Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО
ITMO University

ОТЧЁТ ПО УЧЕБНОЙ, ОЗНАКОМИТЕЛЬНОЙ ПРАКТИКЕ

Обучающийся / Student Никандров Сергей Андреевич
Факультет/институт/кластер/ Faculty/Institute/Cluster факультет систем управления и робототехники
Группа/Group R34352
Направление подготовки/ Subject area 15.03.06 Мехатроника и робототехника
Образовательная программа / Educational program Робототехника и искусственный интеллект 2021
Язык реализации ОП / Language of the educational program Русский
Квалификация/ Degree level Бакалавр
Тема ВКР/ Thesis topic Исследование методов и алгоритмов искусственного интеллекта для обнаружения и классификации поверхностных дефектов листового металлопроката
Руководитель ВКР/ Thesis supervisor Евстафьев Олег Александрович, кандидат технических наук, Университет ИТМО, факультет систем управления и робототехники, преподаватель (квалификационная категория "преподаватель")

СПИСОК ОБОЗНАЧЕНИЙ И СОКРАЩЕНИЙ

СНС	свёрточная нейронная сеть
Датасет	набор данных
НК	неразрушающий контроль
КЗ	компьютерное зрение
ГО	глубокое обучение
МО	машинное обучение
ИИ	искусственный интеллект
МГО	методы глубокого обучения
АП	архитектурный поиск
TP	Правильно предсказанный положительный объект.
TN	Правильно предсказанный отрицательный объект.
FP	Ложно предсказанный положительный объект.
FN	Ложно предсказанный отрицательный объект.

ВВЕДЕНИЕ

В условиях современной экономики, где конкуренция на рынке металлопродукции достигла высокого уровня, обеспечение высокого качества листового металлопроката является критически важным фактором для успешного функционирования предприятий. Поверхностные дефекты, такие как царапины, трещины, вмятины, загрязнения и ржавчина, не только снижают потребительскую ценность продукции, но и могут приводить к значительным экономическим потерям, связанным с браком, затратами на исправление дефектов и потерей репутации. Традиционные методы контроля качества, включающие визуальный контроль, ультразвуковой, рентгеновский, магнитопорошковый и вихретоковый методы, зачастую оказываются трудоемкими, подверженными влиянию человеческого фактора, и не всегда обеспечивают высокую скорость и точность обнаружения дефектов, особенно при больших объемах производства.

В связи с этим, разработка автоматизированных систем обнаружения и классификации поверхностных дефектов металлопроката приобретает особую актуальность. Использование методов и алгоритмов искусственного интеллекта, в частности, глубокого обучения, открывает новые возможности для повышения эффективности контроля качества, обеспечивая высокую точность, скорость обработки данных и минимизацию влияния человеческого фактора. Применение этих технологий позволяет не только выявлять дефекты, но и классифицировать их, что предоставляет ценную информацию для анализа причин возникновения дефектов и оптимизации производственного процесса.

Целью данной работы является исследование методов и алгоритмов обработки цифровых изображений, способных обеспечить высокую достоверность контроля поверхностных дефектов листового металлопроката в разных условиях среды производства.

Для достижения поставленной цели необходимо решить следующие задачи:

- Аналитический обзор алгоритмов обработки изображения: Исследование существующих алгоритмов и методов обработки изображений для выявления поверхностных дефектов на металлических листах.
- Подготовка набора данных: подготовить набор данных включающий разнообразные цифровые изображения листового холоднокатаного металлопроката с различными типами поверхностных дефектов.
- Аналитический обзор алгоритмов машинного обучения: Исследование различных методов и алгоритмов машинного обучения для обработки изображений с целью выявления поверхностных дефектов.
- Выбор алгоритмов машинного обучения: рассмотреть классические алгоритмы машинного обучения и алгоритмы, основанные на искусственной нейронной сети.
- Программная реализация: Разработать программное обеспечения на языке программирования Python, для реализации и сравнения различных алгоритмов обработки изображений, включая методы морфологической обработки, алгоритмы машинного обучения и нейронные сети.
- Анализ результатов: произвести сравнительный анализ результатов выбранных алгоритмов.
- Апробация и выход.

1 ТЕОРЕТИЧЕСКИЙ ОБЗОР

1.1 Актуальность

Алгоритмы обработки изображений, благодаря своей эффективности, находят все большее применение в научных изысканиях и практических задачах, охватывая разнообразные области человеческой деятельности [1]. Применение систем компьютерного зрения (КЗ) в частности используется для автоматизации контроля качества в производстве. Алгоритмы КЗ позволяют обнаруживать дефекты продукции с высокой точностью и скоростью, значительно повышая эффективность производственных процессов.

Начальной стадией разрушения металлических деталей является образование микроскопических трещин и царапин на их поверхности, невидимых человеческим глазом, что служит предвестником дальнейших повреждений [2]. Ранее, чтобы определить дефекты на поверхности металла приходилось кропотливо осматривать его, сейчас же, благодаря системам компьютерного зрения, обладающим высокой точностью анализа изображений, становится возможным выявлять даже самые мелкие и незаметные дефекты на поверхности листового металла. [3].

На сегодняшний день можно с уверенностью сказать, что автоматизированная визуальная дефектоскопия, основанная на компьютерном зрении, способна в значительной степени сократить непосредственное участие работников в процессе проверки качества на всех видах производственных линий, отведя человеку роль руководителя процесса. Современные системы контроля качества способны провести подсчет объектов, снять измерения, проверить цвет, комплектность, наличие маркировок и штрихкодов, выявить дефекты и, при необходимости, сопоставить изделие с эталоном.

1.2 Обзор существующих методов анализа поверхностных дефектов

Выбор оптимального метода неразрушающего контроля (НК) является важным этапом при обеспечении надежности и долговечности изделий. Стандартные методы контроля качества выпускаемой продукции часто оказываются неэффективными в обнаружении скрытых дефектов, которые, оставшись незамеченными, могут прогрессировать и приводить к поломкам во время эксплуатации. [4]. Различные методы НК, такие как магнитный, акустический, вихретоковый и оптический контроль, позволяют выявлять дефекты различного типа и размера, однако обладают разной чувствительностью, применимостью и стоимостью [8].

Каждый из подходов имеет свои ограничения. Понимание этих ограничений и сравнительный анализ различных методов НК необходимы для оптимального выбора метода, обеспечивающего максимальную вероятность обнаружения дефектов при минимальных затратах.

1.2.1 Магнитный метод НК

Магнитный метод, используемый для определения механических свойств металла без разрушения образца, является одним из востребованных и часто применяемых способов неразрушающего контроля в различных отраслях промышленности, как в России, так и за ее пределами. [5-6].

Магнитный метод неразрушающего контроля — это один из наиболее распространенных и эффективных методов для выявления поверхностных, подповерхностных и внутренних дефектов в ферромагнитных материалах. Его широкое применение обусловлено относительной простотой реализации, высокой чувствительностью к дефектам, а также возможностью использования в различных отраслях промышленности. Также в промышленном производстве особое внимание уделяется

основополагающим задачам по ресурсосбережению, которые необходимо соблюдать при использовании данного вида неразрушающего контроля [7].

Магнитный метод основан на способности ферромагнитных материалов намагничиваться под воздействием внешнего магнитного поля. При наличии дефекта (трещины, поры, расслоения и т.д.) в намагниченном материале происходит нарушение однородности магнитного поля вблизи дефекта. Это связано с тем, что дефект, представляющий собой воздушный зазор, обладает более высокой магнитной проницаемостью, чем основной материал. В результате, магнитные силовые линии отклоняются от своего первоначального направления, создавая области рассеяния магнитного потока над дефектом.

Рисунок 1 – Устройство для магнитного контроля

1.2.2 Акустический метод НК

Акустический метод неразрушающего контроля – это группа методов, основанных на использовании упругих волн (звука) для обнаружения дефектов в различных материалах. Этот метод широко применяется в промышленности благодаря своей универсальности, возможности контроля изделий сложной формы и относительно высокой чувствительности к различным видам дефектов. Своевременное выявление дефектов играет решающую роль в предотвращении потенциальных аварийных ситуаций, а также позволяет значительно увеличить срок службы оборудования и конструкций. [10].

Этот метод, относящийся к неразрушающим, основан на использовании упругих волн, которые генерируются и затем регистрируются специальным оборудованием в процессе контроля, что обеспечивает возможность выявления дефектов и оценки состояния материалов без нанесения им какого-либо ущерба. [9]. Упругие волны, распространяющиеся в материале, могут быть:

Продольными: Колебания частиц материала происходят вдоль направления распространения волны.

Поперечными: Колебания частиц материала происходят перпендикулярно направлению распространения волны.

Поверхностными: Волны, распространяющиеся вдоль поверхности материала.

При встрече с дефектом (трещиной, порой, включением и т.д.) упругие волны частично отражаются, преломляются, рассеиваются и поглощаются. Анализ отраженных, преломленных и рассеянных волн позволяет выявить наличие, расположение, размер и ориентацию дефектов.

Рисунок 2 – Устройство для акустического контроля

1.2.3 Вихретоковый метод НК

Вихретоковый метод неразрушающего контроля – это электромагнитный метод, основанный на взаимодействии переменного электромагнитного поля с контролируемым объектом для обнаружения дефектов, оценки свойств материала и измерения геометрических параметров. Этот метод широко применяется в различных отраслях промышленности благодаря своей высокой скорости контроля, универсальности и возможности работы с различными типами материалов.

Вихретоковый метод неразрушающего контроля использует анализ взаимодействия внешнего электромагнитного поля с полем вихревых токов, индуцируемых возбуждающей катушкой в электропроводящем объекте. Обнаружение дефектов основывается на фиксации изменений в параметрах поля вихревых токов в месте дефекта [11]. Обработка данных об этих отклонениях позволяет получить информацию о внутренних дефектах [12]. Эти вихревые токи циркулируют в плоскости, перпендикулярной направлению магнитного поля, и их параметры (величина и фаза) зависят от:

Свойства материала: Электропроводности, магнитной проницаемости.

Геометрии изделия: Толщины стенки, расстояния между катушкой и изделием.

Наличия дефектов: Трещины, коррозия, изменения структуры материала, отклонения в размерах и т.д.

Наличие дефектов, изменение свойств материала или геометрии изделия приводят к изменению параметров вихревых токов, которые, в свою очередь, влияют на параметры исходного электромагнитного поля, созданного катушкой. Измеряя эти изменения (например, изменение импеданса катушки, напряжение в катушке), можно выявлять дефекты и оценивать свойства материала.

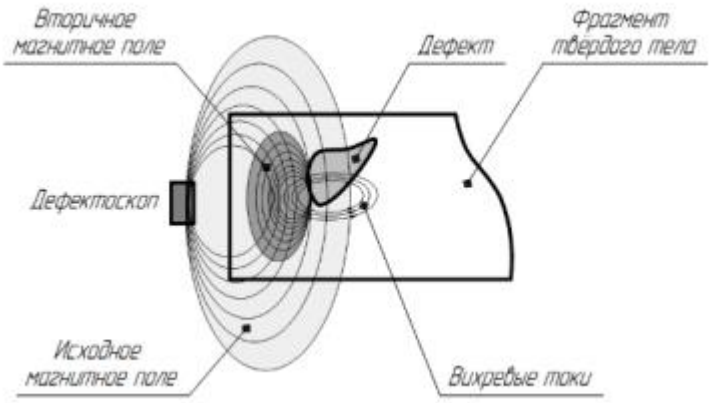

Рисунок 3 – Обнаружение скрытого дефекта с помощью вихретокового дефектоскопа

1.2.4 Оптический метод НК

Оптический метод неразрушающего контроля – это группа методов, использующих взаимодействие электромагнитного излучения оптического диапазона (видимого света, ультрафиолетового и инфракрасного излучения) с контролируемым объектом для выявления дефектов, оценки состояния поверхности, определения размеров и формы объектов.

Технологии машинного зрения, основанные на использовании оптических систем для получения и анализа изображений, демонстрируют растущую значимость в различных технологических процессах, выполняя

функции мониторинга и контроля [14]. Визуальный контроль, включая как ручные методы, так и автоматизированные системы на основе машинного зрения, является одним из наиболее широко используемых и универсальных подходов в современной промышленности.

Оптические методы обладают рядом преимуществ, таких как бесконтактность, быстродействие, высокая точность измерений и возможность визуализации дефектов. Одной из самых главных особенностей оптического метода над всеми другими – это то, что существуют оптические методы с использованием компьютерного зрения представляет собой передовой подход, объединяющий возможности оптических методов для получения изображений с мощными алгоритмами компьютерного зрения для автоматизированного анализа и выявления дефектов. Этот метод значительно расширяет возможности традиционных оптических методов, обеспечивая более высокую скорость контроля, точность и объективность результатов.

Кроме этого, Простота оптического метода заключается в отсутствии необходимости в специальной подготовке образца перед контролем, что соответствует определению неразрушающего метода. Ключевым фактором его применимости является прозрачность материала в определенных областях оптического спектра (видимой, инфракрасной и/или ультрафиолетовой) [13].

1.2.5 Вывод

На основе проведённого поиска, я выделили следующие минусы рассмотренных методов НК:

- Магнитный метод: Для обеспечения надежности выпускаемых деталей на этапах производства, ремонта и диагностики, персонал, задействованный в этих процессах, должен обладать базовыми знаниями и практическими навыками в области магнитопорошкового контроля и других методов магнитного контроля, поскольку они являются неотъемлемой частью производственного цикла [15].
- Акустический метод: Ключевым ограничением современных методов ультразвукового контроля является требование обеспечения надежного акустического контакта между преобразователем и контролируемой деталью, что обычно достигается за счет использования контактной жидкости, такой как индустриальное масло. [11]. Также, Значительным недостатком является высокая трудоемкость, связанная с необходимостью тщательной подготовки поверхности материала перед контролем. Наличие “мертвых зон”, обусловленных сложной геометрией или другими факторами, снижает эффективность метода и требует дополнительных усилий [16].
- Вихретоковый метод: Основным недостатком вихретокового метода НК является возможность взаимовлияния параметров, что может привести к искажению результатов. Кроме того, его применение ограничено электропроводящими материалами, а глубина контроля, обусловленная характером распространения электромагнитных волн, относительно невелика [11].

