изображения — это одна из задач компьютерного зрения, представляет из себя присваивание картинке определенных категорий или меток. То есть компьютер должен определить, что именно перед нами находится, имея лишь наборы чисел.

Возможности классификации изображений невероятно широки. В частности, этот метод активно используется в биологии и медицине. Уже сегодня с его помощью можно:

- Идентифицировать раковые клетки, что значительно ускоряет диагностику заболеваний.
- Классифицировать различные типы клеток, помогая учёным в их изучении.
- Обнаруживать злокачественные лейкоциты, вызывающие лейкемию, что играет важную роль в раннем выявлении болезни.

1.1.1 Виды классификации изображений

Существует множество подходов к классификации изображений, каждый из которых используется в зависимости от задачи и особенностей данных. Среди них можно выделить следующие:

- Бинарная классификация — самый простой метод, при котором изображение относят к одной из двух категорий. Например, определение наличия человека на фото (либо он есть, либо его нет).
- Многоклассовая классификация — позволяет распределять изображения между более чем двумя классами. Например, модель может классифицировать изображение как собаку, кошку или черепаху.
- Многомарковая классификация — даёт возможность присваивать изображению сразу несколько меток. Например, на фото могут

одновременно присутствовать цветы и дома, поэтому изображение получит обе метки.

- Иерархическая классификация – применяется, когда категории образуют многоуровневую структуру. Например, изображение сначала классифицируется как «млекопитающее», а затем уточняется до «кошки» или «собаки».
- Классификация подклассов – фокусируется на различении схожих объектов. Примером может служить распознавание различных видов птиц или пород собак, где небольшие различия играют ключевую роль.
- Классификация с нулевым выстрелом (Zero-Shot Learning) – используется для распознавания изображений, относящихся к категориям, с которыми модель ранее не сталкивалась.
- Классификация с малым количеством выстрелов (Few-Shot Learning) – применяется, когда для обучения доступно всего несколько примеров каждого класса, что делает задачу сложнее, но жизненно важной для работы с редкими или уникальными объектами.

1.1.2 Методы классификации изображений

Рассмотрим два наиболее известных метода классификации изображений: контролируемая (Supervised Machine Learning) и неконтролируемая классификация (Unsupervised Learning). Контролируемое машинное обучение отличается тем, что включает в себя обучение модели с использованием помеченных данных, где каждый вход поставляется с соответствующим правильным выводом. В то время как задача неконтролируемого машинного обучения - найти закономерности и отношения в данных без каких-либо предварительных знаний о значении данных.

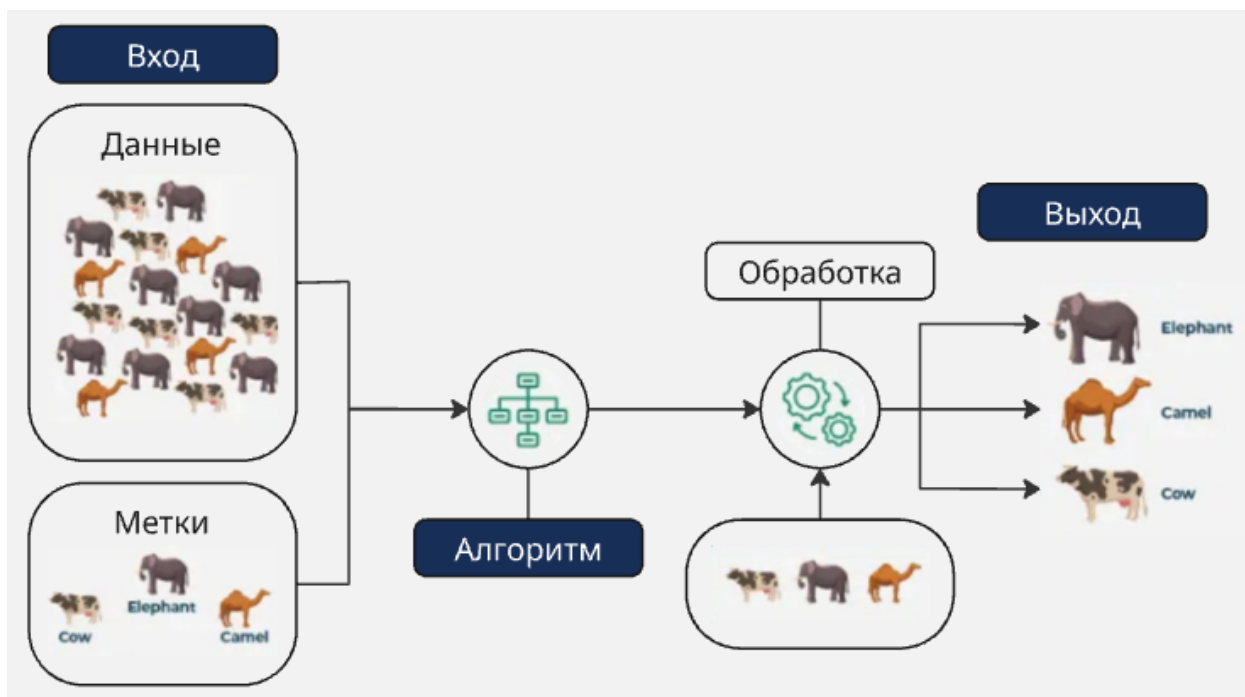


Рис. 1. Схема работы контролируемой классификации

Известные алгоритмы контролируемой классификации включают метод k-ближайших соседей, деревья решений, метод опорных векторов, случайные леса, логистическую регрессию, а также нейронные сети.

Одним из самых простых классификаторов является "метод ближайшего соседа". Во время обучения он запоминает все исходные данные, а потом пытается найти наиболее похожее на классифицируемое изображение. Для сравнения двух изображений сравнивают между собой пиксели фотографий, используют манхэттенское, евклидово или какое-нибудь другое расстояние. Однако этот метод неэффективен, обучение проходит быстро, а вот во время прогнозирования выполняется за линейное время $O(N)$ из-за того, что изображение сравнивается со всем датасетом. Еще одной проблемой является то, что расстояние между цветами пикселей не говорит и сходстве изображений.

Деревья решений (Decision Tree) – это метод классификации, который строит древовидную структуру, где каждый узел отвечает за разделение данных по определенному признаку, а конечные листья представляют собой

классы. Оно задаёт последовательность вопросов о данных (например, "Эта картинка чёрно-белая?", "Есть ли на ней круги?") и шаг за шагом приближает нас к правильному классу.

Метод опорных векторов (SVM) строит гиперплоскость, которая наилучшим образом разделяет классы, максимизируя разницу между ближайшими точками каждого класса. Они хорошо работают с высокоразмерными данными и малыми выборками, но требуют тщательной настройки гиперпараметров, долго обучаются на больших объемах данных.

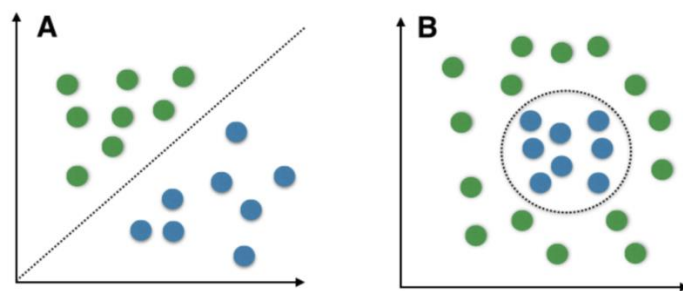


Рис. 2. Разделение классов методом опорных векторов

Случайные леса (Random Forest) представляют собой ансамбль деревьев решений, голосующих за итоговый класс. Этот метод отличается высокой точностью, но требует значительных вычислительных ресурсов, а также долго обучается при увеличении числа деревьев.

Логистическая регрессия, также известная как бинарная классификация, использует сигмоидную функцию для предсказания вероятности принадлежности объекта к определенному классу.



Рис. 3. Схема работы неконтролируемой классификации

Тем временем неконтролируемая классификация работает с непомеченными данными. Сначала они проходят стадию интерпретации, алгоритм пытается сгруппировать данные на основе схожих признаков, и уже потом модель обучается при помощи кластеризации, метода, который группирует данные в группы на основе их сходства.

Среди широкого спектра методов классификации изображений сверточные нейронные сети являются революционным решением для задач компьютерного зрения. CNN широко используются как в контролируемых, так и в неконтролируемых задачах классификации изображений. О них подробнее написано в следующем подпункте.

1.2 Сверточные нейронные сети

Сверточная нейронная сеть (англ. *convolutional neural network, CNN*) — специальная архитектура нейронных сетей, изначально нацеленная на эффективное распознавание изображений. Такие нейросети хорошо улавливают локальный контекст. Например, если сеть «распознала» часть какого-то объекта в одном пикселе, то вероятность того, что соседние пиксели тоже принадлежат этому объекту, будет высокой. Ее используют для решения ряда задач: сегментации изображений, детекции, распознавания видео, задачи идентификации и так далее.

1.2.1 Принцип работы

Сверточная нейросеть анализирует изображение, представляя его в виде многомерных массивов чисел (тензоров). В случае цветных изображений RGB каждый пиксель закодирован в виде трех целых чисел (от 0 до 255), отражающих интенсивность красного, зелёного и синего цветов.

Работа сверточной нейросети напоминает воронку: сначала анализируется общее изображение, а затем внимание сосредотачивается на его деталях. Для этого на каждом слое выполняются специфические операции,

ключевой из которых является свёртка. Сверточные фильтры позволяют выделять важные особенности изображения, такие как границы, текстуры и формы, отбрасывая незначительные детали.

После свёрточного слоя идёт слой пулинга. Его задача — уменьшить размерность данных, сохранив наиболее важные признаки. Из признаков, которые выделил свёрточный слой, выбирает самые важные, а несущественные удаляет. Это помогает нейросети работать быстрее и предотвращает переобучение. Например, после нескольких слоев обработки сеть может сначала обнаружить контуры объектов, затем — их форму, а на более глубоких слоях — распознать конкретные элементы, такие как глаза кошки или её уши.

На первых слоях сеть анализирует простые признаки, например, перепады яркости и границы объектов. Постепенно выделяются более сложные формы, такие как круги и кривые линии. Чем глубже слой сети, тем более высокоуровневые характеристики извлекаются.

На последних слоях сложные признаки передаются в полносвязный слой, который выполняет финальную классификацию и определяет, что именно изображено на картинке — кошка, сова или другой объект.

1.2.2 Структура CNN

Сверточные нейронные сети состоят из нескольких слоёв, вот ее основные элементы:

- Сверточный слой (англ. convolutional layer)

Свертка - операция над парой матриц A (размера $n_x \times n_y$) и B (размера $m_x \times m_y$), результатом которой является матрица $C=A*B$ размера $(n_x - m_x + 1) \times (n_y - m_y + 1)$. Каждый элемент результата вычисляется как скалярное произведение матрицы B и некоторой

подматрицы A такого же размера (подматрица определяется положением

$$\text{элемента в результате). То есть: } C_{i,j} = \sum_{u=0}^{m_x-1} \sum_{v=0}^{m_y-1} A_{i+u,j+v} B_{u,v}$$

На примере ниже можно увидеть, как матрица B «двигается» по матрице A, и в каждом положении считается скалярное произведение матрицы B и той части матрицы A, на которую она сейчас наложена. Получившееся число записывается в соответствующий элемент результата. Логический смысл свертки такой - чем больше величина элемента свертки, тем больше эта часть матрицы A была похожа на матрицу B (похожа в смысле скалярного произведения). Поэтому матрицу A называют изображением, а матрицу B - фильтром или образцом.

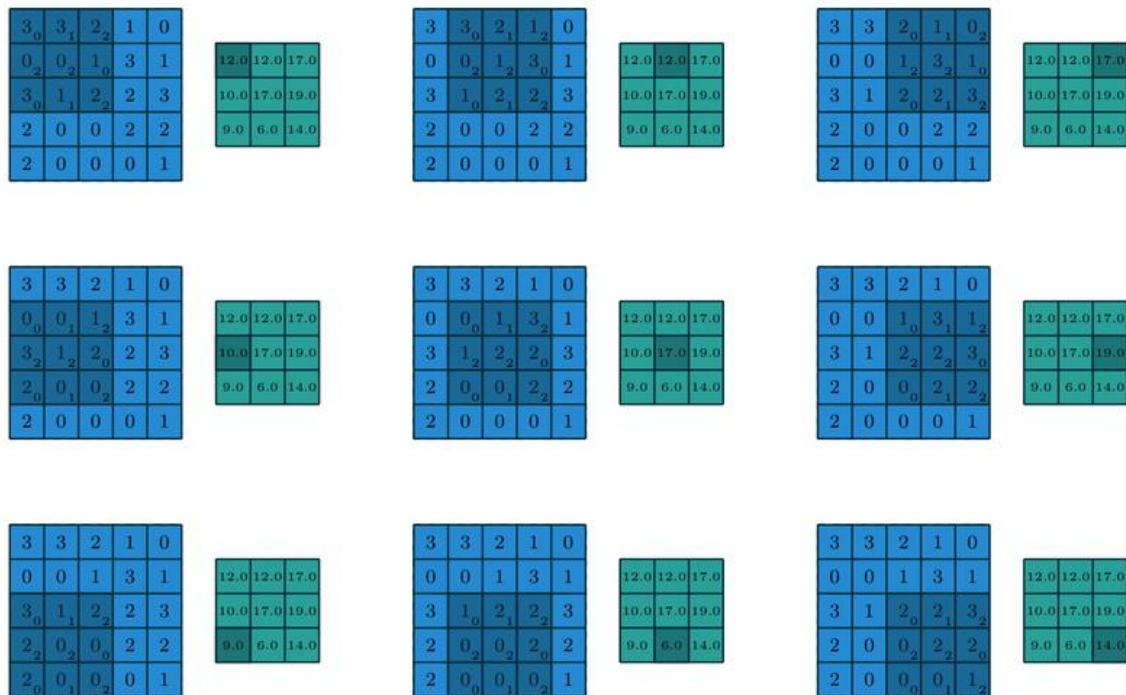


Рис.4. Пример свертки двух матриц с дополнением нулями и сдвигом 2

В одном сверточном слое может быть несколько сверток. В этом случае для каждой свертки на выходе получится своё изображение. Например, если

вход имел размерность $w \times h$, а в слое было n сверток с ядром размерности $k_x \times k_y$, то выход будет иметь размерность $n \times (\omega - k_x + 1) \times (h - k_y + 1)$

Можно заметить, что применение операции свертки уменьшает изображение. Также пиксели, которые находятся на границе изображения участвуют в меньшем количестве сверток, чем внутренние. В связи с этим в сверточных слоях используется дополнение изображения (padding). Выходы с предыдущего слоя дополняются пикселями так, чтобы после свертки сохранился размер изображения. Такие свертки называют одинаковыми (same convolution), а свертки без дополнения изображения называются правильными (valid convolution). Среди способов, которыми можно заполнить новые пиксели, можно выделить следующие:

- zero shift: 00[ABC]00
- border extension: AA[ABC]CC
- mirror shift: BA[ABC]CB
- cyclic shift: BC[ABC]AB

Еще одним параметром сверточного слоя является сдвиг (англ. stride). Хотя обычно свертка применяется подряд для каждого пикселя, иногда используется сдвиг, отличный от единицы — скалярное произведение считается не со всеми возможными положениями ядра, а только с положениями, кратными некоторому сдвигу s . Тогда, если вход имел размерность $w \times h$, а ядро свертки имело размерность $k_x \times k_y$ и использовался сдвиг s , то выход будет иметь размерность $\left\lfloor \frac{\omega - k_x}{s} + 1 \right\rfloor \times \left\lfloor \frac{h - k_y}{s} + 1 \right\rfloor$

Свертка трехмерного входа с трехмерным ядром происходит аналогично, просто скалярное произведение считается еще и по всем слоям изображения. Например, для усреднения информации о цветах исходного изображения, на первом слое можно использовать свертку размерности $3 \times w \times h$. На выходе такого слоя будет уже одно изображение (вместо трёх).

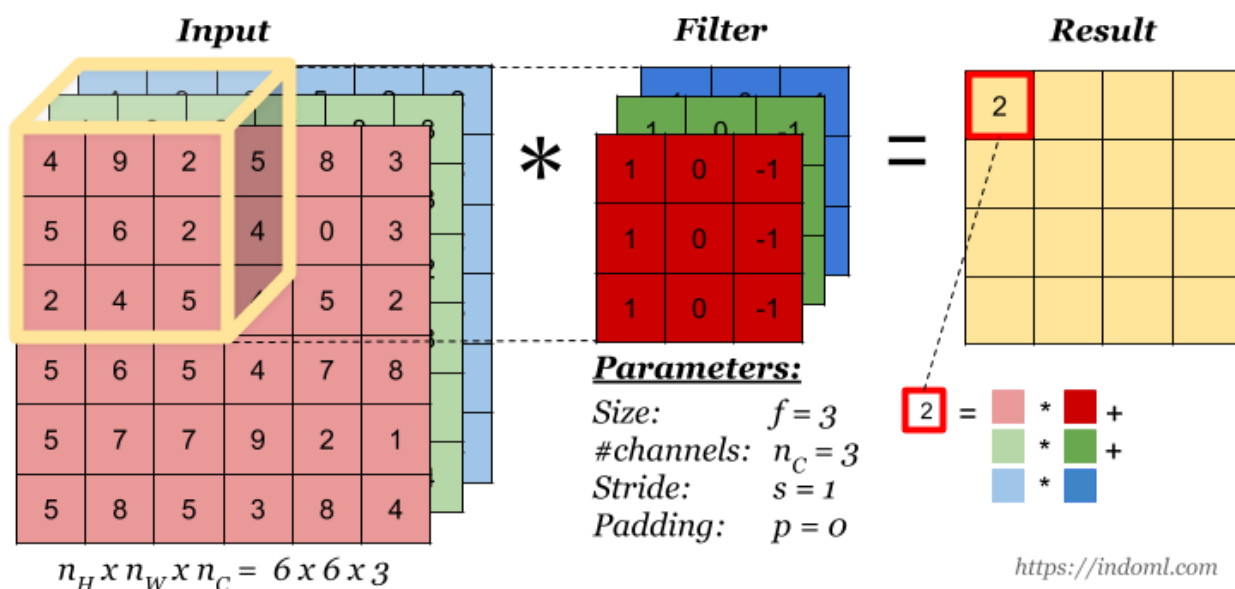


Рис.5. Пример свертки с трехмерным ядром

- Пулинг

Пулинг-слой призван снижать размерность изображения. Исходное изображение делится на блоки размером $w \times h$ и для каждого блока вычисляется некоторая функция. Чаще всего используется функция максимума или (взвешенного) среднего.

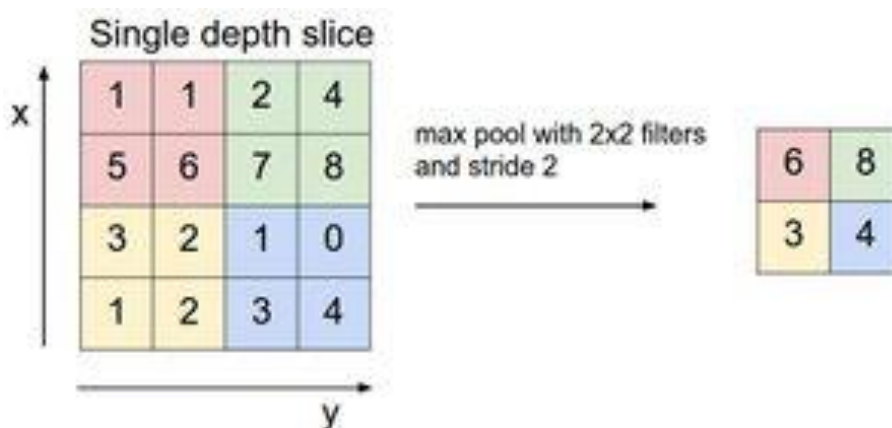


Рис.6. Пример операции пулинга с функцией максимума

- Полносвязный слой (англ. fully-connected layer)

Полносвязный слой — это слой, в котором каждый элемент на входе связан с каждым элементом на выходе. Он состоит из линейных слоев и функций активации. Линейный слой - линейное преобразование над

входящими данными. Функция активации - нелинейное преобразование, поэлементно применяющееся к пришедшим на вход данным. Благодаря функциям активации нейронные сети способны порождать более информативные признаковые описания, преобразуя данные нелинейным образом.