- Оптический метод: Для данного метода, с использованием компьютерного зрения, я не смог найти недостатки для нашей задачи

Составим сравнительную таблицу рассмотренных методов:

Таблица 1 – Сравнение видов НК

Вид НК	Тип дефекта			Скорость проката до 6 м/с и более	Обеспечение обр. данных	Неинвазивность	Работа с ГО и МО
	Внутренний	Подповерхностный	Поверхностный				
Магнитный	Да	Да	Да	Низкое	Среднее	Да	Нет
Акустический	Да	Да	Да	Низкое	Среднее	Да	Нет
Вихретоковый	Нет	Да	Да	Низкое	Среднее	Да	Нет
Оптический	Нет	Нет	Да	Высокое	Высокое	Да	Да

Так как мы стремимся к обнаружению и классификации поверхностных дефектов, при этом хотим получить максимальную скорость проката и интеграцию с ГО и МО, то самым лучшим вариантом для нас будет использование оптических методов НК.

1.3 Обзор существующих подходов

Качество листового металлопроката является критически важным параметром, определяющим его пригодность для широкого спектра промышленных применений. Поверхностные дефекты, возникающие в процессе производства, могут значительно снижать механические свойства, коррозионную стойкость и эстетическую привлекательность материала, приводя к браку и увеличению затрат. В связи с этим, автоматизация процесса обнаружения и классификации таких дефектов представляет собой актуальную и востребованную задачу.

Традиционные методы контроля качества, основанные на визуальном осмотре [17], являются трудоемкими, субъективными и не всегда позволяют выявлять дефекты на ранних стадиях. В последние годы, благодаря развитию технологий искусственного интеллекта, появились новые возможности для автоматизации этой задачи.

В данном разделе представлен обзор существующих подходов к обнаружению и классификации поверхностных дефектов листового металлопроката с использованием методов и алгоритмов ИИ. Особое внимание уделяется анализу преимуществ и недостатков различных подходов, а также определению перспективных направлений исследований в этой области. Рассматриваются как классические методы машинного обучения, основанные на ручном извлечении признаков, так и современные методы глубокого обучения, позволяющие автоматически извлекать сложные иерархические представления из изображений поверхности. Анализируются гибридные подходы, объединяющие преимущества различных методов, а также подходы, основанные на анализе текстуры и спектральных характеристик поверхности. Данный обзор послужит основой для выбора наиболее эффективных методов и алгоритмов для решения задачи обнаружения и классификации дефектов в рамках настоящей выпускной квалификационной работы.

1.3.1 Обзор методов машинного обучения

На данный момент существует две основные группы методов машинного зрения: традиционные методы и методы глубоко обучения [18].

Методы глубокого обучения (МГО), значительно превосходят традиционные методы машинного обучения во многих задачах [20], особенно в тех, где данные обладают сложной структурой. Вот основные преимущества глубокого обучения перед традиционными подходами:

Во-первых, МГО автоматически извлекает признаки из данных. Нейронные сети учатся выявлять наиболее релевантные признаки для решения задачи. Это особенно важно для сложных данных, таких как изображения, текст и звук. Например, при использовании традиционных методов машинного обучения, алгоритм, разработанный для распознавания лиц при хорошем освещении, может показать снижение точности на 30-50% при плохом освещении, тогда как глубокие сети более устойчивы.

Во-вторых, разработаны специальные архитектуры для работы с неструктурированными данными. Например, рассматриваемые нами сверточные нейронные сети для изображений, полезность которой можно увидеть в научных работах [19]. Например, при задачах классификации изображений с высокой сложностью, традиционные методы машинного обучение, такие как SVM, могут достигать максимальной точности 60%, в то время как глубокие свёрточные нейронные сети достигают 90% и выше.

В-третьих, Производительность обычно улучшается при увеличении объема данных. Глубокие нейронные сети способны извлекать сложные зависимости из больших объемов данных, что позволяет достигать высокой точности.

Также, при использовании традиционных методов приходится вручную на каждом изображении выбирать какие признаки стоит учитывать [26].

Одними из самых популярных СНС на данный момент являются: EfficientNet [21], ResNet [22], VGG [23]. Их мы и будем использовать в нашей работе.

1.3.2 Трансферное обучение

Это метод машинного обучения, при котором знания, полученные при решении одной задачи, используются для решения другой, похожей задачи [24]. Вместо того чтобы обучать модель с нуля для каждой новой задачи,

трансферное обучение позволяет переносить уже изученные признаки и паттерны, что значительно ускоряет процесс обучения и улучшает производительность модели на новой задаче.

Так как мы не обладаем большим количеством данных для обучения с нуля нашей СНС, то будем использовать уже предобученные, чтобы улучшить наши результаты.

Трансферное обучение работает, потому что многие реальные задачи имеют общие черты и закономерности. Например, модель, обученная распознавать объекты на изображениях, может быть использована для распознавания объектов на новых изображениях, даже если эти изображения взяты из другой сферы (например, медицинские изображения вместо обычных фотографий). Модель уже научилась выделять базовые признаки, такие как края, углы и текстуры, которые полезны для многих задач распознавания изображений и тем самым, она будет лучше справляться с новой задачей. Трансферное обучение зарекомендовало себя как эффективный инструмент в широком спектре задач, связанных с дефектоскопией, включая обнаружение, распознавание и классификацию дефектов различной природы. В частности, проводятся активные исследования по адаптации и применению данного подхода для решения задачи распознавания поверхностных дефектов, возникающих в процессе производства и обработки металлопроката [25].

1.4 Обзор набора данных

Для обнаружения и классификации поверхностных дефектов листового металлопроката будем использовать набор данных NEU-surface-defect-database.

Набор данных содержит 1800 изображений в оттенках серого: по 300 образцов с шестью различными типичными дефектами поверхности на каждом из них.

Он содержит следующие 6 видов поверхностных дефектов: Прокатная окалина (RS), заплатка (Pa), деформация (Cr), поверхность с ямками (PS), металловключения (In) и царапины (Sc).

1.5 Экономическое обоснование необходимости автоматизации

Внедрение систем компьютерного зрения (КЗ) на производственных предприятиях представляет собой перспективное направление повышения эффективности, снижения издержек и улучшения качества продукции. Экономическое обоснование такого внедрения базируется на анализе затрат и выгод, связанных с переходом от традиционных методов контроля и автоматизации к современным решениям на основе КЗ.

Внедрение автоматизации на производство приносит ряд значительных преимуществ, включая снижение производственных затрат за счет сокращения брака (что предотвращает дальнейшую обработку дефектных деталей и снижает затраты на ресурсы), снижения затрат на контроль качества (благодаря уменьшению потребности в ручном труде и увеличению скорости контроля), а также уменьшения затрат на гарантийное обслуживание (вследствие повышения качества продукции). Кроме того, автоматизация повышает производительность за счет увеличения скорости производства и возможности круглосуточной работы без перерывов. Наконец, автоматизация улучшает качество продукции, обеспечивая более высокую точность и объективность контроля, что позволяет выявлять мельчайшие дефекты, обеспечивать соответствие строгим стандартам качества и повышать конкурентоспособность предприятия.

1.6 Вывод по теоретическому обзору

Проведенный анализ текущего состояния в области контроля качества листового металлопроката подчеркивает актуальность разработки автоматизированных систем обнаружения и классификации поверхностных дефектов.

Традиционные методы, основанные на визуальном осмотре и других неавтоматизированных подходах, обладают существенными ограничениями в скорости, точности и объективности. В связи с этим, в работе исследуется возможность применения методов искусственного интеллекта, а именно глубокого обучения, для решения поставленной задачи.

Определено, что для реализации практической части исследования будет использован набор данных NEU-surface-defect-database, содержащий изображения с шестью различными типами дефектов поверхности листового металлопроката.

Результаты теоретического анализа и выбор набора данных послужат основой для последующей разработки и сравнения алгоритмов автоматического обнаружения и классификации дефектов, что позволит оценить эффективность применения методов глубокого обучения в данной предметной области.

2 Анализ существующих архитектур

2.1 Архитектура CHC

Свёрточная нейронная сеть — это класс глубоких нейронных сетей, которые особенно хорошо подходят для обработки данных, имеющих структуру, похожую на сетку, таких как изображения, видео, аудио и текст. CHC стали доминирующим подходом во многих задачах компьютерного зрения, а также успешно применяются в других областях.

2.1.1 Свёрточный слой

В основе свёрточных нейронных сетей (CHC) лежит операция свёртки, где обучаемые фильтры (ядра) скользят по входным данным, поэлементно умножаются на них и суммируют результаты, создавая карты признаков, которые указывают на наличие определённых признаков. В отличие от полносвязных слоев, свёрточные слои обладают локальной связностью, то есть нейроны связаны только с небольшой областью входных данных, что позволяет эффективно выявлять локальные признаки и снижать число обучаемых параметров. Благодаря разделению весов, когда все нейроны в карте признаков используют один и тот же фильтр, достигается дальнейшее сокращение параметров и устойчивость к сдвигам входных данных. Свёрточный слой обычно включает несколько фильтров, каждый из которых извлекает различные признаки из входных данных.

2.1.2 Функция активации

ReLU: После каждого свёрточного слоя обычно применяется функция активации. ReLU — это простая, но эффективная функция, которая возвращает входное значение, если оно положительное, и ноль в противном

случае. ReLU помогает сети быстрее обучаться и избежать проблемы затухающего градиента.

2.1.3 Слои пулинга

Слои пулинга уменьшают пространственное разрешение карт признаков, что позволяет снизить количество параметров и вычислительных операций. Пулинг также повышает устойчивость сети к небольшим сдвигам и искажениям входных данных. Наиболее распространенным типом пулинга является max-pooling, который выбирает максимальное значение в каждой небольшой области карты признаков.

2.1.4 Полносвязные слои

В конце СНС обычно находятся один или несколько полносвязных слоев, которые выполняют классификацию признаков, извлеченных свёрточными слоями. Каждый нейрон связан со всеми нейронами предыдущего слоя в полносвязных слоях каждый нейрон связан со всеми нейронами предыдущего слоя.

2.1.5 Выходной слой

Softmax: Для задач многоклассовой классификации используется функция softmax, которая преобразует выходные значения в вероятности для каждого класса.

2.1.6 Как работает СНС

Свёрточные нейронные сети (СНС) начинают с принятия на вход данных, структурированных как сетка (например, изображение). Затем свёрточные слои извлекают локальные признаки, а функция активации ReLU добавляет нелинейность к картам признаков. Слои пулинга уменьшают размерность карт признаков, повышая устойчивость сети к сдвигам. После этого, полносвязные слои классифицируют извлеченные признаки, и, наконец, выходной слой предоставляет вероятности принадлежности к каждому классу в задачах классификации или прогноз в задачах регрессии.

2.2 Архитектура ResNet

ResNet – семейство СНС, которая совершила прорыв в области глубокого обучения, решив проблему затухающего градиента и позволяя успешно обучать очень глубокие сети (до 152 слоев и более).

Одна из основных проблем при обучении глубоких нейронных сетей – это затухающий градиент. При распространении градиента ошибки обратно по сети для обновления весов, градиент может экспоненциально уменьшаться с каждым слоем, особенно в очень глубоких сетях. Это приводит к тому, что веса в ранних слоях не обновляются эффективно, и сеть не обучается.

2.2.1 Ключевая идея ResNet

Остаточные связи**:** ResNet решает проблему затухающего градиента с помощью остаточных связей. Вместо того, чтобы каждый слой обучался отображению входов x в новые признаки $H(x)$, ResNet обучается остаточному отображению $F(x) = H(x) - x$. Затем выход блока ResNet

получается путем добавления входа x к остаточному отображению: $H(x) = F(x) + x$.

2.2.2 Преимущества остаточных связей

Остаточные связи в ResNet решают проблему затухающего градиента, позволяя градиенту ошибки распространяться напрямую, минуя сверточные слои, что помогает сохранить градиент и эффективно обучать даже очень глубокие сети. Эти связи также упрощают обучение, давая сети возможность “выучить” тождественное отображение, если это необходимо, устанавливая веса сверточных слоев в ноль, если они не нужны, и передавая вход без изменений. Благодаря остаточным связям ResNet позволяет обучать сети с огромной глубиной, насчитывающей сотни и даже тысячи слоев.

2.2.3 Основные компоненты ResNet

ResNet начинается с начального слоя, называемого Stem, который включает свёртку 7x7 с шагом 2, батч-нормализацию и функцию активации ReLU. Основу архитектуры составляют остаточные блоки, содержащие два или три свёрточных слоя с остаточной связью. В конце сети применяется глобальное среднее объединение, за которым следует полносвязный слой для классификации. Существуют два типа остаточных блоков: базовый блок, используемый в ResNet-18 и ResNet-34, состоящий из двух свёрточных слоев 3x3 с батч-нормализацией и ReLU, и блок узких мест, используемый в более глубоких сетях, который включает три свёрточных слоя: 1x1, 3x3 и 1x1, где свёртки 1x1 используются для уменьшения и последующего увеличения количества каналов, что позволяет снизить вычислительную сложность.

2.2.4 Преимущества ResNet

ResNet обеспечивает эффективное обучение даже очень глубоких сетей, решая проблему затухающего градиента. Остаточные связи в архитектуре облегчают процесс обучения, позволяя сети “выучить” тождественное отображение. ResNet демонстрирует высокую точность в различных задачах компьютерного зрения и широко используется, благодаря доступности множества предварительно обученных моделей. Решение проблемы градиентов позволяет создавать сети с большим количеством слоев, давая модели возможность изучать более сложные и абстрактные представления данных.

2.2.5 Недостатки ResNet

ResNet обладает более сложной архитектурой по сравнению с более простыми свёрточными нейронными сетями, такими как VGGNet, из-за наличия остаточных соединений и блоков с разнообразными конфигурациями, что может потребовать больше времени на понимание и реализацию. Хотя остаточные соединения способствуют обучению, они также требуют хранения активаций промежуточных слоев, что увеличивает потребление памяти, особенно при обучении очень глубоких сетей. Кроме того, ResNet может быть подвержена переобучению при обучении на небольших наборах данных, поэтому необходимо использовать методы регуляризации, такие как аугментация данных, для предотвращения этой проблемы.

2.3 Архитектура VGGNet:

VGGNet использует последовательные слои сверточных фильтров для изучения особенностей изображений. Каждый сверточный слой применяет

несколько фильтров, за которыми следует нелинейная функция активации. Фильтры захватывают разные аспекты изображения, такие как края и текстуры. VGGNet стала важным этапом в развитии СНС, продемонстрировав, что увеличение глубины сети с использованием небольших фильтров свертки может значительно улучшить производительность.

До VGGNet часто использовались большие фильтры свертки (например, 11x11, 7x7) в ранних слоях СНС. VGGNet ставила целью исследовать, как глубина сети влияет на точность, используя только небольшие фильтры свертки размером 3x3. Авторы предполагали, что более глубокие сети с меньшими фильтрами могут быть более эффективными в извлечении признаков.