Рис.7. Пример комбинации линейного слоя и функции активации

1.2.3 Параметры сверточной нейронной сети

Производительность CNN сильно зависит от множества параметров, чтобы понять, как их настроить, разберемся как она обучается. Вот параметры, от которых зависит работа нейросети:

- Количества слоев
- Размера ядер свёртки
- Числа фильтров на каждом слое
- Шага свёртки
- Типа пулинга
- Выбора функции активации
- Архитектуры финального полносвязного слоя

В самом начале мы должны подготовить данные, набор данных делим на тренировочную, валидационную и тестовую выборки. Далее создаем модель из сверточных слоев, слоев пулинга и полносвязных слоев. Многие параметры зависят от задачи и размера данных. После обучения на обучающей выборке, проверяем ее точность на валидационном и тестовом наборе путем сравнения предсказанных значений с истинными. После оценки уже можно приступить к тонкой настройке модели, добавить или удалить слои, изменить параметры.

Но как же выбрать оптимальное количество слоев и параметры? Начать стоит либо с простой модели, либо с предобученной (VGG, ResNet). Для предотвращения переобучения можно использовать методы регуляризации, такие как Dropout, L1 и L2. Так же можно использовать различные методы оптимизации, такие как Adam, SGD, RMSprop и т.д. Так же можно объединить несколько моделей в одну.

Обычно для больших и сложных данных требуется большее количество слоев, такая же динамика при большом количестве классов, простая модель не подойдет для многоклассовой классификации. Однако если данных слишком мало, то сложная модель переобучится, и будет показывать плохие результаты.

Какие типы слоев можно использовать для создания модели:

- Сверточные слои: извлекают признаки из входных данных, фильтры производят свертку, затем результаты свертки проходят через функцию активации.
- Слои пулинга: уменьшают размерность данных, выбирают наибольшее/наименьшее значение среди смежных пикселей. (уменьшают количество параметров и предотвращают переобучение)
- Слои нормализации: ускоряют обучение и предотвращают переобучение.
- Полносвязные слои: классифицируют данные.
- Слои активации: применяют функцию активации к предыдущему слою. (улучшают нелинейность и ускоряют обучение)
- Слои векторизации: преобразуют категориальные признаки в векторы фиксированной длины.

Благодаря функциям активации нейросеть может обучаться на сложных данных, они вносят нелинейность в нейросеть. Самые популярные виды:

- ReLU (Rectified Linear Unit): $f(x) = \max(0, x)$ - Если вход x положительный, функция возвращает, иначе — 0. Для отрицательных

значений производная равна нулю, что может приводить к проблеме «мертвых» нейронов (dead neurons).

- Leaky ReLU: $f(x) = \max(\alpha x, x)$, $\alpha \approx 0.01$ - В отличие от ReLU, позволяет небольшие отрицательные значения вместо того, чтобы просто обнулять их.
- Sigmoid: $f(x) = \frac{1}{1+e^{-x}}$ - Преобразует любое значение в диапазон от 0 до 1, делая его удобным для задач бинарной классификации. Проблема исчезающего градиента: при больших или маленьких значениях входа градиент становится слишком малым, что замедляет обучение.
- Tanh: $f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ - Преобразует входные значения в диапазон (-1, 1). Также страдает от исчезающего градиента.
- Softmax: $f(x) = \frac{e^{x_i}}{\sum e^{x_j}}$ - Преобразует вектор значений в вероятностное распределение (все значения находятся в диапазоне (0,1) и их сумма равна 1).

Оптимизаторы определяют, как веса нейросети обновляются во время обучения для минимизации функции ошибки (loss function). Они ускоряют сходимость, помогают избежать локальных минимумов и адаптируются к различным условиям обучения. Вот основные методы оптимизации:

- SGD (Stochastic Gradient Descent): $\omega \leftarrow \omega - \eta \nabla L(w)$ - стохастический градиентный спуск. Обновляет веса после каждого обучающего примера (или небольшого батча).
- Momentum: $v_t = \beta v_{t-1} - \eta \nabla L(\omega)$ - Добавляет «инерцию» градиентному спуску, сохраняя часть скорости предыдущих шагов. Если градиент движется в одном направлении, шаги становятся больше. В колеблющихся областях помогает быстрее выбраться.
- Adam (Adaptive Moment Estimation): $m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$; $v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$ - Адаптивно регулирует шаг обучения для каждого

параметра. Использует моментум для ускорения и RMSprop для стабилизации.

- RMSprop: $v_t = \beta v_{t-1} + (1 - \beta)g_t^2$ - Контролирует скорость обучения, уменьшая шаги для больших градиентов.

Регуляризация снижает вероятность переобучения модели, делая её более устойчивой к шуму и улучшая её способность обобщать знания на новые данные. Без регуляризации сеть может запомнить обучающий набор, но плохо работать на тестовых данных. Вот основные методы регуляризации:

- L1 (Lasso) и L2 (Ridge): $L_1 = \lambda \sum |\omega|$; $L_2 = \lambda \sum \omega^2$ - L1 создает разреженные веса (некоторые становятся нулевыми), L2 уменьшает влияние больших весов.
- Dropout: случайное отключение нейронов в обучении.
- Batch Normalization: нормализация входов слоя.
- Early Stopping: остановка обучения при росте ошибки на валидации.

1.2.3 Популярные архитектуры

Рассмотрим строение популярных эффективных архитектур: LeNet-5, AlexNet, VGG-19.

LeNet-5 – одна из самых первых и самых известных CNN архитектур, была создана в 1998 году, чаще всего используется для рукописного метода распознавания цифр. Имеет 2 сверточных и 3 полных слоя, имеет 60000 параметров. Каждый сверточный слой использует ядро 5×5 и сигмоидную функцию активации, поскольку тогда ReLU и max-pooling еще не были открыты.

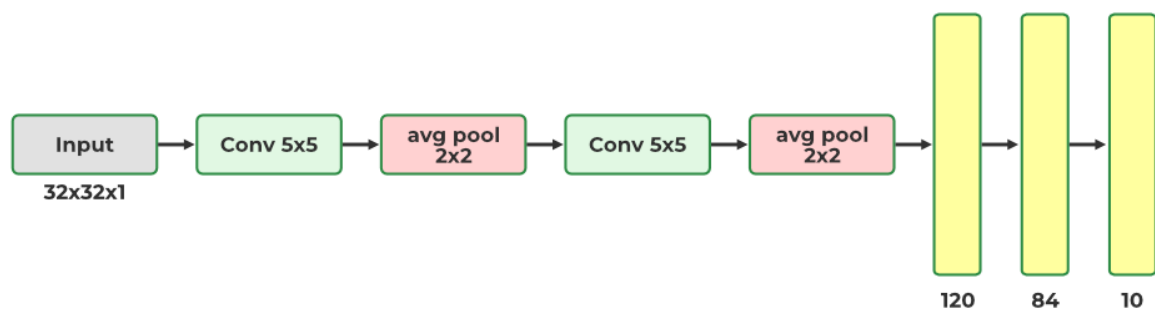


Рис.8. Архитектура LeNet-5

AlexNet – имеет всего 8 слоев: 5 сверточных и 3 полных слоя. Она стала первой, использующей ReLU в качестве функции активации. В первом слое AlexNet форма окна свёртки равна 11×11 , так как изображения в восемь раз выше и шире, чем изображения для LeNet-5.

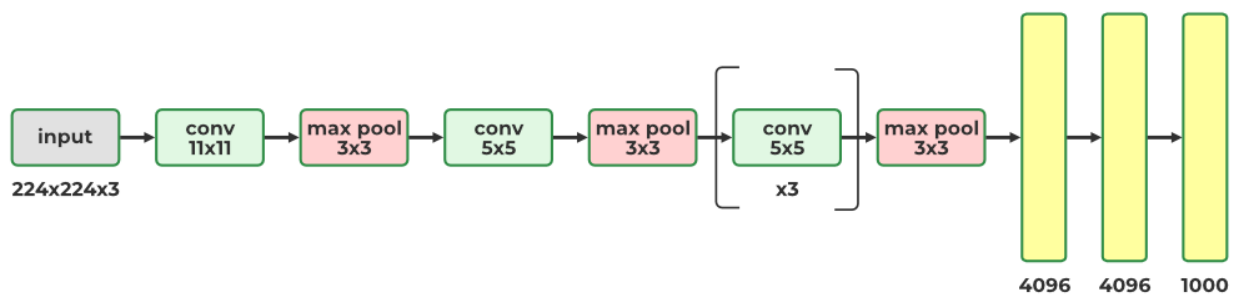


Рис.9. Архитектура AlexNet

VGG-19 - это глубокая сверточная нейронная сеть с 19 весовыми слоями, включающая 16 сверточных слоев и 3 полносвязных слоя.

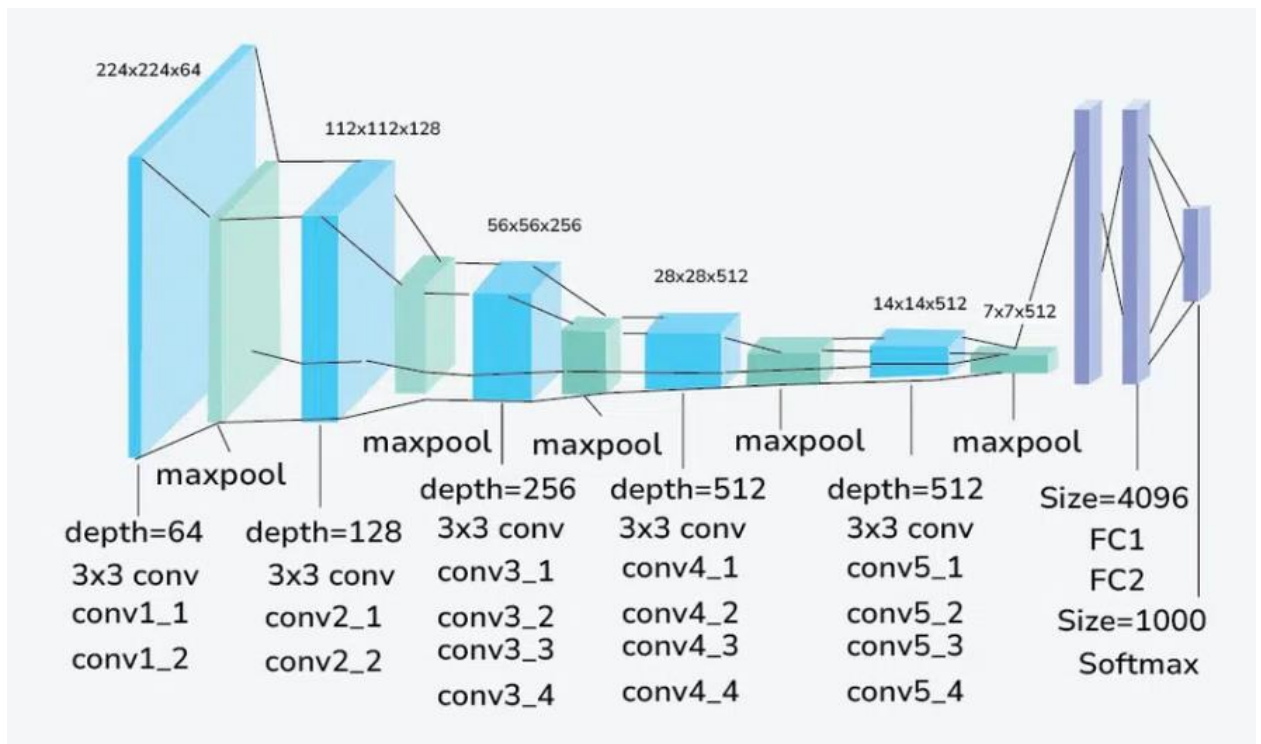


Рис.10. Архитектура VGG-19

1.2.4 Преимущества и недостатки

Сверточные нейронные сети обладают рядом преимуществ, которые делают их одним из наиболее эффективных инструментов для распознавания изображений. Однако, как и любая технология, они имеют свои ограничения.

Достоинства CNN:

- Обобщение информации и сохранение топологии изображения (Благодаря локальному восприятию данных, сеть не запоминает изображение на уровне отдельных пикселей, а выделяет значимые паттерны, что улучшает способность к обобщению.)
- Робастность к повороту и сдвигу. (Алгоритмы CNN способны успешно обрабатывать изображения с небольшими поворотами, сдвигами и вариациями освещения, что делает их более устойчивыми к изменениям входных данных.)

- Частичная инвариантность к масштабу. (Благодаря слоям пулинга сеть может сохранять ключевые признаки объекта даже при изменении его размеров.)

Недостатки CNN:

- Долгое обучение. (Глубокие сверточные сети с более чем двумя сверточными слоями требуют значительного времени на обучение, вплоть до нескольких дней)
- Требовательность к данным. (Для эффективного обучения CNN необходимы большие объемы размеченных данных. Например, база данных MNIST, используемая для распознавания рукописных цифр, содержит более 60 000 примеров. Недостаток данных может привести к переобучению.)
- Сложность настройки гиперпараметров. (Оптимальные параметры приходится подбирать эмпирически для каждой новой задачи, что делает процесс настройки сложным и трудоёмким.)

1.3 Гибридные квантовые нейросети

1.3.1 Основные определения

Кубит — это квантовый бит. Отличается от бита он тем, что может находиться не только в состоянии 0 или 1, но и в суперпозиции состояний 0 и 1 одновременно. Это свойство позволяет квантовым компьютерам обрабатывать большое количество информации параллельно.

$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, где $|0\rangle$ и $|1\rangle$ — базисные состояния, а α и β — комплексные числа, называемые амплитудами вероятности, удовлетворяющие условию нормировки: $|\alpha|^2 + |\beta|^2 = 1$

Квантовая запутанность - состояние, при котором два или более кубита становятся коррелированными таким образом, что состояние одного кубита

мгновенно влияет на состояние другого, независимо от расстояния между ними.

Измерение – это процесс получения информации о состоянии кубита, который приводит к коллапсу суперпозиции в одно из классических состояний $|0\rangle$ или $|1\rangle$ с вероятностями, зависящими от коэффициентов α и β . После измерения кубит теряет свою суперпозицию, что накладывает ограничения на копирование квантовой информации.

Квантовые компьютеры манипулируют кубитами с помощью квантовых ворот, аналогичных логическим вентилям в классической схеме. Они выполняют унитарные преобразования, обеспечивающие обратимость вычислений.

1.3.2 Квантовый слой

Квантовый слой представляет собой компонент нейронной сети, использующий принципы квантовых вычислений для более эффективной обработки информации. В отличие от классических слоёв, основанных на матричных операциях с фиксированными весами, квантовый слой использует квантовые гейты, которые позволяют выполнять многомерные преобразования данных. Это даёт потенциальные преимущества в скорости вычислений, представлении сложных зависимостей и сокращении количества параметров модели.

Основой квантового слоя являются кубиты — квантовые аналоги классических битов, способные находиться в суперпозиции, то есть одновременно представлять несколько состояний. Дополнительно, использование квантового запутывания создаёт связи между кубитами, что помогает моделировать сложные зависимости в данных. Основные вычисления в таком слое выполняются при помощи унитарных квантовых операций, которые трансформируют входные данные, создавая новые

представления, более выразительные, чем те, что можно получить в классических нейросетях.

1.3.3 Интеграция

Рассмотрим архитектуру гибридной сети, состоящей из сверточных слоёв и квантовой схемы.

Квантовый слой можно встроить на нескольких этапах обработки изображения. Один из наиболее эффективных вариантов — размещение его после свёрточных слоёв, чтобы использовать квантовые вычисления для работы с уже выделенными признаками. Также возможна замена традиционного пулинга квантовой обработкой, что позволит более точно выделять значимые особенности изображения. Наконец, квантовый слой можно вставить перед полносвязными слоями, где он будет играть роль дополнительного преобразования признаков перед финальной классификацией.

Интеграция квантового слоя позволяет уменьшить количество параметров модели и потенциально ускорить обучение, однако современные квантовые процессоры не позволяют работать с большими объемами данных, поэтому многие эксперименты пока выполняются на симуляторах. Кроме того, квантовые алгоритмы требуют точной настройки, так как ошибки в вычислениях могут привести к ухудшению результатов. Однако с развитием технологий эти ограничения будут постепенно преодолеваться, а квантовые слои смогут занять важное место в будущих нейросетях, ускоряя и улучшая классификацию изображений.

1.4 Квантовые схемы

Рассмотрим следующую квантовую схему в качестве примера:

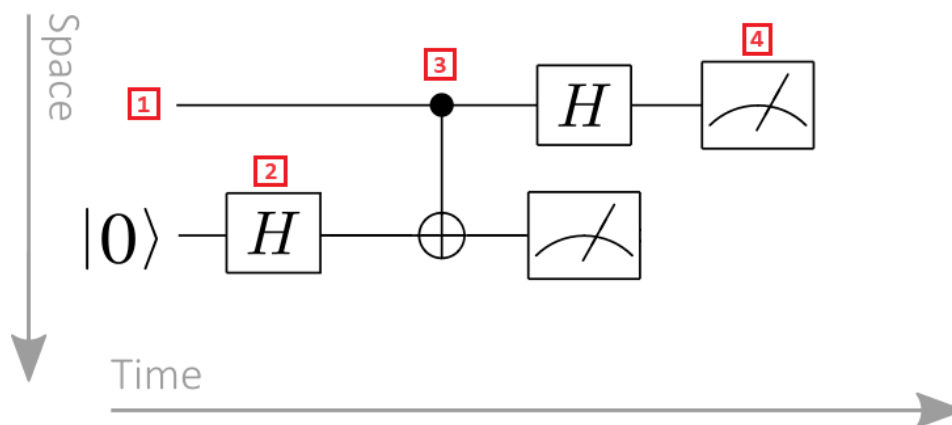


Рис.11. Пример квантовой схемы

Описание элементов на ней:

1. Регистр кубитов: отображаются как горизонтальные линии, каждая из которых представляет кубит. Верхняя линия — это регистр кубитов 0, следующая — регистр кубитов 1 и так далее.
2. Квантовые ворота: представляют квантовые операции. Ворота, действующие на один или несколько регистров, обозначаются прямоугольниками. Например, символ представляет операцию Адамара.
3. Контролируемые ворота: действуют на два или более кубита. Например, CNOT. Черный круг — это управляющий кубит, а крест в круге — это целевой кубит.
4. Операция измерения: символ "измеритель" обозначает операцию измерения, которая принимает регистр кубитов как вход и выводит классическую информацию.