2.3.1 Ключевые особенности VGGNet

VGGNet отличается значительной глубиной, например, VGG16 и VGG19 содержат 16 и 19 сверточных слоев соответственно. Кроме того, архитектура VGGNet характеризуется однородностью, в основном, за счет использования сверточных слоев с фильтрами 3x3 и слоев max-pooling 2x2 с шагом 2, что упрощает проектирование и понимание сети. Ключевой особенностью также является использование небольших сверточных фильтров размером 3x3.

2.3.2 Основные компоненты VGGNet

VGGNet состоит из нескольких последовательных блоков, каждый из которых содержит несколько сверточных слоев 3x3, за которыми следует слой max-pooling 2x2. В конце сети расположены три полносвязных слоя.

Архитектура VGGNet включает в себя последовательность сверточных слоев 3x3 с ReLU, количество которых варьируется в каждом блоке, за

которыми следуют слои max-pooling 2x2 с шагом 2, предназначенные для уменьшения размера карты особенностей. Далее располагаются три полносвязных слоя: два слоя с 4096 нейронами каждый и выходной слой с количеством нейронов, соответствующим числу классов. В завершение применяется функция Softmax для получения вероятностей классов.

2.3.3 Преимущества VGGNet

VGGNet отличается простотой, благодаря однородной и легко понимаемой архитектуре. Она наглядно демонстрирует эффективность увеличения глубины сети, использует маленькие сверточные фильтры размером 3x3 и обладает хорошей переносимостью признаков (Transfer learning), что позволяет использовать признаки, изученные VGGNet на больших наборах данных, таких как ImageNet, для других задач компьютерного зрения.

2.3.4 Недостатки VGGNet

VGGNet имеет очень большое количество параметров, например, около 138 миллионов в VGG16, что требует значительного объема памяти для хранения и вычислений, делая ее менее подходящей для развертывания на устройствах с ограниченными ресурсами или для задач, требующих быстрой обработки. Большая глубина VGGNet, достигающая 16-19 слоев, может приводить к проблемам исчезающего градиента, затрудняя процесс обучения. Кроме того, из-за большого количества параметров и слоев VGGNet требует значительных вычислительных ресурсов для обучения и использования, что может потребовать мощного оборудования и больших временных затрат.

2.4 Архитектура EfficientNet

EfficientNet — семейство сверточных нейронных сетей, которое позволяет достичь высокой точности при меньшем количестве вычислительных ресурсов по сравнению с предыдущими архитектурами. Главная особенность EfficientNet заключается в эффективном масштабировании базовой сети, получившей название **EfficientNet-B0**, для достижения высокой точности и эффективности (меньшего количества параметров и вычислений) по сравнению с другими СНС.

Традиционно, СНС масштабируют, увеличивая глубину (количество слоев), ширину (количество каналов в слоях) или разрешение входного изображения. Однако, такое масштабирование часто выполняется эмпирически (путем проб и ошибок) и может привести к: Снижению точности: Несбалансированное масштабирование может привести к переобучению. Увеличению вычислительных затрат: Простое увеличение всех параметров может привести к чрезмерному увеличению количества операций и, как следствие, к замедлению работы.

2.4.1 Ключевые особенности EfficientNet

EfficientNet предлагает новый подход к масштабированию, названный сложное масштабирование. Идея заключается в том, чтобы масштабировать все три измерения (глубину, ширину и разрешение) согласованно, используя набор коэффициентов, определенный с помощью архитектурного поиска (АП).

Формально:

Глубина:

$$d = \alpha^{\phi}, \tag{1}$$

Ширина:

$$w = \beta^{\phi}, \tag{2}$$

Разрешение:

$$r = \gamma^{\phi}, \tag{3}$$

Где α, β, γ – константы, определённые с помощью АП, задающие пропорции масштабирования каждого измерения;

ϕ – коэффициент масштабирования, определяемый пользователем, который контролирует общее количество ресурсов, выделяемых на масштабирование.

2.4.2 Основные компоненты EfficientNet-B0

EfficientNet использует блоки MBConv, которые применяют inverted residual blocks для снижения числа параметров и вычислений, а также depthwise separable convolutions для дальнейшего уменьшения вычислительной сложности. Кроме того, архитектура использует блоки Squeeze-and-Excitation для адаптивной перекалибровки каналов в карте особенностей, позволяя сети уделять больше внимания наиболее важным каналам.

2.4.3 Преимущества EfficientNet

EfficientNet обеспечивает высокую точность в различных задачах компьютерного зрения, таких как классификация изображений и обнаружение объектов. Она характеризуется высокой эффективностью, требуя значительно меньшее количество параметров и вычислений по сравнению с другими свёрточными нейронными сетями, что делает ее более быстрой и энергоэффективной. EfficientNet является универсальной

моделью, применимой для различных задач компьютерного зрения, и обладает простотой масштабирования, позволяющей легко создавать модели разной вычислительной сложности. Благодаря своей эффективной архитектуре, EfficientNet часто обучается быстрее, чем другие СНС, достигая высокой точности за меньшее время.

2.4.4 Недостатки EfficientNet

Несмотря на то, что архитектура EfficientNet относительно проста по сравнению с некоторыми другими моделями, она все же сложнее базовых сверточных нейронных сетей, таких как VGGNet, что может усложнить понимание и реализацию. EfficientNet, несмотря на свою масштабируемость и эффективность, требует большого объема данных для достижения наилучших результатов, и другие архитектуры могут быть более подходящими для обучения с нуля на небольших наборах данных. Несмотря на свою общую превосходную эффективность, EfficientNet может не превосходить другие архитектуры на небольших задачах классификации изображений, особенно при ограниченном объеме данных, в таких случаях, архитектуры, разработанные специально для небольших наборов данных, могут показывать лучшие результаты.

2.5 Архитектура ViT

ViT (Vision Transformer) - это архитектура глубокого обучения. Она произвела революцию в компьютерном зрении, успешно перенеся архитектуру Transformer, изначально разработанную для обработки естественного языка, на задачи классификации изображений.

2.5.1 Ключевые особенности ViT

Основная идея: в отличие от сверточных нейронных сетей, которые извлекают признаки локально и постепенно, ViT рассматривает изображение

как последовательность патчей (небольших фрагментов изображения) и применяет к этой последовательности стандартный Transformer Encoder. Это позволяет модели улавливать глобальные взаимосвязи между различными частями изображения, что может быть важно для решения сложных задач.

2.5.2 Основные компоненты ViT

В архитектуре Vision Transformer (ViT) входное изображение разбивается на неперекрывающиеся патчи фиксированного размера, например, изображение 224x224 пикселей может быть разбито на 196 патчей размером 16x16 пикселей каждый. Затем, после векторизации, каждый патч линейно проецируется в d-мерное пространство вложения, создавая “токен патча”. К началу последовательности вложений патчей добавляется специальный обучаемый токен класса ([CLS]), состояние которого после прохождения через Transformer используется для классификации изображения. Для кодирования информации о положении патчей в изображении, к вложениям патчей добавляются вложения позиции (обучаемые или фиксированные). Полученная последовательность вложений (токен класса, вложения патчей и вложения позиции) передается через стандартный Transformer Encoder. И, наконец, после Transformer Encoder, состояние токена класса используется для классификации изображения, что обычно реализуется многослойным персептроном с одним или несколькими полносвязными слоями и функцией активации softmax для получения вероятностей классов.

2.5.3 Преимущества ViT

ViT, в отличие от свёрточных нейронных сетей (СНС), обладает глобальным пониманием контекста, захватывая глобальные зависимости между различными частями изображения благодаря механизму само-

внимания, что позволяет сети понимать контекст в целом и выявлять долгосрочные связи между различными регионами изображения. ViT хорошо масштабируется с увеличением объема данных, демонстрируя значительное улучшение производительности при обучении на очень больших наборах данных, превосходя традиционные CHC. В отличие от CHC, которые в значительной степени полагаются на сверточные слои для извлечения признаков, ViT обучается извлекать признаки непосредственно из данных, требуя меньше ручных настроек.

2.5.4 Недостатки ViT

ViT требует обучения на огромных наборах данных, таких как JFT-300M или ImageNet-21K, для достижения хорошей производительности, в противном случае модель склонна к переобучению, особенно на меньших наборах данных. Само-внимание имеет квадратичную сложность по отношению к количеству патчей, что делает ViT вычислительно затратной для изображений высокого разрешения, увеличивая время и ресурсы, необходимые для обучения и вывода, особенно для больших изображений. Производительность ViT может быть чувствительной к размеру патча: слишком маленькие патчи могут увеличить вычислительные затраты, а слишком большие могут привести к потере важных деталей.

2.6 Архитектура SWIN

Swin Transformer - это иерархическая архитектура Transformer для компьютерного зрения. Она была разработана для решения некоторых ограничений оригинального Vision Transformer (ViT) и обеспечения большей эффективности и масштабируемости, особенно при работе с изображениями высокого разрешения и задачами, требующими локального контекста.

2.6.1 Основные компоненты SWIN

В архитектуре Swin Transformer, подобно ViT, входное изображение сначала разбивается на неперекрывающиеся патчи фиксированного размера, чаще всего 4x4 пикселя, а затем каждый патч линейно проецируется в d-мерное пространство вложения. Основным строительным блоком Swin Transformer является Swin Transformer Block, который включает окно с несколькими головками для самоконтроля, где многоголовочное само-внимание вычисляется внутри каждого локального окна, и смещенное окно с несколькими головками для самонаблюдения, где в последующих блоках окна сдвигаются, обеспечивая взаимодействие информации между разными окнами. После вычисления внимания, данные проходят через двухслойный многослойный перцептрон с функцией активации GELU, а нормализация слоя и остаточные соединения используются для стабилизации обучения и улучшения производительности. После нескольких Swin Transformer блоков размерность представления уменьшается с помощью слоев объединения патчей, которые объединяют соседние патчи в группы, уменьшая количество токенов и создавая иерархическую структуру, где каждый уровень обрабатывает признаки на разных масштабах. В завершение, в зависимости от задачи, после последних Swin Transformer блоков добавляется классификационная голова (для классификации изображений) или другие модули (для обнаружения или сегментации объектов).

2.6.2 Преимущества SWIN

Swin Transformer отличается высокой эффективностью и масштабируемостью, превосходя ViT, особенно для изображений высокого разрешения, благодаря оконному вниманию и иерархической структуре. Эта архитектура строит иерархическое представление изображений, позволяя сети обрабатывать признаки на разных масштабах, что важно для задач, требующих понимания контекста и локализации объектов. Кроме того, Swin

Transformer может достигать хорошей производительности при меньшем количестве данных по сравнению с ViT.

2.6.3 Недостатки SWIN

Swin Transformer имеет более сложную архитектуру по сравнению с ViT, что может потребовать больше времени на понимание и реализацию. Сдвинутые окна, используемые в Swin Transformer, добавляют сложности в реализацию и могут потребовать применения специализированных операций.

2.7 Метрики для оценок модели

Выбор правильной метрики для оценки модели машинного обучения - критически важен для понимания её реальной производительности и сравнения с другими моделями. Разные метрики подходят для разных задач и типов данных. Так как у нас исследование посвящено классификации, то рассмотрим метрики для задач классификации:

2.7.1 Метрики для задач классификации:

1. **Accuracy (Точность) -** доля правильно классифицированных объектов:
   Формула**:**

$$Accuracy = \frac{(TP + TN)}{TP + TN + FP + FN}$$

Применение**:** Подходит для сбалансированных классов (когда количество объектов каждого класса примерно одинаково).

2. **Precision (Точность) -** доля правильно предсказанных положительных объектов среди всех объектов, предсказанных как положительные:
   Формула**:**

$$Precision = \frac{TP}{TP + FP}$$

Применение**:** Важна, когда нужно минимизировать количество ложноположительных срабатываний (FP). Например, в медицинской диагностике, когда важно не поставить ложный диагноз.

3. **Recall (Полнота) -** доля правильно предсказанных положительных объектов среди всех реальных положительных объектов:
   Формула**:**

$$Recall = \frac{TP}{TP + FN}$$

Применение**:** Важна, когда нужно минимизировать количество ложноотрицательных срабатываний (FN). Например, в системе обнаружения мошенничества, когда важно не пропустить ни одного мошеннического действия.

4. **F1-Score (F1-мера) –** среднее гармоническое между точностью и полнотой:
   Формула**:**

$$F1 - score = \frac{2 * Precision * Recall}{Precision + Recall}$$

Применение**:** Хорошо подходит для несбалансированных классов, так как учитывает и точность, и полноту.

2.8 Вывод

Таблица 2 – Сравнительные характеристики рассмотренных архитектур

Характеристика	**ResNet**	**EfficientNet**	**VGGNet**	**Swin Transformer**	**(ViT)**
Базовая идея	Остаточные соединения	Compound Scaling	Использование небольших сверточных	Иерархический Transformer с оконным вниманием.	Применение Transformer Encoder непосредственно к патчам изображений.
Глубина сети	Очень глубокая (до 152 слоев и больше)	Зависит от варианта, глубокая	глубокая (16-19 слоев)	Зависит от варианта, глубокая	Зависит от варианта, глубокая
Количество параметров	Может быть большим для очень глубоких ResNet	Значительно меньше, чем у других моделей	Очень много, VGG16 ~ 138 млн	Варьируется, меньше чем у ViT	Варьируется, может быть очень большим,
Тип внимания	Отсутствует	Отсутствует	Отсутствует	Оконное внимание и сдвинутое оконное внимание	Глобальное само-внимание

Требования к данным	Требует относительно большого объема данных для обучения, но меньше, чем ViT.	Требует относительно большого объема данных для обучения, но часто меньше, чем ResNet и VGGNet.	Требует большого объема данных для обучения, склонна к переобучению	Требует меньше данных, чем ViT, но все равно нуждается в приличном наборе.	Требует огромного объема данных для обучения, чтобы избежать переобучения.
Вычислительная сложность	Умеренная, зависит от глубины сети.	Относительно низкая, оптимизирована для эффективности.	Высокая, особенно для глубоких вариантов.	Относительно высокая, но эффективнее чем ViT.	Очень высокая, из-за квадратичной сложности глобального внимания.

Каждая из этих архитектур обладает уникальными преимуществами, которые делают их подходящими для задачи классификации изображений. ResNet хорошо зарекомендовала себя благодаря глубоким слоям и остаточным соединениям. EfficientNet делает акцент на эффективном использовании ресурсов. Swin Transformer предлагает синергию между локальным и глобальным контекстом и подходит для высококачественных изображений. ViT, используя Transformer, имеет сильные стороны в глобальном контексте, но, как правило, требует больше данных.