1.4.1 Основные одно-кубитные операции

Фазовые ворота (S, T) – изменяют фазу кубита.

$$T = \begin{bmatrix} 1 & 0 & 0 & e^{i\frac{\pi}{4}} \end{bmatrix}$$

$$S = \begin{bmatrix} 1 & 0 & 0 & i \end{bmatrix} = T^2$$

Операторы Паули (X, Y, Z) – изменяют состояние кубита, аналогично классическим логическим операциям.

$$X = [0 \ 1 \ 1 \ 0] = HT^4H$$

$$Y = [0 \ -i \ i \ 0] = T^2HT^4HT^6$$

$$Z = [1 \ 0 \ 0 \ -1] = T^4$$

Ворота Адамара (H) – создают суперпозицию.

$$H = \frac{1}{\sqrt{2}} [1 \ 1 \ 1 \ -1]$$

Операции вращения (R_x, R_y, R_z) – позволяют осуществлять произвольные унитарные преобразования на сфере Блоха.

$$R_z(\theta) = e^{-i\theta Z/2} = \begin{bmatrix} e^{-i\theta/2} & 0 \\ 0 & e^{i\theta/2} \end{bmatrix}$$

$$R_x(\theta) = e^{-i\theta X/2} = HR_z(\theta)H = \begin{bmatrix} \cos(\theta/2) & -i\sin(\theta/2) \\ -i\sin(\theta/2) & \cos(\theta/2) \end{bmatrix}$$

$$R_y(\theta) = e^{-i\theta Y/2} = SHR_z(\theta)HS^\dagger = \begin{bmatrix} \cos(\theta/2) & -\sin(\theta/2) \\ \sin(\theta/2) & \cos(\theta/2) \end{bmatrix}$$

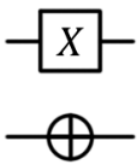
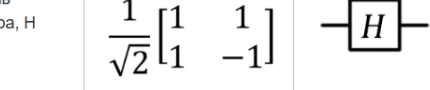
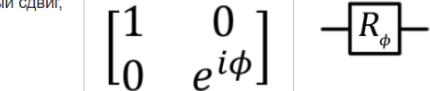
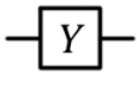
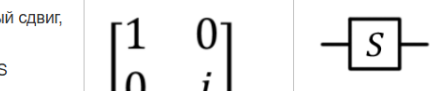
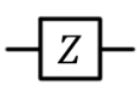
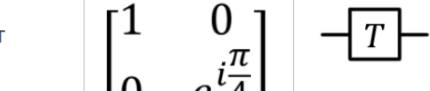
σ_x	$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$		Вентиль Адамара, H	$\frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$	
			Фазовый сдвиг, R_ϕ	$\begin{bmatrix} 1 & 0 \\ 0 & e^{i\phi} \end{bmatrix}$	
σ_y	$\begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$		Фазовый сдвиг, $\frac{\pi}{4}, S$	$\begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}$	
σ_z	$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$		$\frac{\pi}{8}, T$	$\begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{bmatrix}$	

Рис.12. Обозначение одно-кубитных операций на квантовой схеме

1.4.2 Основные многокубитные операции

Представителями многокубитных операций являются вентиль SWAP, меняющий местами два входных кубита, CNOT и другие.

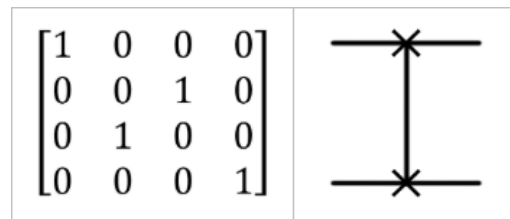


Рис.13. Обозначение многокубитного вентиля SWAP на квантовой схеме

На вход любого управляемого вентиля подается по меньшей мере один управляющий и один управляемый кубит, причем вентиль выполнит операцию над управляемым кубитом только в том случае, если управляющий кубит находится в определенном состоянии. Вентили, которые выполняют операцию при управляющем кубите $|1\rangle$, обозначаются заполненным кругом на проводе управляющего кубита. Вентили, которые выполняют операцию при управляющем кубите, равном $|0\rangle$, обозначаются пустой окружностью, как показано ниже.

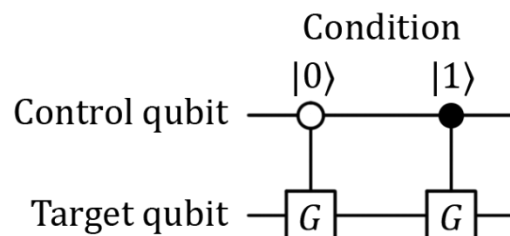


Рис.14. Обозначение управляемых вентилях на квантовой схеме

Вентиль CNOT является одним из самых важных вентилях в квантовых вычислениях. Он используется для создания запутанных кубитов. Вот так работает CNOT: он берет два входа, $|\psi\rangle_1$ и $|\psi\rangle_2$. Если $|\psi\rangle_1$ равно $|1\rangle$, то состояние $q1$ остается $|\psi\rangle_1$, но $|\psi\rangle_2$ становится $X|\psi\rangle_1$. В противном случае состояния $q1$ и $q2$ не изменяются.

CNOT	$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$	
------	--------------------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------

Рис.15. Обозначение CNOT на квантовой схеме

1.4.3 Примеры реализации

Рассмотрим вариант реализации квантового слоя на основе квантовой схемы, состоящей из 2–4 кубитов.

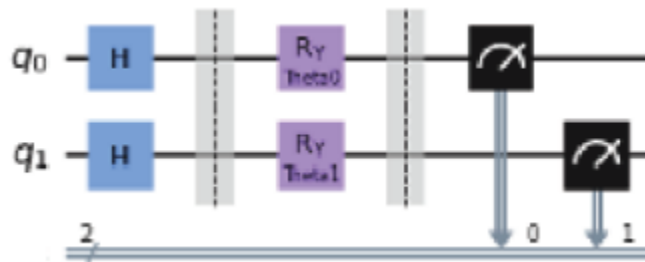


Рис.16. Квантовая схема с 2-я кубитами, включая один обучаемый параметр на кубит (R_y)

Квантовая схема R_y имеет 1 обучаемый параметр на кубит. С целью взаимодействия алгоритма градиентного спуска и квантовой схемы необходимо использовать формулу:

$$gradient = qc \cdot (\theta + \epsilon) - qc \cdot (\theta - \epsilon)$$

Посредством данной формулы выполняется настройка обучаемых параметров нейросети, где θ – параметр квантовой схемы; ϵ – сдвиг. Под градиентом понимается разница между показателями квантовой схемы, которые оценивается при её сдвиге вправо ($\theta + \epsilon$) и влево ($\theta - \epsilon$). Это позволяет дифференцировать квантовую схему, как часть алгоритма обратного распространения ошибки.

Рассмотрим еще один пример реализации - 2-кубитный Q-узел. Этот квантовый слой выполняет 3 ключевые операции: кодирование входных данных (Embedding), квантовую обработку с запутыванием, измерение и преобразование обратно в классические данные.

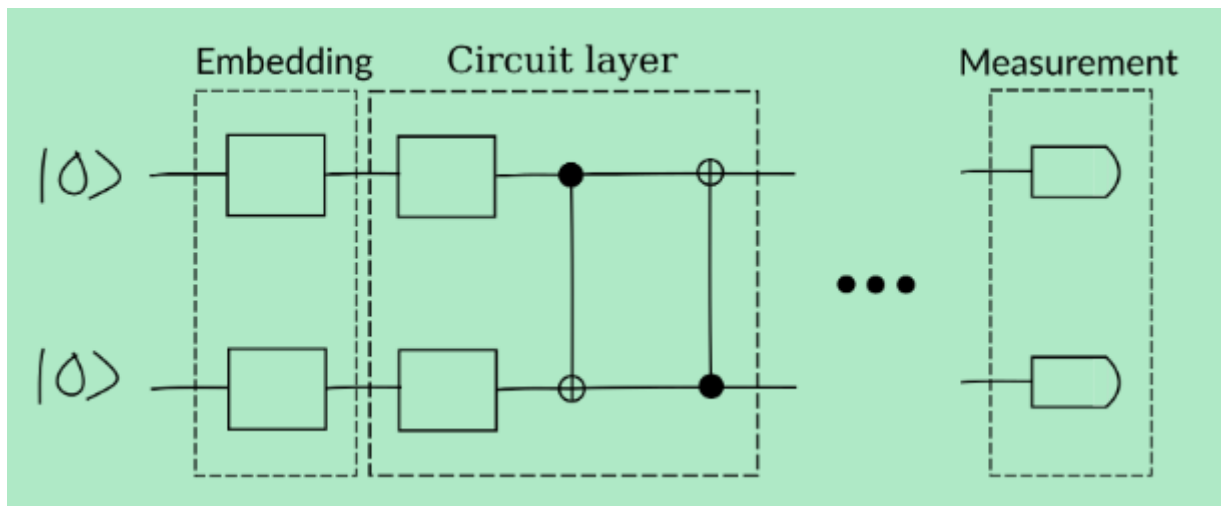


Рис.17. 2-кубитный Q-узел.

1.5 Острый лимфобластный лейкоз

Острый лимфобластный лейкоз – это рак крови, представляет из себя неконтролируемое разрастание незрелых лимфоидных клеток (лимфобластов) путем деления.

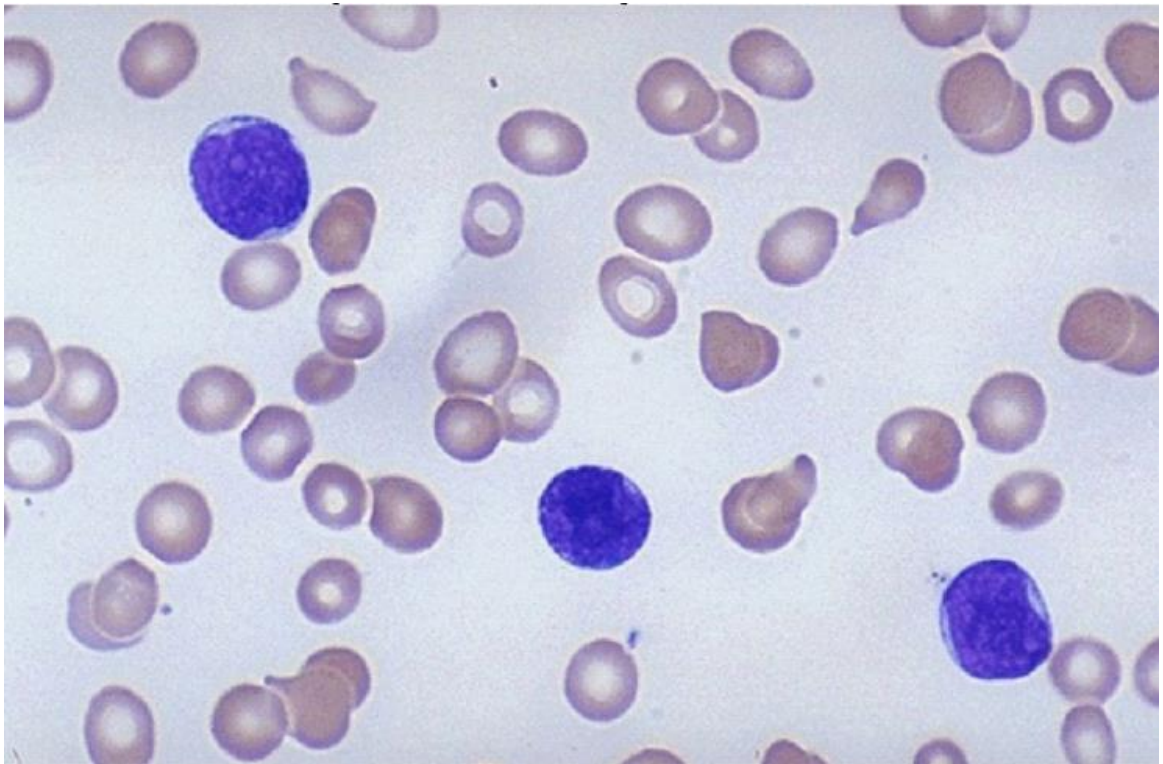


Рис.18. Лимфобласты в мазке крови при остром лимфолейкозе

Лимфобласты в крови можно спутать с лимфоцитами, миелобластами, плазматическими клетками. Задача выявления незрелых лейкозных взрывов

среди нормальных клеток под микроскопом очень сложна из-за морфологического сходства.