Несколько пунктов, которые выделяют каждую из архитектур:

- **Swin Transformer и ViT** - перспективные, но ресурсоемкие архитектуры, требующие больших объемов данных. Swin Transformer, как правило, предпочтительнее ViT.
- **EfficientNet** является отличным выбором, когда необходима высокая эффективность.
- **ResNet** остается надежной и проверенной архитектурой, особенно при ограниченных ресурсах или небольшом объеме данных.
- **VGGNet** скорее всего не очень подойдёт для нашего исследования, так как она требует большого объёма данных для обучения, у неё высокая вычислительная сложность и требует большого объёма памяти.

3 РЕАЛИЗАЦИЯ

3.1 Аугментация данных

Аугментация данных — это важнейший метод предварительной обработки, используемый в машинном обучении, особенно в задачах, связанных с изображениями. Метод служит разным целям, но его использование может значительно улучшить производительность, обобщающую способность и стабильность моделей.

Основная задача аугментации заключается в увеличении разнообразия обучающей выборки путем создания новых, слегка измененных версий существующих данных. Это помогает модели лучше обобщать и избегать переобучения, особенно когда обучающая выборка ограничена.

Методы аугментации зависят от типа данных. Для изображений это геометрические преобразования (повороты, сдвиги, масштабирование, отражения, сдвиг, растяжение) и цветовые преобразования (яркость, контраст, насыщенность), а также обрезка, скрытие части изображения. При применении нужно соблюдать ограничения (не менять класс), применять только к обучающей выборке.

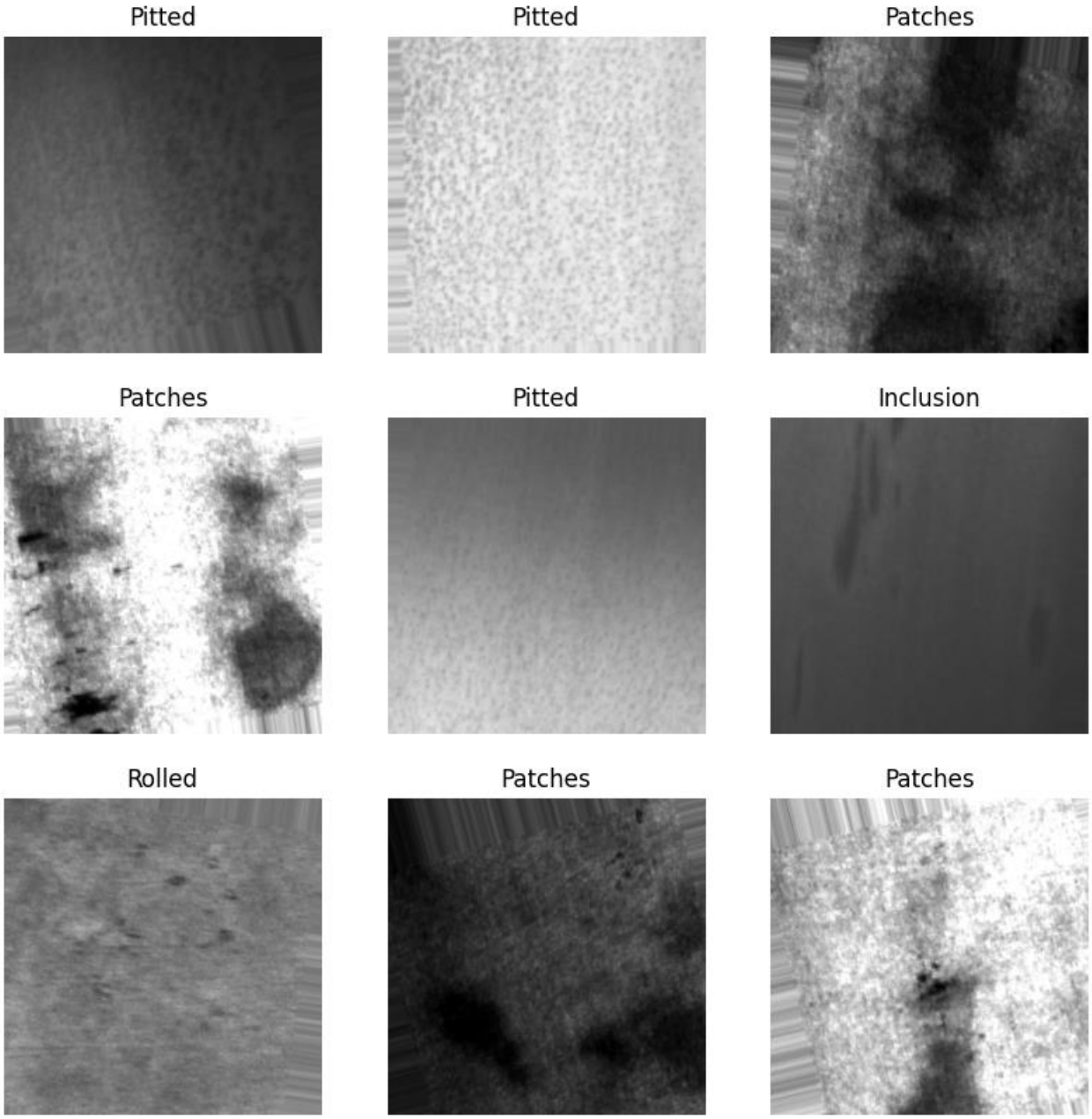

Рисунок 4 - Пример аугментированных данных

3.2 Разработка и обучение модели EfficientNet

3.2.1 Разработка и обучение нейронной сети

На основе архитектуры CHC на базе EfficientNetB3 был разработан программный код на языке python. Полный код представлен в приложении А. Для компиляции модели были использованы оптимизатор Adam и Категориальная перекрёстная энтропия в качестве функции потерь.

Набор данных для обучения модели был обработан и разделён на обучающую, валидационную и тестовую в соотношении 92%, 4%, 4% соответственно. После этого, обучил модель на 20-ти эпохах.

3.2.2 Оценка результатов работы

После обучения были получены следующие результаты:

Accuracy составила 0.47

Precision составила 0.55

Recall составила 0.49

F1-score составила 0.34

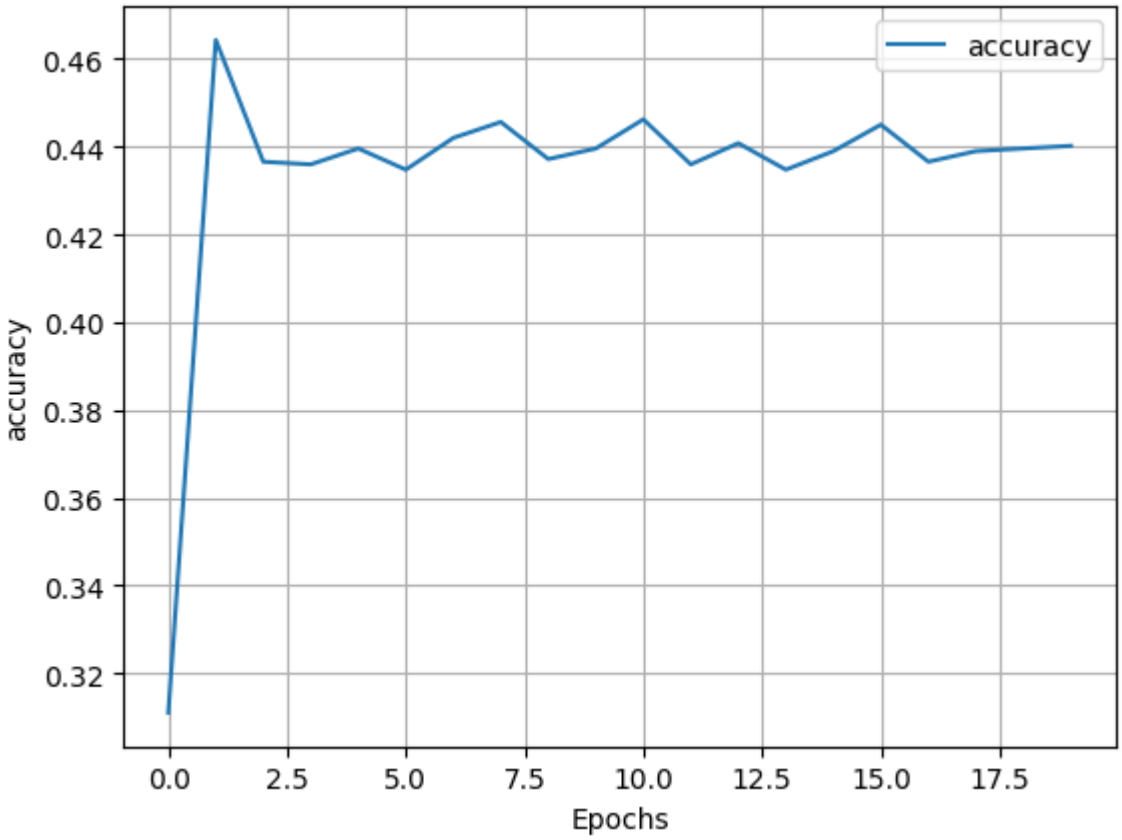

Рисунок 5 – Зависимость accuracy от эпохи

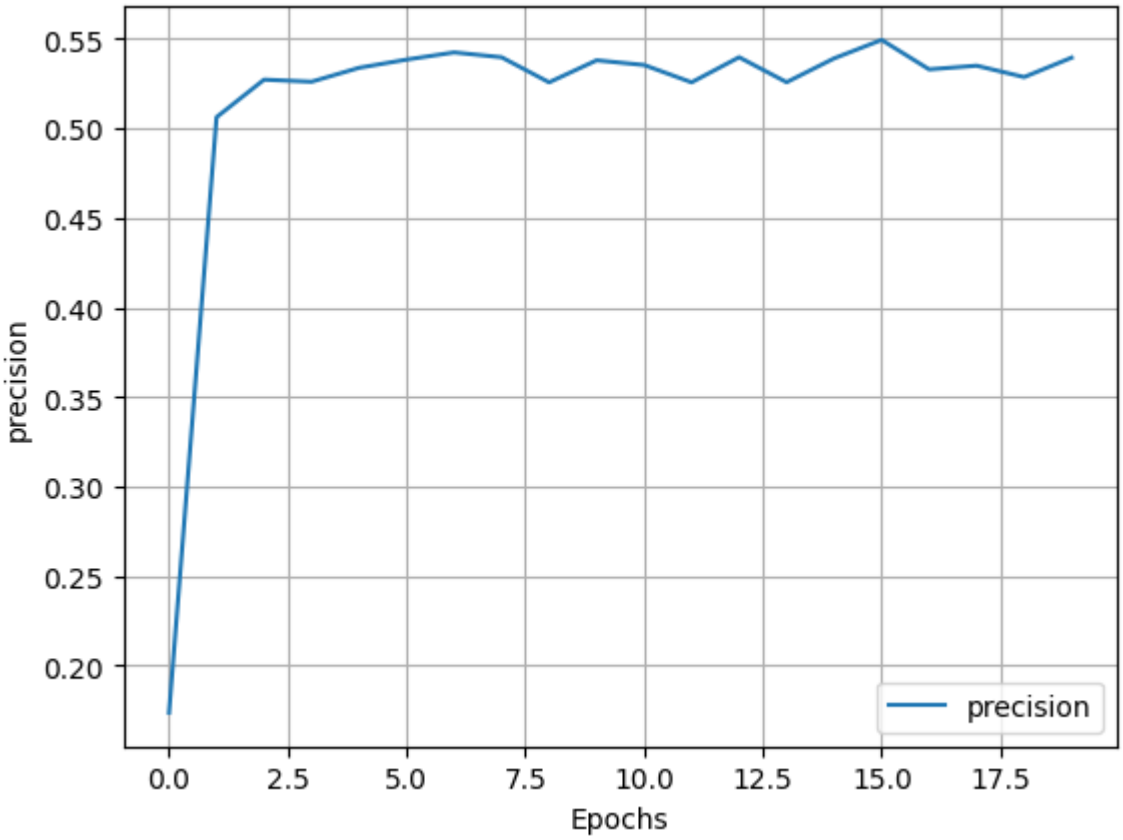

Рисунок 6 – Зависимость precision от эпохи

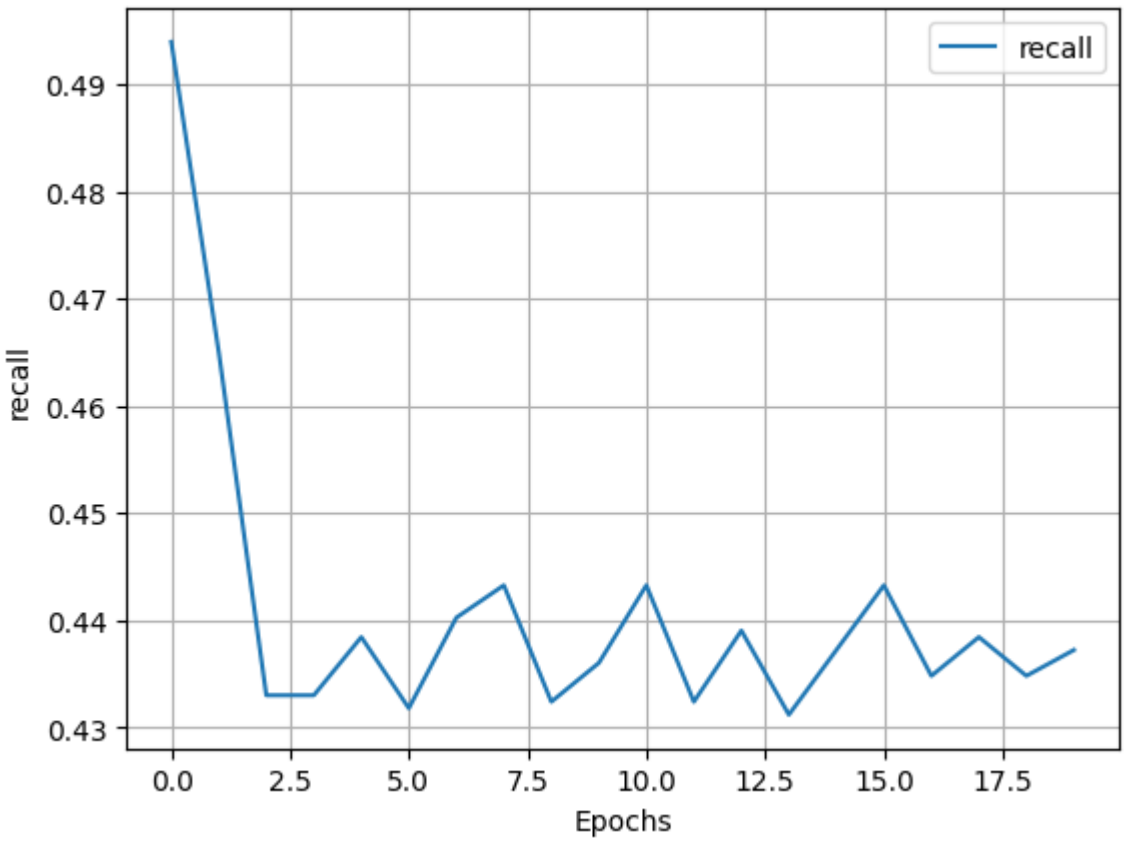

Рисунок 7 – Зависимость recall от эпохи

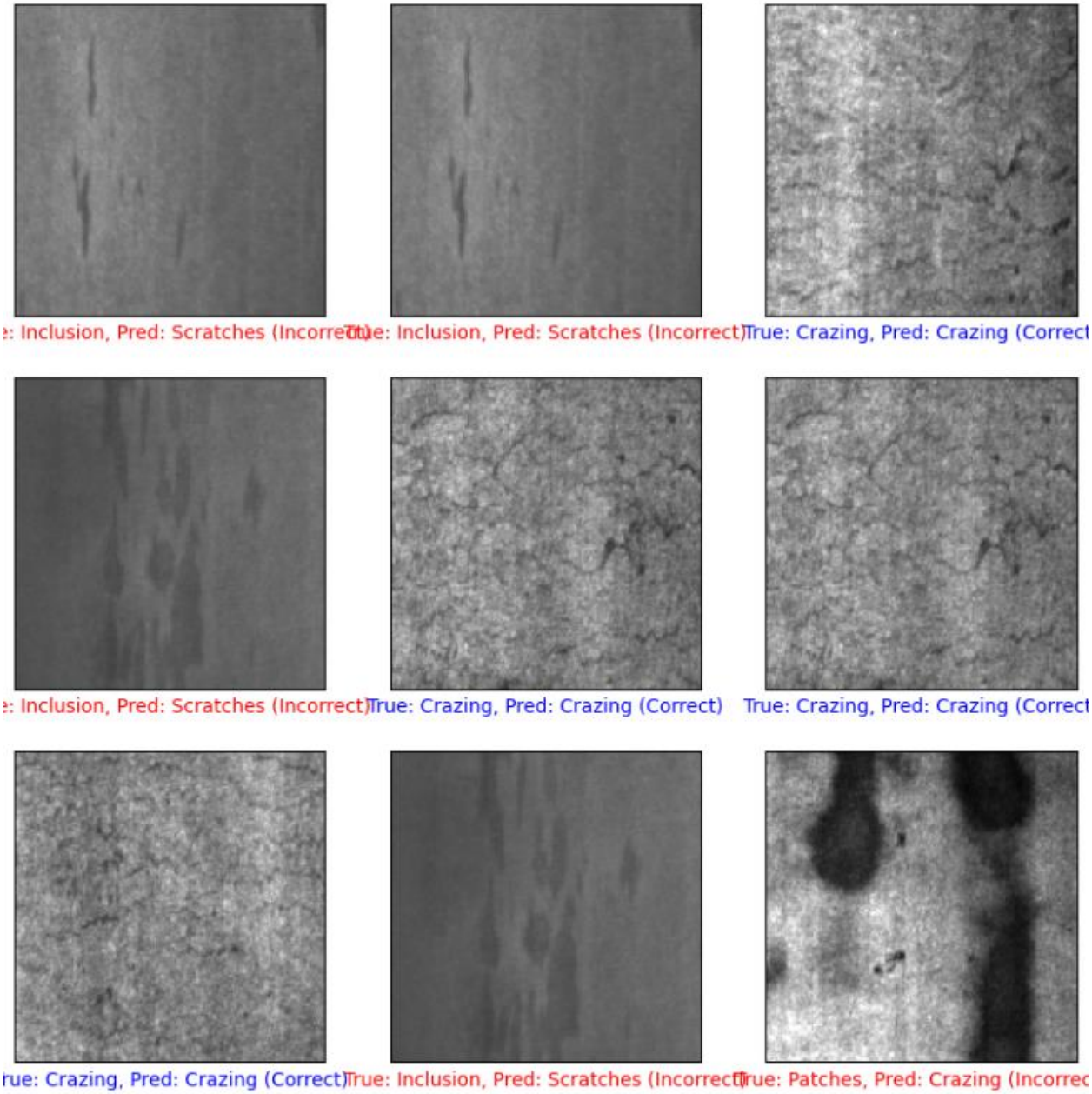

Рисунок 8 – Пример результата на тестовых данных

3.3 Разработка и обучение модели ResNet

3.3.1 Разработка и обучение нейронной сети

На основе архитектуры СНС на базе ResNet был разработан программный код на языке python. Полный код представлен в приложении Б. Для компиляции модели были использованы оптимизатор Adam и Категориальная перекрёстная энтропия в качестве функции потерь.

Набор данных для обучения модели был обработан и разделён на обучающую, валидационную и тестовую в соотношении 92%, 4%, 4% соответственно. После этого, обучил модель на 20-ти эпохах.

3.3.2 Оценка результатов работы

После обучения были получены следующие результаты:

Accuracy составила 0.9

Precision составила 0.67

Recall составила 0.93

F1-score составила 0.9

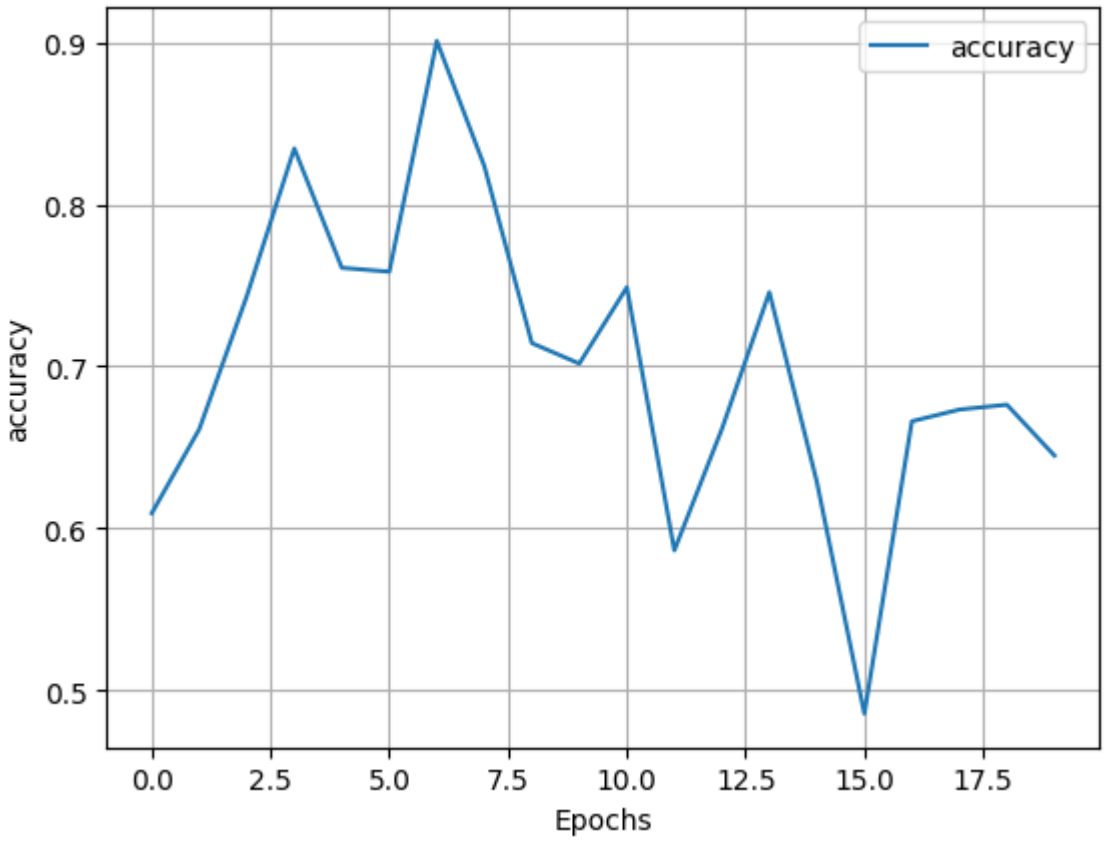

Рисунок 9 - Зависимость accuracy от эпохи

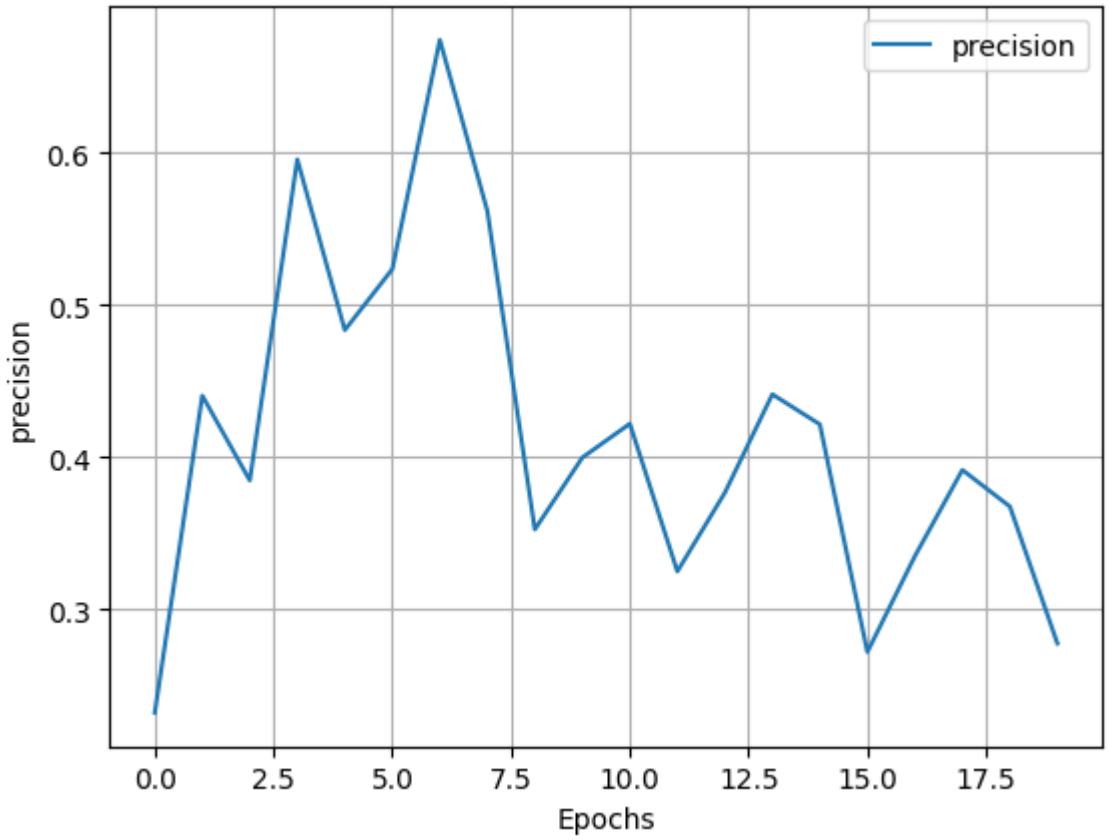

Рисунок 10 - Зависимость precision от эпохи

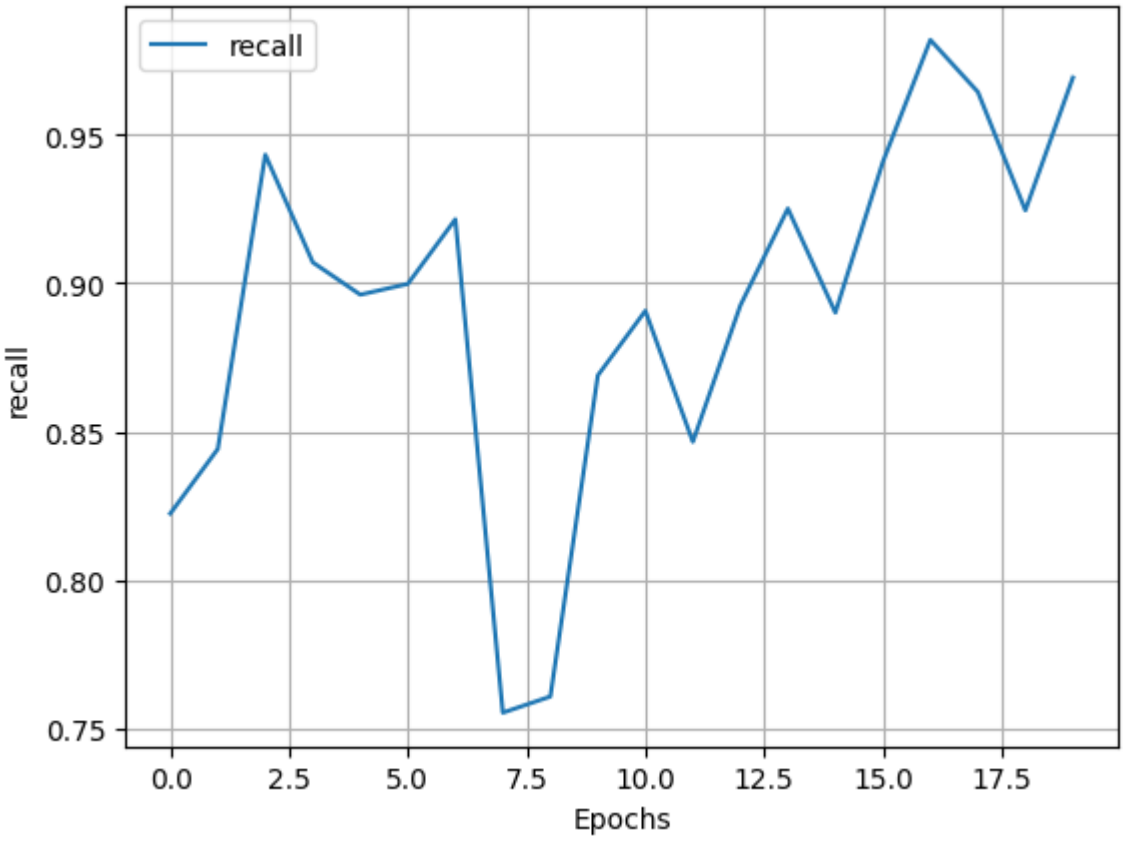

Рисунок 11 - Зависимость recall от эпохи

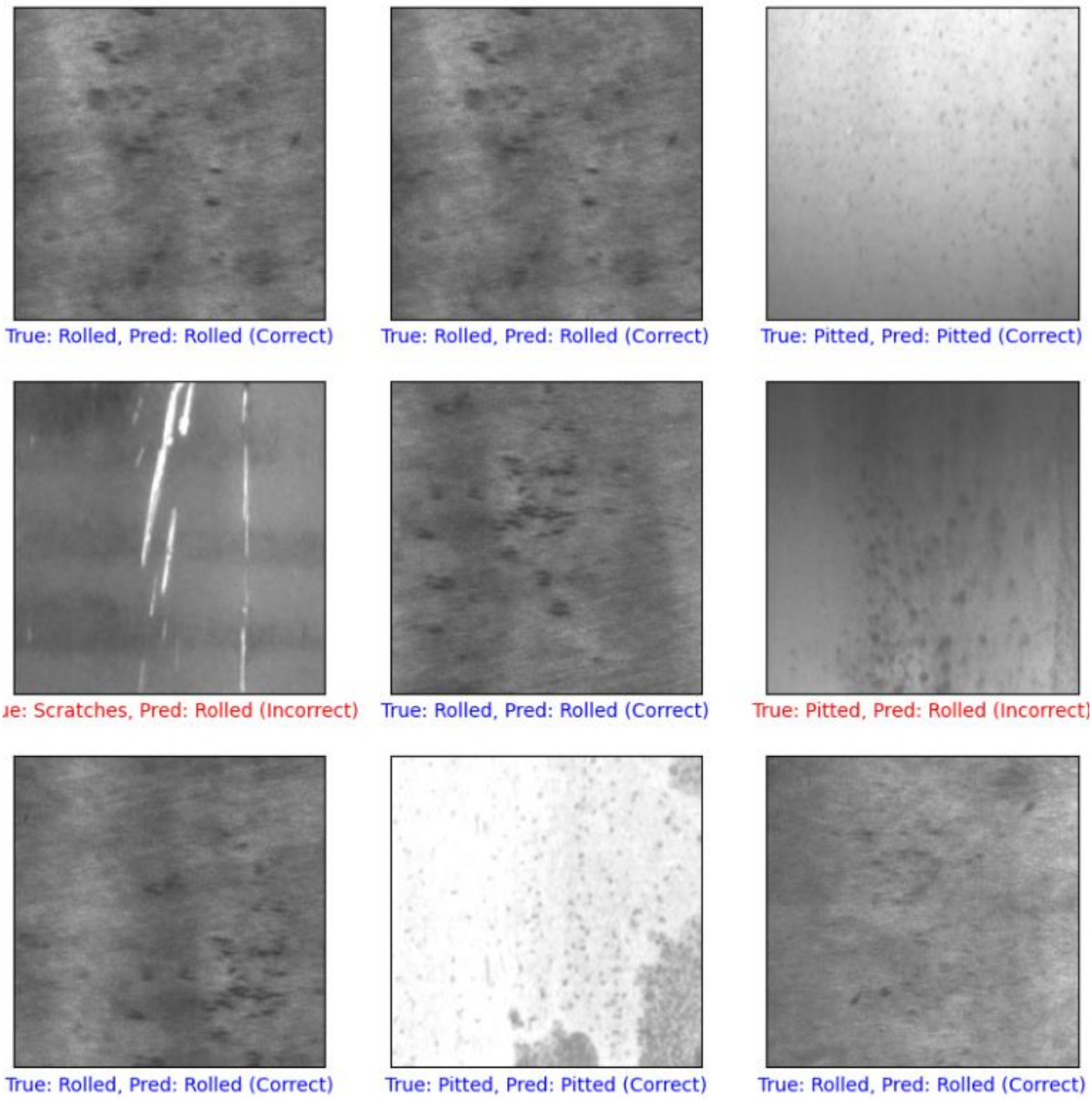

Рисунок 12 - Пример результата на тестовых данных

3.4 Разработка и обучение модели с собственной архитектурой

3.4.1 Разработка и обучение нейронной сети

Была разработана собственная модель. Архитектура этой модели представлена в приложении В. Полный код представлен в приложении Г. Для компиляции модели были использованы оптимизатор Adam и Категориальная перекрёстная энтропия в качестве функции потерь.

Набор данных для обучения модели был обработан и разделён на обучающую, валидационную и тестовую в соотношении 92%, 4%, 4% соответственно. После этого, обучил модель на 20-ти эпохах.

3.4.2 Оценка результатов работы

После обучения были получены следующие результаты:

Accuracy составила 0.95

Precision составила 0.95

Recall составила 0.94

F1-score составила 0.95

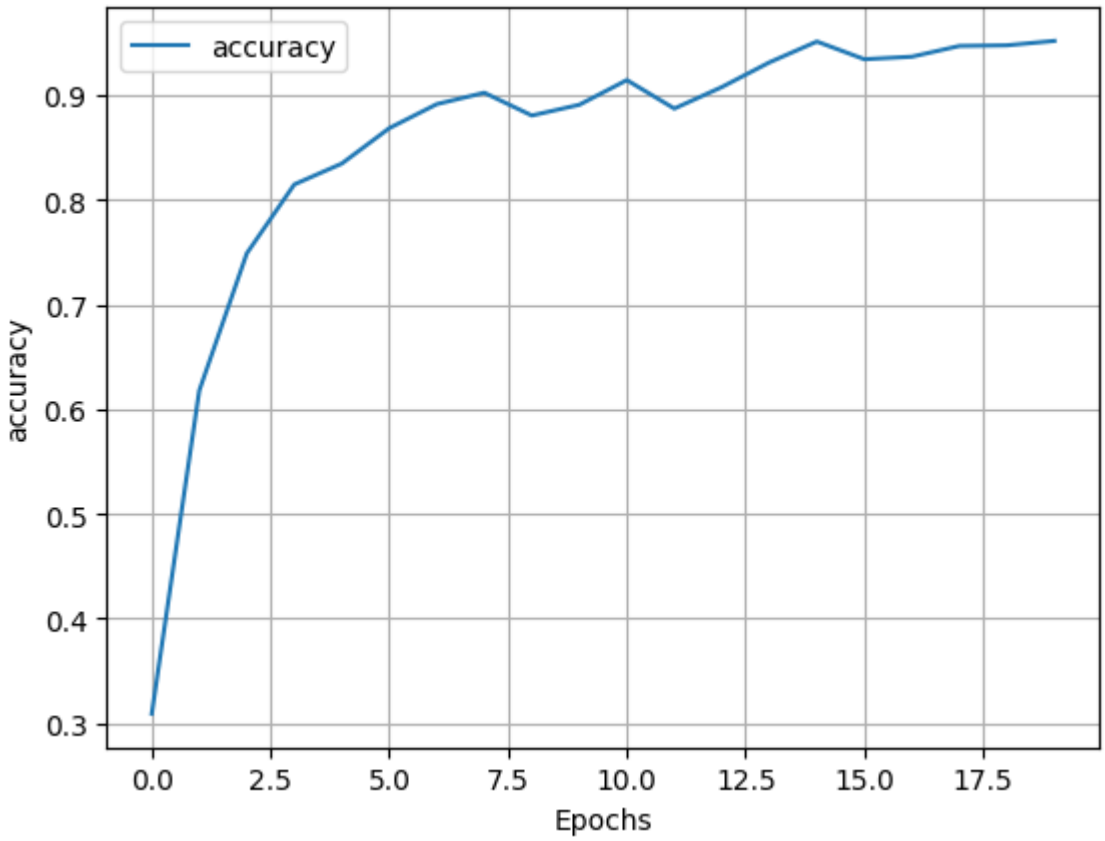

Рисунок 13 - Зависимость accuracy от эпохи

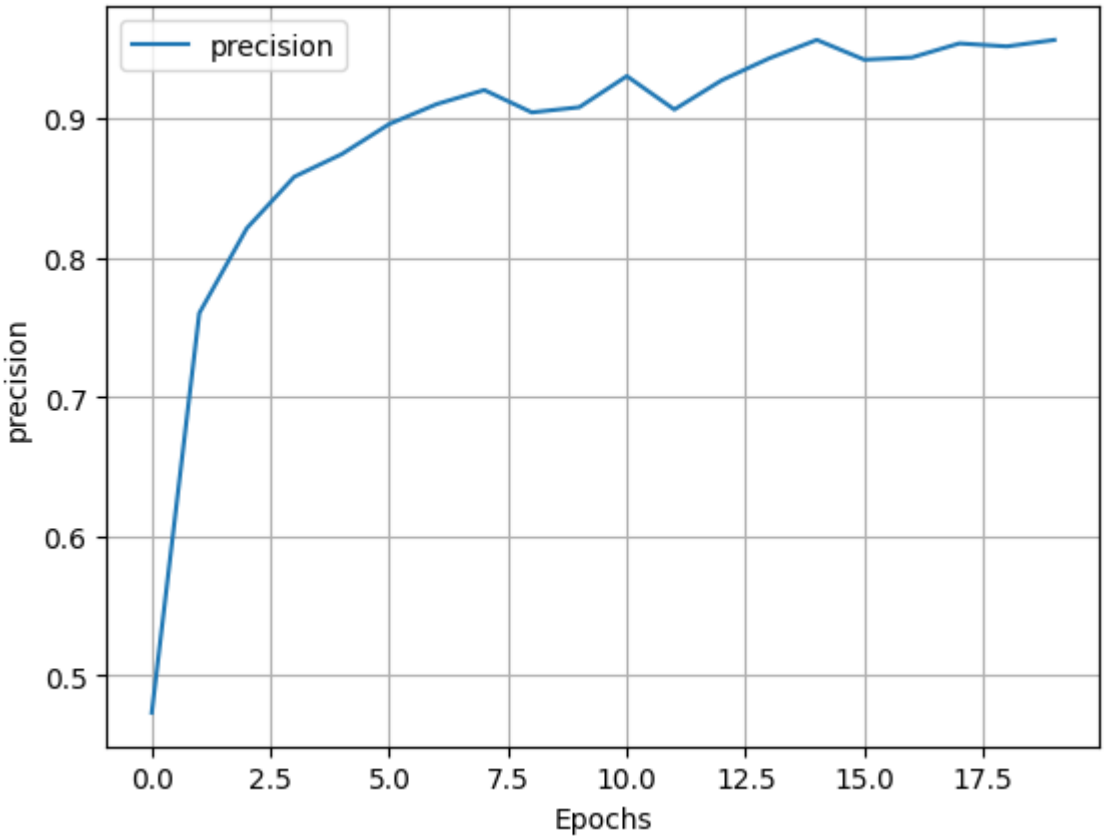

Рисунок 14 - Зависимость precision от эпохи

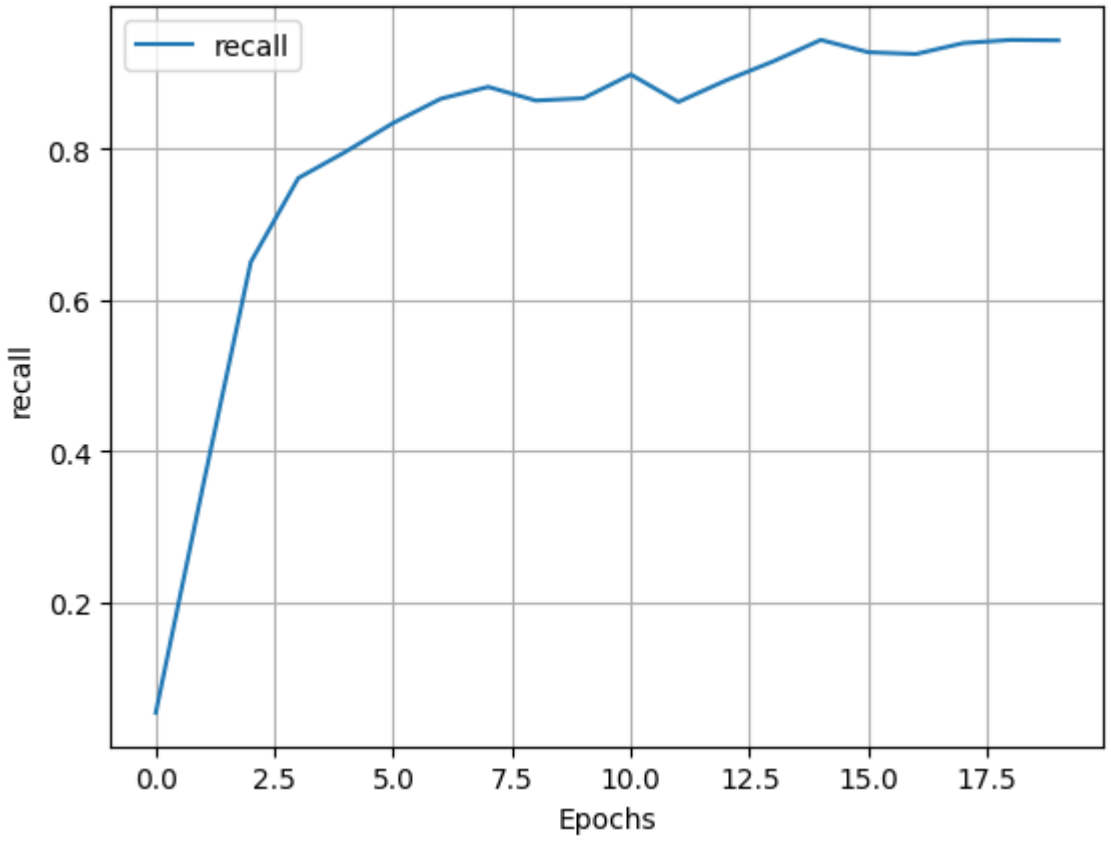

Рисунок 15 - Зависимость recall от эпохи

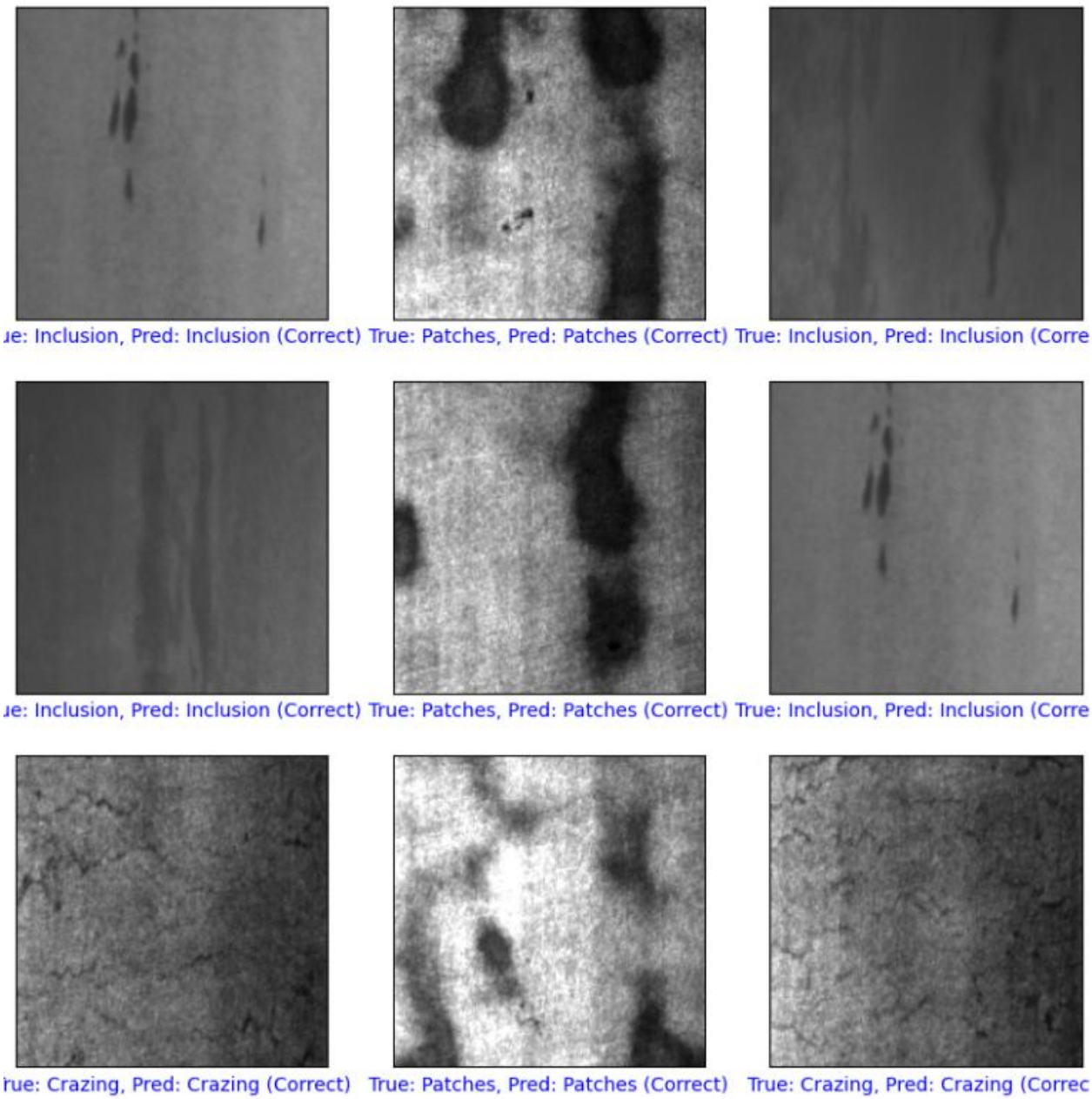

Рисунок 16 - Пример результата на тестовых данных

3.5 Разработка и обучение модели VggNet

3.5.1 Разработка и обучение нейронной сети

На основе архитектуры СНС на базе VggNet был разработан программный код на языке python. Полный код представлен в приложении Д. Для компиляции модели были использованы оптимизатор Adam и Категориальная перекрёстная энтропия в качестве функции потерь.

Набор данных для обучения модели был обработан и разделён на обучающую, валидационную и тестовую в соотношении 92%, 4%, 4% соответственно. После этого, обучил модель на 5-ти эпохах.

3.5.2 Оценка результатов работы

После обучения были получены следующие результаты:

Accuracy составила 0.16

Precision составила 0.17

Recall составила 0.21

F1-score составила 0.14

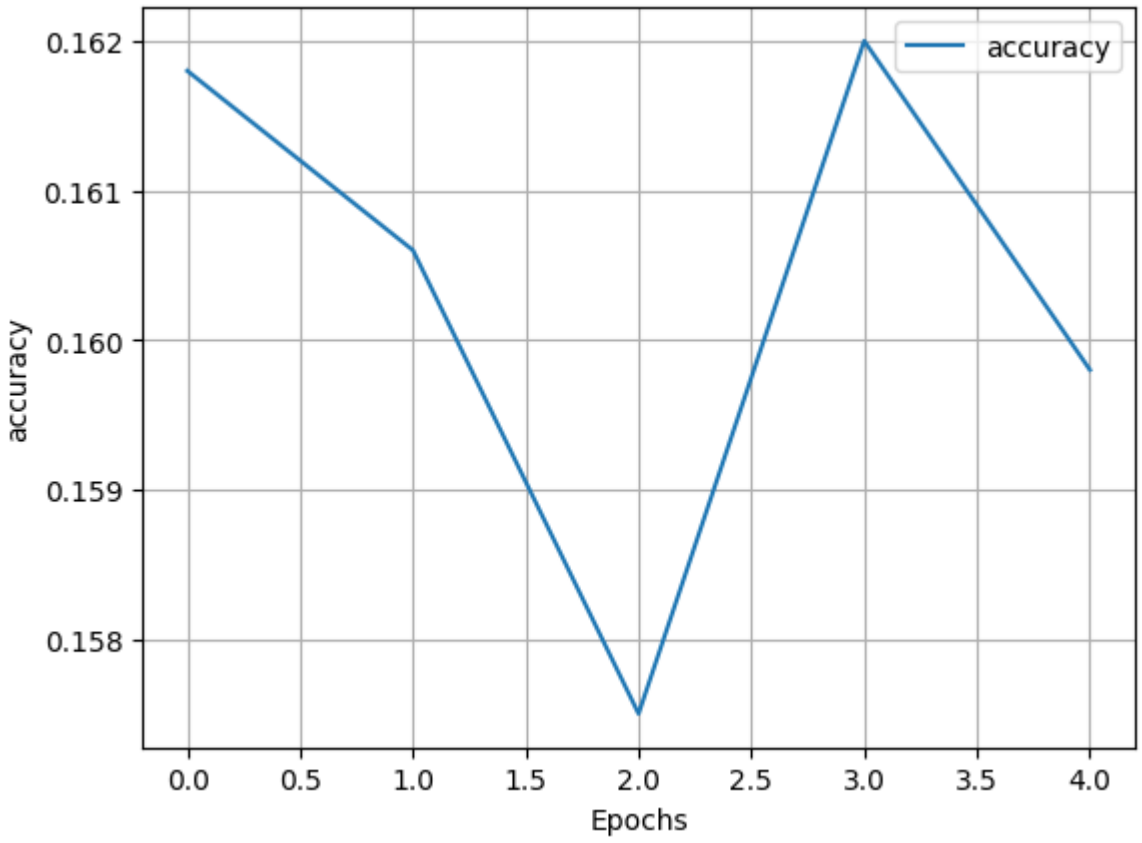

Рисунок 17 - Зависимость accuracy от эпохи

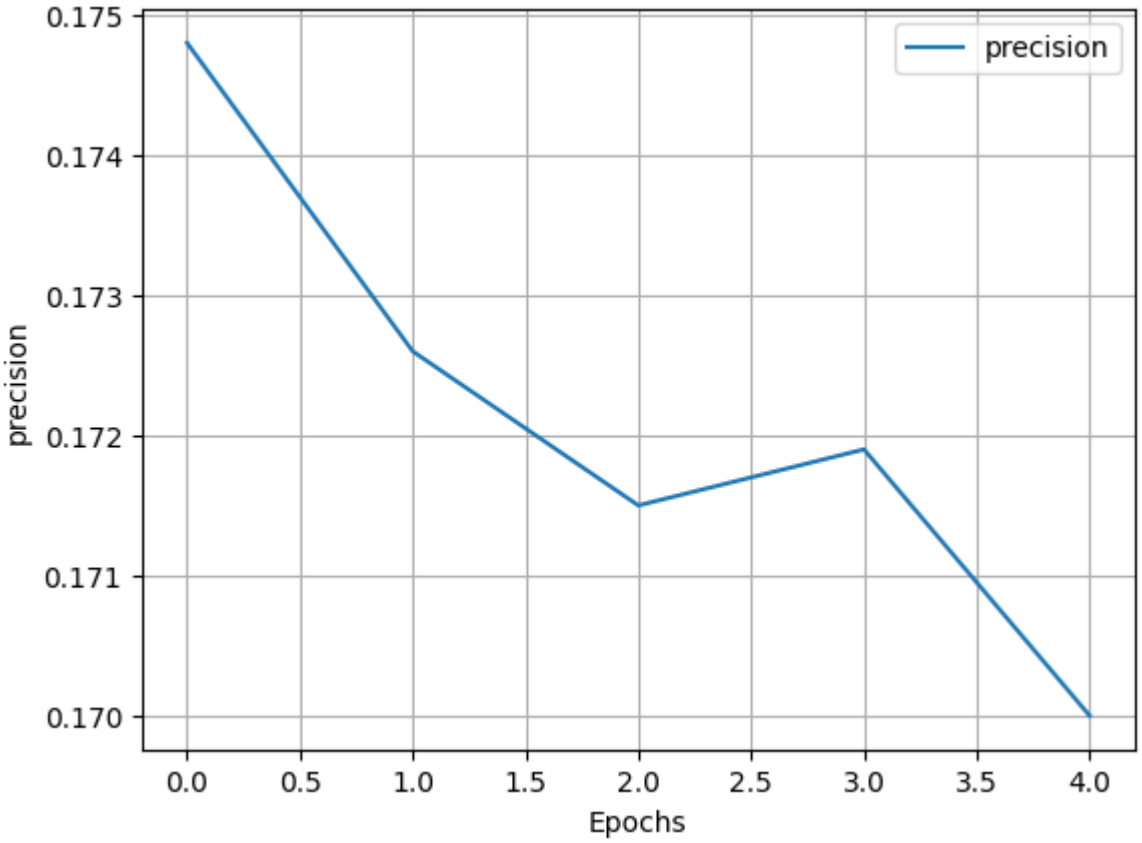

Рисунок 18 - Зависимость precision от эпохи

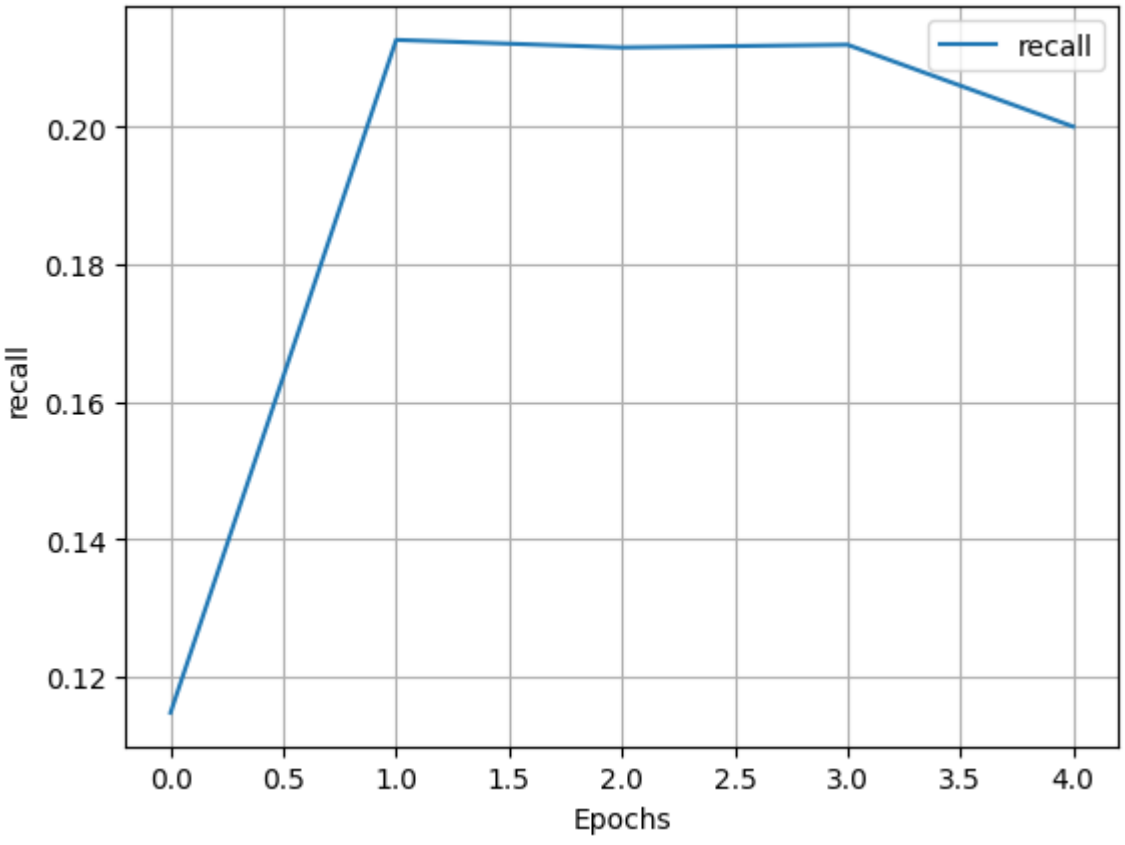

Рисунок 19 - Зависимость recall от эпохи

3.6 Разработка и обучение модели SWIN Transformer

3.6.1 Разработка и обучение нейронной сети

На основе архитектуры SWIN Transformer был разработан программный код на языке python. Полный код представлен в приложении Е. Для компиляции модели были использованы оптимизатор Adam и Категориальная перекрёстная энтропия в качестве функции потерь.

Набор данных для обучения модели был обработан и разделён на обучающую, валидационную и тестовую в соотношении 92%, 4%, 4% соответственно. После этого, обучил модель на 7-ти эпохах.