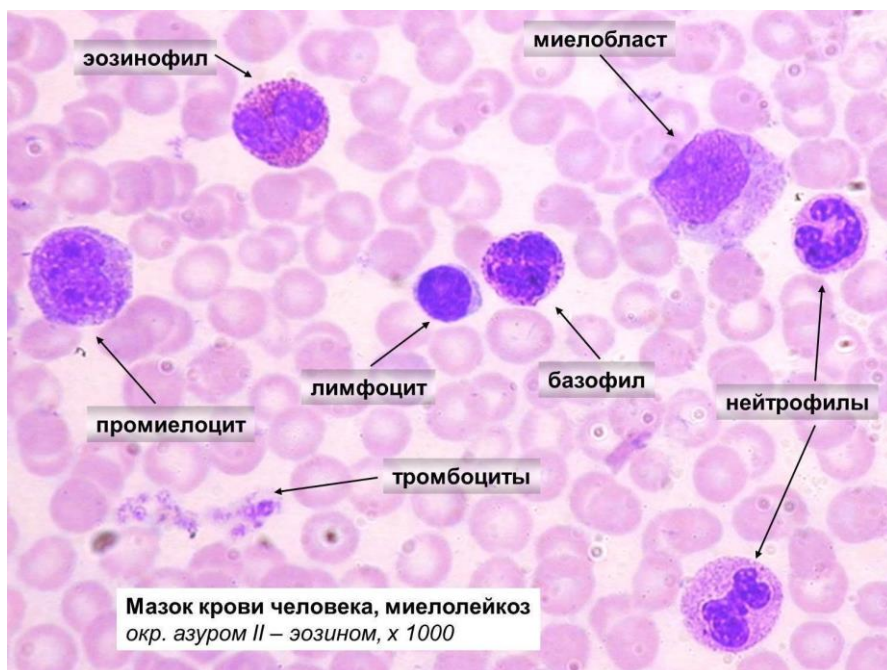


Рис.19. Нормальные клетки в крови

Лимфобласт от остальных клеток отличается размерами (13-16 мкм). Он имеет ядро округлой или овальной формы (обычно оно меньше, чем у миелобластов), светло-фиолетового цвета, расположено в центре. Хроматин тонкодисперсный (состоит из тонко измельченных частиц), конденсированный. Цитоплазма узкая светло-синего цвета с небольшой перинуклеарной зоной просветления или отсутствует.

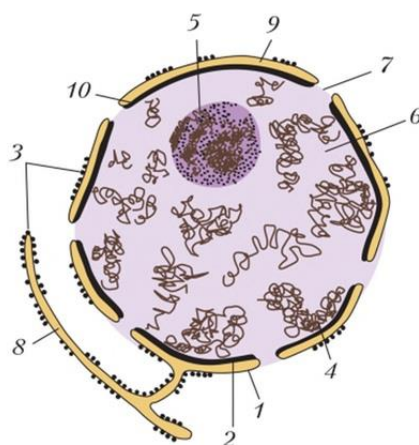


Рис.20. Строение клеточного ядра: 4 – хроматин; 5 – ядрышко; 9 – перинуклеарное пространство

2 ОСНОВНАЯ ЧАСТЬ

2.1 Содержательная постановка задачи

Целью данного исследования является разработка и экспериментальная проверка гибридной квантово-классической нейронной сети для задачи классификации изображений. В качестве тестового датасета будет использован набор данных с лейкозными и нормальными клетками, где основная задача модели — определить, является клетка раковой или нет. Для этого будет создана архитектура, сочетающая классические слои сверточной нейронной сети (CNN) для извлечения признаков из изображений, квантовый слой в виде квантовой схемы, и полносвязный классический слой для финальной классификации.

В процессе исследования будет проведена подготовка и предобработка данных, включая нормализацию и преобразование изображений для подачи на вход модели. Датасет с изображениями будет разделен на обучающую, валидационную и тестовую выборки. Качество модели будет оцениваться на тестовой выборке с помощью метрики Accuracy.

Особое внимание будет уделено сравнению производительности гибридной квантово-классической модели с классической нейронной сетью, не использующей квантовый слой. Это позволит выявить влияние квантового слоя на точность классификации и способность модели к обобщению. Сложность и устойчивость модели к различным типам данных.

В рамках исследования будут использованы методы линейной алгебры, квантовых вычислений и теории вероятностей, а также инструменты машинного обучения, такие как TensorFlow и PennyLane. Итогом работы станет подтверждение или опровержение гипотезы о том, что интеграция квантовых вычислений в классические нейронные сети может повысить точность и эффективность моделей в задачах классификации изображений.

2.2 Математическая постановка задачи

Главная цель задачи — построение гибридной квантово-классической нейронной сети для классификации изображений, которая минимизирует функцию потерь \mathcal{L} на заданном датасете изображений $D = \{(x_i, y_i)\}_{i=1}^N$, где:

$x_i \in R^{H \times W \times C}$ (H — высота изображения, W — ширина, C — количество цветовых каналов (в нашем случае RGB изображение, $C = 3$)),

$y_i \in \{0, 1\}$ — истинная метка класса (0 — нормальная клетка, 1 — лимфобласт (раковая клетка)).

Формально цель задачи можно записать как минимизацию бинарной кросс-энтропии (измеряет "расстояние" между реальными метками и предсказаниями модели):

$$\mathcal{L}(\hat{y}_i, y_i) = \min \frac{1}{N} \sum_{i=1}^N -[y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)], \text{ где:}$$

$\hat{y}_i \in [0, 1]$ — предсказанная метка класса,

N — количество изображений.

2.3 Средства разработки

В качестве редактора для написания и отладки кода станет Visual Studio Code, из-за его хорошей интеграции с системами контроля версий и удобным инструментарием для работы с Python.

Работа над проектом ведется на языке программирования Python, который предоставляет мощные библиотеки для обработки данных, машинного обучения и квантовых вычислений. Для реализации квантового слоя был использован фреймворк PennyLane, который позволяет интегрировать квантовые вычисления в классические модели машинного обучения.

Для математических и аналитических операций используется библиотека NumPy, которая предоставляет эффективные инструменты для работы с многомерными массивами и матрицами. Для визуализации результатов и анализа данных была использована библиотека Matplotlib. Для предобработки изображений и работы с датасетами - библиотека OpenCV. Код проекта размещён в репозитории на GitHub.

2.4 Сведения о проведенных исследованиях и полученных результатах

Для исследования был использован датасет с Kaggle: C_NMC_2019 Dataset: ALL Challenge dataset of ISBI 2019.

Острый лимфобластический лейкоз (данные в папке “All”) является наиболее распространенным типом детского рака и составляет приблизительно 25% педиатрического рака. Клетки в датасете были сегментированы с реальных микроскопических изображений. Задача выявления незрелых лейкозных взрывов среди нормальных клеток под микроскопом сложна из-за морфологического сходства, поэтому этикетки истины были аннотированы экспертом онкологом.

Всего есть 15,135 изображений от 118 пациентов с двумя помеченными классами: нормальная ячейка, лейкомия.

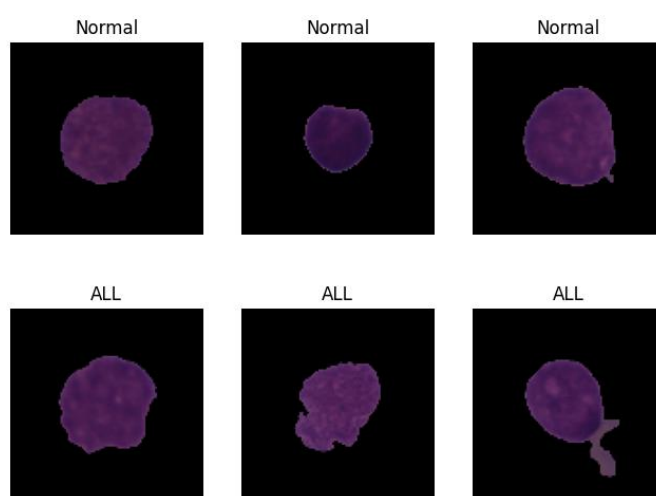


Рис.21. Пример изображений из датасета

```
validation_list = get_path_image(path_val)
#0 - нормальные клетки, 1 - раковые

validation_dict = {"x_col":validation_list , "y_col":val_labels["labels"]}
validation_df = pd.DataFrame(validation_dict)
validation_df["y_col"].replace(to_replace = [1,0], value = ["ALL","HEM"], inplace = True)
```

✓ 0.0s

Рис.22. Пример присваивания изображениям метки

```
val_datagen = ImageDataGenerator(rescale=1./255)
validation_generator = val_datagen.flow_from_dataframe(
    validation_df,
    x_col = "x_col",
    y_col = "y_col",
    target_size = (256, 256),
    batch_size = 32,
    color_mode = "rgb",
    shuffle = True,
    class_mode = "binary")
```

✓ 0.0s

Found 1867 validated image filenames belonging to 2 classes.

Рис.23. Пример загрузки данных

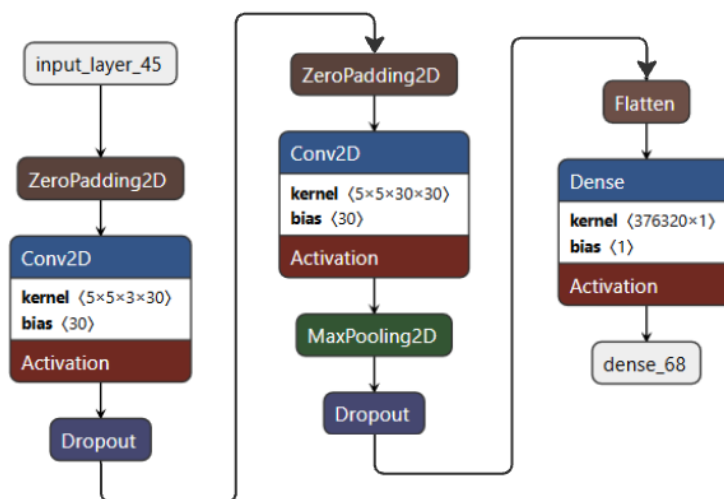
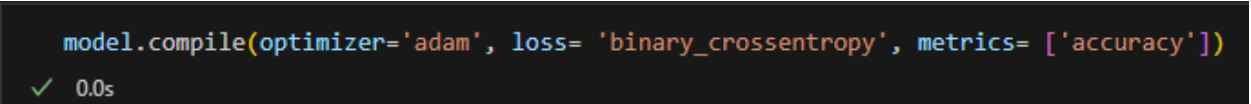


Рис.24. Схема реализованной нейросети

Для классификации клеток была реализована сверточная нейросеть. Она состоит из нескольких слоев:

Вот описание каждого слоя модели:

- Входной слой (ZeroPadding2D): Этот слой добавляет нулевые отступы (padding) вокруг входного изображения увеличивая размер.
- Сверточные слои (Conv2D): Выполняют двумерную свертку с 30 фильтрами размером 5×5 .
- Dropout: Случайным образом отключает какой-то процент нейронов во время обучения, чтобы предотвратить переобучение модели и повысить ее обобщающую способность.
- Слой пулинга (MaxPooling2D): Уменьшает размер изображения в два раза.
- Flatten: Преобразует многомерный выход из предыдущих слоев в одномерный вектор, чтобы его можно было подать на полносвязные слои.
- Полносвязный слой (Dense): Создает линейную зависимость между признаками (линейная активация).
- Выходной слой (Dense): Полносвязный слой с одним нейроном и функцией активации сигмоида.



```
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
✓ 0.0s
```

Рис.25. Компиляция модели

В качестве функции потерь использовалась бинарная кросс-энтропия, для оценки качества модели используется метрика “точность”. Используется оптимизатор – Adam (Adaptive Moment Estimation). Размер батча – 32.

Поскольку количество эпох подбирается эмпирически, было решено обучать модель в течение 20 эпох, и подобрать такое их количество, чтобы модель не переобучилась. Ниже представлена визуализация процесса изменения точности на разных эпохах.

Так как количество двух видов фотографий сильно отличалось - {0: 7272, 1: 3389}, было решено добавить классам веса для лучшего обучения $\text{class_weight}=\{0: 0.73, 1: 1.57\}$.

```
Epoch 16/20
334/334 ————— 507s 2s/step - accuracy: 0.9212 - loss: 0.1973 - val_accuracy: 0.6213 - val_loss: 1.3478
Epoch 17/20
334/334 ————— 504s 2s/step - accuracy: 0.9294 - loss: 0.1797 - val_accuracy: 0.6213 - val_loss: 1.3082
Epoch 18/20
334/334 ————— 506s 2s/step - accuracy: 0.9253 - loss: 0.1874 - val_accuracy: 0.6069 - val_loss: 1.3532
Epoch 19/20
334/334 ————— 504s 2s/step - accuracy: 0.9390 - loss: 0.1550 - val_accuracy: 0.6251 - val_loss: 1.7565
Epoch 20/20
334/334 ————— 505s 2s/step - accuracy: 0.9209 - loss: 0.1862 - val_accuracy: 0.6144 - val_loss: 1.5577
Total Time Elapsed: 168 minutes 10 seconds
```

Рис.26. Статистики при обучении

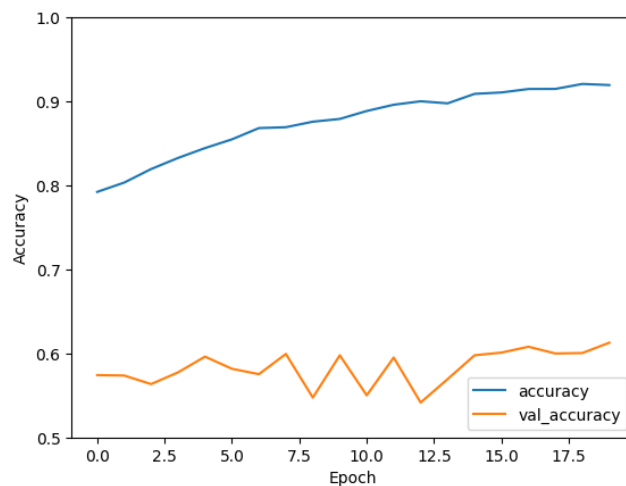


Рис.27. График изменения точности на разных эпохах

В результате работы нейросети были получены следующие статистики:

- Точность на обучающей выборке = 92%
- Точность на тестовой выборке = 62%

Модель показала себя неплохо. Далее был проведен эксперимент по внедрению квантового слоя в качестве отдельного слоя перед полносвязным. Для внедрения квантового слоя был реализован 2-кубитный Q узел из аналитического обзора.

```

import pennylane as qml

n_qubits = 2
dev = qml.device("default.qubit", wires=n_qubits)

@qml.qnode(dev)
def qnode(inputs, weights):
    qml.AngleEmbedding(inputs, wires=range(n_qubits))
    qml.BasicEntanglerLayers(weights, wires=range(n_qubits))
    return [qml.expval(qml.PauliZ(wires=i)) for i in range(n_qubits)]

n_layers = 6
weight_shapes = {"weights": (n_layers, n_qubits)}

qlayer = qml.qnn.KerasLayer(qnode, weight_shapes, output_dim=n_qubits)
#qlayer добавляем в модель

```

Рис.28. Код квантового слоя

```

Epoch 1/10
334/334 [=====] - 1750s 5s/step - loss: 0.7100 - accuracy: 0.4361 - val_loss: 0.6827 - val_accuracy: 0.6529
Epoch 2/10
334/334 [=====] - 1660s 5s/step - loss: 0.6846 - accuracy: 0.6810 - val_loss: 0.6690 - val_accuracy: 0.6513
Epoch 3/10
334/334 [=====] - 1843s 6s/step - loss: 0.6491 - accuracy: 0.6346 - val_loss: 0.7068 - val_accuracy: 0.5040
Epoch 4/10
334/334 [=====] - 1811s 5s/step - loss: 0.6908 - accuracy: 0.5521 - val_loss: 0.6915 - val_accuracy: 0.5308
Epoch 5/10
334/334 [=====] - 1660s 5s/step - loss: 0.6297 - accuracy: 0.6582 - val_loss: 0.8447 - val_accuracy: 0.3487
Epoch 6/10
334/334 [=====] - 1571s 5s/step - loss: 0.5425 - accuracy: 0.7565 - val_loss: 0.7900 - val_accuracy: 0.5062
Epoch 7/10
334/334 [=====] - 1572s 5s/step - loss: 0.5205 - accuracy: 0.7718 - val_loss: 1.0628 - val_accuracy: 0.3482
Epoch 8/10
334/334 [=====] - 1571s 5s/step - loss: 0.5024 - accuracy: 0.7823 - val_loss: 0.8356 - val_accuracy: 0.4697
Epoch 9/10
334/334 [=====] - 1572s 5s/step - loss: 0.5015 - accuracy: 0.7841 - val_loss: 0.7986 - val_accuracy: 0.5260
Epoch 10/10
334/334 [=====] - 1572s 5s/step - loss: 0.4865 - accuracy: 0.7937 - val_loss: 0.7552 - val_accuracy: 0.5988
Total Time Elapsed: 276 minutes 22 seconds

```

Рис.29. Статистика по эпохам модели с квантовым слоем

В результате работы нейросети с квантовым слоем были получены следующие статистики:

- Точность на обучающей выборке = 80%
- Точность на тестовой выборке = 60%

Пока что не удалось добиться положительного результата с квантовым слоем, в будущем планируется провести еще больше экспериментов с другими архитектурами квантового слоя, и другими его расположениями.

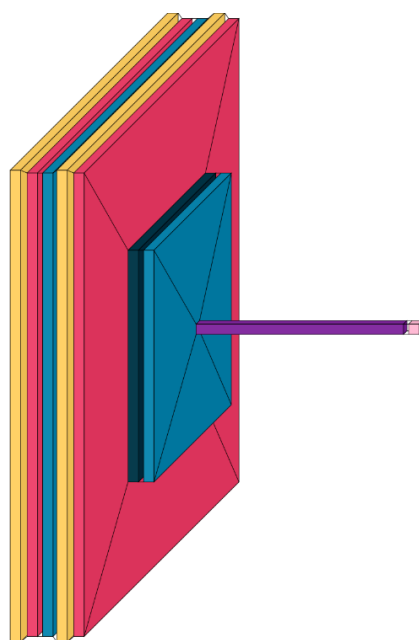


Рис.30. Реализованная сверточная нейронная сеть



Рис.31. Пример работы модели

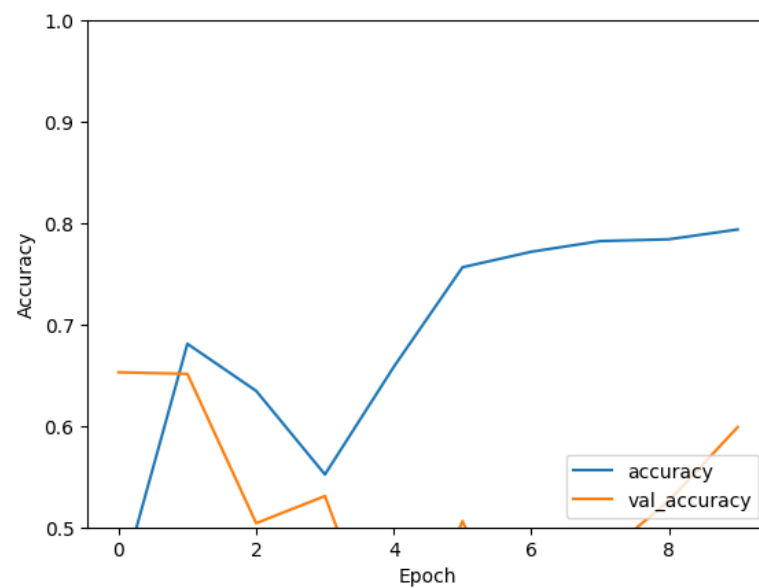


Рис.32. График изменения точности сети с квантовым слоем на разных эпохах

ЗАКЛЮЧЕНИЕ

В рамках выполнения курсовой научно-исследовательской работы были проведены исследования, направленные на анализ возможностей применения квантово-классических нейронных сетей для задачи классификации изображений. Основное внимание было уделено исследованию математического аппарата и принципов интеграции квантовых вычислений в классические нейронные сети, что позволило сформулировать гипотезу о потенциальных преимуществах такого подхода.

Основные цели КНИР, связанные с исследованием возможностей квантово-классических нейронных сетей, были достигнуты. Были изучены теоретические основы и предложена возможная архитектура модели, а также ее реализация и экспериментальная проверка. Сверточная нейросеть без квантового слоя показала неплохие результаты: 92%, в то время как модель с квантовым слоем всего 80%. Эффективная архитектура квантового слоя будет найдена экспериментальным путём в ходе последующих этапов работы, включая тестирование различных конфигураций квантовых схем и их интеграции с классическими слоями.

В процессе выполнения КНИР были приобретены следующие компетенции:

- Навыки анализа теоретических основ квантовых вычислений и их применения в машинном обучении.
- Понимание принципов работы гибридных квантово-классических моделей.
- Умение работать с научной литературой и проводить исследования в области квантового машинного обучения.
- Навыки формализации задач и предложения возможных решений на основе теоретического анализа.

Для дальнейшей подготовки выпускной квалификационной работы планируется найти оптимальную конфигурацию квантового слоя.

Были использованы следующие методы и средства информационных технологий:

- Программное обеспечение для квантовых вычислений (PennyLane).
- Библиотека для машинного обучения (TensorFlow).
- Инструменты для визуализации данных (Matplotlib).

Общее заключение об успешности выполнения КНИР на данном этапе является положительным. Были проведены важные исследования, которые легли в основу для дальнейшей работы.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Нильсен М., Чанг И. Н66 Квантовые вычисления и квантовая информация. Пер. с англ - М : Мир, 2006 г. — 824 с
2. Hafeez, M.A.; Munir, A.; Ullah, H. H-QNN: A Hybrid Quantum–Classical Neural Network for Improved Binary Image Classification. AI 2024, 5, 1462–1481. <https://doi.org/10.3390/ai5030070>
3. Гушанский, С.М., Буглов, В.Е. Разработка гибридной нейросети для классификации изображений // Южный федеральный университет. — Таганрог, 2023.
4. Вики-учебник по машинному обучению // НИУ ВШЭ. — URL: <https://neerc.ifmo.ru/wiki/>.
5. Stanford Course: Image Classification // REG.RU Blog. — URL: <https://www.reg.ru/blog/stehnfordskij-kurs-lekciya-2-klassifikaciya-izobrazhenij/>.
6. Image Classification: What It Is and How It Works // GeeksforGeeks. — URL: <https://www.geeksforgeeks.org/what-is-image-classification/>.
7. Сверточные нейронные сети // Яндекс.Практикум. — URL: <https://practicum.yandex.ru/blog/svertochnye-neyronnye-seti/>.
8. Сверточные нейронные сети: основы // Справочник по машинному обучению Яндекс. — URL: <http://education.yandex.ru/handbook/ml/article/svyortochnye-nejroseti>.
9. PennyLane AI. Quantum Neural Network with TensorFlow // PennyLane Demos. — URL: http://pennylane.ai-prod.s3-website-us-east1.amazonaws.com/qml/demos/tutorial_qnn_module_tf.html.
10. Gupta, A., & Gupta, R. (2019). ALL Challenge dataset of ISBI 2019 [Data set]. The Cancer Imaging Archive. <https://doi.org/10.7937/tcia.2019.dc64i46r>

ПРИМЕЧАНИЕ

Полный код программы:

```
import os
import random

import pandas as pd
import numpy as np

import cv2
from PIL import Image

import matplotlib.pyplot as plt
from matplotlib.pyplot import imshow

from timeit import default_timer as timer

import tensorflow as tf
from tensorflow.keras.models import Sequential
from keras.preprocessing import image
from tensorflow.keras import models, layers, optimizers
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import Input, Dense, Activation, Flatten, Conv2D, MaxPool2D, Dropout

import visualekera
```

Python

```
all_0 = "C-NMC_Leukemia/training_data/fold_0/all"
all_1 = "C-NMC_Leukemia/training_data/fold_1/all"
all_2 = "C-NMC_Leukemia/training_data/fold_2/all"

hem_0 = "C-NMC_Leukemia/training_data/fold_0/hem"
hem_1 = "C-NMC_Leukemia/training_data/fold_1/hem"
hem_2 = "C-NMC_Leukemia/training_data/fold_2/hem"
path_val = 'C-NMC_Leukemia/validation_data/C-NMC_test_prelim_phase_data'
val_labels = pd.read_csv('C-NMC_Leukemia/validation_data/C-NMC_test_prelim_phase_data_labels.csv')
```

Python

```
def get_path_image(folder):
    image_paths = []
    image_fnames = os.listdir(folder)
    for img_id in range(len(image_fnames)):
        img = os.path.join(folder, image_fnames[img_id])
        image_paths.append(img)

    return image_paths
```

Python

```

cancer_lst = []

for i in [all_0,all_1,all_2]:
    paths = get_path_image(i)
    cancer_lst.extend(paths)
print('No. of cancer images:', len(cancer_lst))

normal_lst = []
for i in [hem_0,hem_1,hem_2]:
    paths = get_path_image(i)
    normal_lst.extend(paths)
print('No. of normal images:', len(normal_lst))

```

Python

```

cancer_dict = {"x_col":cancer_lst, "y_col":[np.nan for x in range(len(cancer_lst))]}
cancer_dict["y_col"] = "ALL"

normal_dict = {"x_col":normal_lst, "y_col":[np.nan for x in range(len(normal_lst))]}
normal_dict["y_col"] = "HEM"

cancer_df = pd.DataFrame(cancer_dict)
normal_df = pd.DataFrame(normal_dict)

#train_df = cancer_df.append(normal_df, ignore_index=True)

plt.pie([len(cancer_lst),len(normal_lst)],labels=["ALL","Normal"],autopct='%f')
plt.title('Pie Chart for percentage of each cell type')
plt.show()

```

Python

```

select_normal = np.random.choice(normal_lst, 3, replace = False)
select_all = np.random.choice(cancer_lst, 3, replace = False)

```

```

train_df = pd.concat([cancer_df, normal_df], ignore_index=True)

```

Python

```

validation_list = get_path_image(path_val)
#0 - нормальные клетки, 1 - раковые

validation_dict = {"x_col":validation_list, "y_col":val_labels["labels"]}
validation_df = pd.DataFrame(validation_dict)
validation_df["y_col"].replace(to_replace = [1,0], value = ["ALL","HEM"], inplace = True)

```

Python

```

# Code Resizing For testing data
test_data = 'C-NMC_Leukemia/testing_data/C-NMC_test_final_phase_data'
test_list = get_path_image(test_data)

test_dict = {"x_col":test_list}

test_df = pd.DataFrame(test_dict)

```

Python

```

size = 224

train_datagen = ImageDataGenerator(rescale=1./255)
train_generator = train_datagen.flow_from_dataframe(
    train_df,
    x_col = "x_col",
    y_col = "y_col",
    target_size = (size, size),

    batch_size = 32,
    color_mode = "rgb",
    shuffle = True,
    class_mode = "binary"
)

val_datagen = ImageDataGenerator(rescale=1./255)
validation_generator = val_datagen.flow_from_dataframe(
    validation_df,
    x_col = "x_col",
    y_col = "y_col",
    target_size = (size, size),
    batch_size = 32,
    color_mode = "rgb",
    shuffle = True,
    class_mode = "binary")

test_datagen = ImageDataGenerator(rescale=1./255 )
test_generator = test_datagen.flow_from_dataframe(
    test_df,
    x_col = "x_col",
    target_size = (size, size),
    color_mode = "rgb",
    class_mode = None,
    shuffle = False
)

```

Python

```

train_df = pd.concat([cancer_df, normal_df], ignore_index=True)

```

Python

```

validation_list = get_path_image(path_val)
#0 - нормальные клетки, 1 - раковые

validation_dict = {"x_col":validation_list , "y_col":val_labels["labels"]}
validation_df = pd.DataFrame(validation_dict)
validation_df["y_col"].replace(to_replace = [1,0], value = ["ALL","HEM"], inplace = True)

```

Python

```

test_data = 'C-NMC_Leukemia/testing_data/C-NMC_test_final_phase_data'
test_list = get_path_image(test_data)

test_dict = {"x_col":test_list}

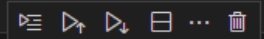
test_df = pd.DataFrame(test_dict)

```

Python

```
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

Python



```
from tensorflow.keras.callbacks import EarlyStopping
early_stop = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)

start = timer()
history = model.fit(train_generator, epochs=20, validation_data=validation_generator,
                    class_weight={0: 0.73, 1: 1.57})

end = timer()
elapsed = end - start
print('Total Time Elapsed: ', int(elapsed//60), ' minutes ', (round(elapsed%60)), ' seconds')
```

Python

```
def testing(image_path):
    test_image = image.load_img(image_path, target_size=(size, size))
    plt.imshow(test_image)
    test_image = image.img_to_array(test_image)
    test_image = np.expand_dims(test_image, axis=0)
    result = model.predict(test_image)
    print(result)
```

Python

```
image_path = 'C-NMC_Leukemia/training_data/fold_0/all/UID_1_6_1_all.bmp'
testing(image_path)
```

Python