3.6.2 Оценка результатов работы

После обучения были получены следующие результаты:

Accuracy составила 0.98

Test Accuracy of Crazing: 90% (10/11)
Test Accuracy of Inclusion: 100% (10/10)
Test Accuracy of Patches: 100% (12/12)
Test Accuracy of Pitted: 100% (9/ 9)
Test Accuracy of Rolled: 100% (12/12)
Test Accuracy of Scratches: 100% (10/10)
Test Accuracy of 98% (63/64)

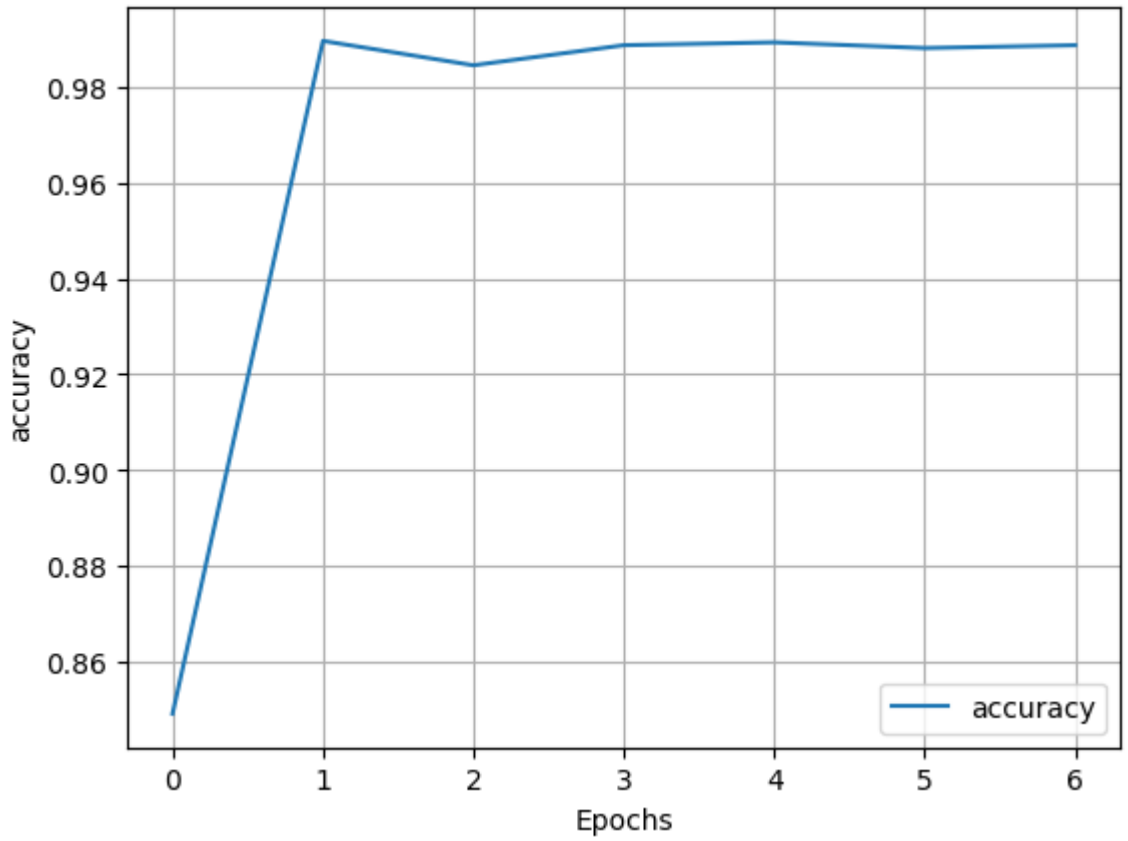

Рисунок 20 - Зависимость accuracy от эпохи

3.7 Разработка и обучение модели ViT

3.7.1 Разработка и обучение нейронной сети

На основе архитектуры ViT был разработан программный код на языке python. Полный код представлен в приложении Ж. Для компиляции модели были использованы оптимизатор Adam и Категориальная перекрёстная энтропия в качестве функции потерь.

Набор данных для обучения модели был обработан и разделён на обучающую, валидационную и тестовую в соотношении 92%, 4%, 4% соответственно. После этого, обучил модель на 20-ти эпохах.

3.7.2 Оценка результатов работы

После обучения были получены следующие результаты:

Accuracy составила 0.21

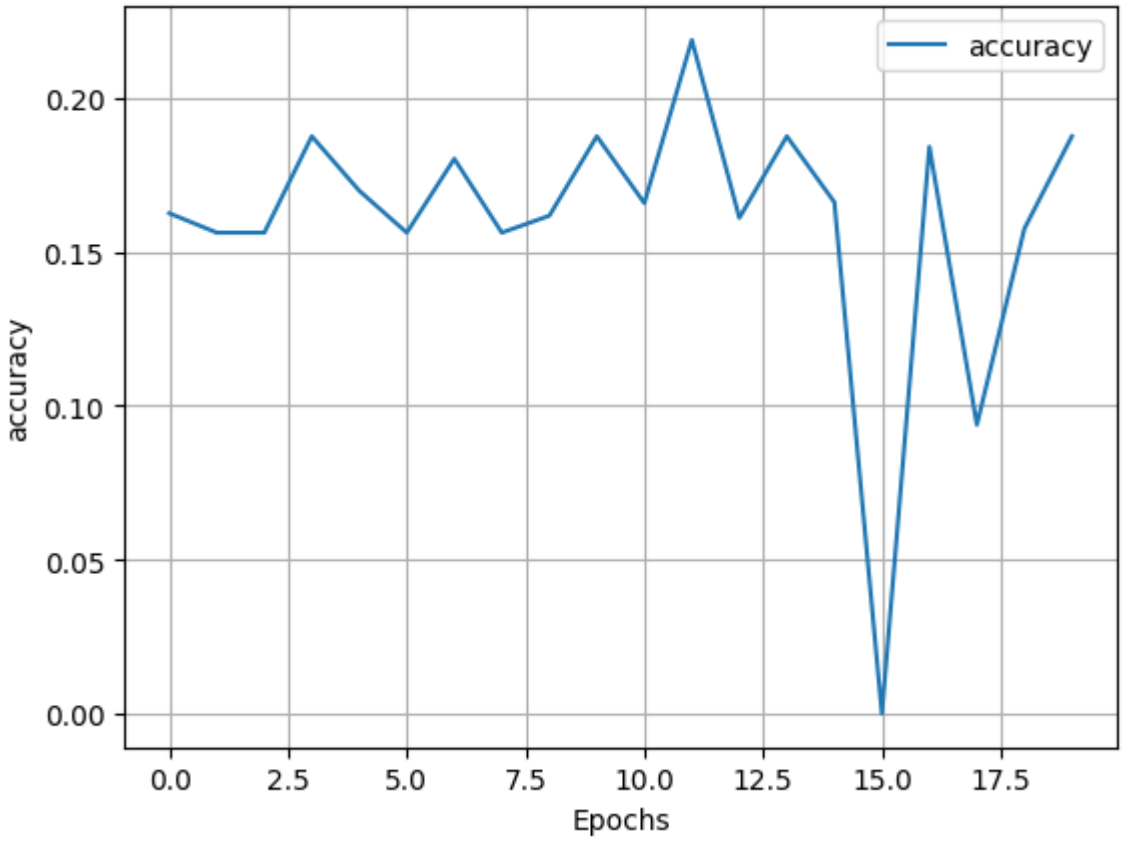

Рисунок 21 - Зависимость accuracy от эпохи

3.8 Сравнение всех моделей

Таблица 3 – сравнение всех моделей

	EfficientNet	ResNet	Sequential	VggNet	SWIN	Vit
Accuracy	0.47	0.9	0.94	0.16	0.98	0.21
Время обучения	Среднее (На одну эпоху около 16 минут)	Быстрое (на одну эпоху около 10 минут)	Быстрое (на одну эпоху около 10 минут)	Долгое (на одну эпоху около 40 минут)	Быстрое (на одну эпоху около 10 минут)	Среднее (на одну эпоху около 14 минут)

3.9 Вывод

Классический архитектура VggNet показал самую худшую точность, при этом имел очень долгое время обучения. Архитектуры EfficientNet и ViT имеют среднее время обучения, но при этом их точность оставляет желать лучшего. Самыми успешными являются современные архитектуры ResNet и SWIN, которые имеют очень короткое время обучения одной эпохи, при этом достигают точности свыше 0.9.

ЗАКЛЮЧЕНИЕ

В данной работе были исследованы различные архитектуры машинного обучения для распознавания поверхностных дефектов листового металлопроката. В ходе работы были выполнены следующие задачи:

- Проведён аналитический обзор алгоритмов обработки изображения.
- Выбран набор данных включающий разнообразные цифровые изображения листового холоднокатаного металлопроката с различными типами поверхностных дефектов.
- Проведён аналитический обзор алгоритмов машинного обучения.
- Были рассмотрены классические алгоритмы машинного обучения, алгоритмы, основанные на искусственной нейронной сети и трансформеры.
- Было реализовано программное обеспечение на языке программирования Python.
- Выполнен анализ результатов. В ходе исследования, теория подтвердила практику.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Хамухин А.В. Актуальные проблемы современной науки. Москва, 2014.
2. Бойко Н.И., Фисенко К.С. Инженерный вестник Дона, 2012, №2.
3. Контроль качества листового металла компьютерным зрением. URL: https://mlsense.nordclan.com/kontrol-kachestva-listovogo-metalla-kompyuternym-zreniem
4. Азин А.В. Метод неразрушающего контроля. Решетневские чтения 2017.
5. Толмачёв И.И. Повышение качества проведения магнитопорошковой дефектоскопии объектов с подповерхностными дефектами. Вестник науки Сибири. 2014.
6. Клюев В.В. Неразрушающий контроль и диагностика. 2005.
7. Фёдоров О.В. Задачи ресурсообеспечения. Москва. 2021.
8. ГОСТ Р 56542-2019. Контроль неразрушающий. Классификация видов и методов. 2019.
9. Герасименко А.А. Международный научный журнал «Символ науки». # 10-2-1 / 2024.
10. http://www.ato.ru/content/nerazrushayushchiy-kontrol
11. Тенишева Е.Д. Сравнительный анализ ультразвуковой и вихретоковой дефектоскопии. Актуальные проблемы авиации и космонавтики. 2019.
12. Moluch [Электронный ресурс]. URL: https://moluch.ru/archive/106/25262/
13. Плетнёв П.М. Вестник СГУПСа. Выпуск 28.
14. Ананьев А.И. Керамический кирпич и его место в строительстве современных зданий. 2013.
15. Бобров А.Л. Основы магнитного неразрушающего контроля.
16. Голкова Р.Д. Особенности акустического метода дефектоскопии металлоизделий.

17. Madhav Moganti. Automatic PCB Inspection Algorithms: A Survey.

18. Yongbing Zhou. Review of vision-based defect detection research and its perspectives for printed circuit board.

19. A. Litvintseva, O. Evstafev. Real-time Steel Surface Defect Recognition Based on CNN.

20. Y. Wang. Knowledge Graphguided Convolutional Neural Network for Surface Defect Recognition.

21. International Conference on Machine Learning, 2019. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks.

22. Kaiming He. Deep Residual Learning for Image Recognition.

23. D. Alves. Detecting Customer Induced Damages in Motherboards with Deep Neural Networks.

24. Chen Xing. A Comprehensive Review of Deep Learning-based PCB Defect Detection.

25. Abu Masyitah. The Performance Analysis of Transfer Learning for Steel Defect Detection by Using Deep Learning.

26. А. В. Сеничев. Сравнение глубокого обучения с традиционными методами компьютерного зрения в задачах идентификации дефектов.

ПРИЛОЖЕНИЕ А

```
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow import keras
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing.image import ImageDataGenerator,
load_img, img_to_array
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Activation,
Flatten, Dense, Dropout


train_dir = "/content/drive/MyDrive/train"
test_dir = "/content/drive/MyDrive/test"
valid_dir = "/content/drive/MyDrive/valid"


train_datagen = ImageDataGenerator(rescale=1./255,
                                   rotation_range=20,
                                   width_shift_range=0.1,
                                   height_shift_range=0.1,
                                   horizontal_flip=True)
test_datagen = ImageDataGenerator(rescale=1./255)


train_generator = train_datagen.flow_from_directory(train_dir,
                                                    target_size=(260, 260),
                                                    batch_size=32,
                                                    class_mode='categorical',
                                                    shuffle=True)
valid_generator = test_datagen.flow_from_directory(valid_dir,
                                                    target_size=(260, 260),
                                                    batch_size=32,
                                                    class_mode='categorical',
                                                    shuffle=False)

test_generator = test_datagen.flow_from_directory(test_dir,
                                                    target_size=(260, 260),
                                                    batch_size=32,
                                                    class_mode='categorical',
                                                    shuffle=False)



from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Activation,
Flatten, Dense, Dropout
from tensorflow.keras.applications.efficientnet import EfficientNetB3
```

```
from keras.layers import Dense


base_model = EfficientNetB3(include_top = False , weights = 'imagenet' ,
                                                              input_shape
= (260,260,3), pooling= 'max')

efficientnet_model = Sequential()
efficientnet_model.add(base_model)
efficientnet_model.add(Dense(units = 256, activation='relu'))
efficientnet_model.add(Dense(units = 6, activation = 'relu'))
efficientnet_model.summary()



efficientnet_model.compile(optimizer="adam",loss="categorical_crossentropy
",metrics=["accuracy", "F1Score", "precision", "recall"])



efficient_history = efficientnet_model.fit(train_generator,
                    epochs=20,
                    batch_size=32,
                    validation_data=valid_generator)
```

ПРИЛОЖЕНИЕ Б

```
train_datagen = ImageDataGenerator(rescale=1./255,
                                rotation_range=20,
                                width_shift_range=0.1,
                                height_shift_range=0.1,
                                horizontal_flip=True)
test_datagen = ImageDataGenerator(rescale=1./255)


train_generator = train_datagen.flow_from_directory(train_dir,
                                                    target_size=(200, 200),
                                                    batch_size=32,
                                                    class_mode='categorical',
                                                    shuffle=True)
valid_generator = test_datagen.flow_from_directory(valid_dir,
                                                    target_size=(200, 200),
                                                    batch_size=32,
                                                    class_mode='categorical',
                                                    shuffle=False)

test_generator = test_datagen.flow_from_directory(test_dir,
```

```
                                                target_size=(200, 200),
                                                batch_size=32,
                                                class_mode='categorical',
                                                shuffle=False)


from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Activation,
Flatten, Dense, Dropout
from tensorflow.keras.applications.resnet import ResNet50
from keras.layers import Dense


base_model = ResNet50(include_top = False , weights = 'imagenet' ,
                                                                    input_shape
= (200,200,3), pooling= 'max')

resnet_model = Sequential()
resnet_model.add(base_model)
resnet_model.add(Dense(units = 256, activation='relu'))
resnet_model.add(Dense(units = 6, activation = 'relu'))
resnet_model.summary()


resnet_model.compile(optimizer="adam",loss="categorical_crossentropy",metr
ics=["accuracy", "F1Score", "precision", "recall"])


resnet_history = resnet_model.fit(train_generator,
                    epochs=10,
                    batch_size=32,
                    validation_data=valid_generator)
```

ПРИЛОЖЕНИЕ В

```
Model: "sequential_8"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 199, 199, 32)      416

 max_pooling2d (MaxPooling2  (None, 99, 99, 32)        0
 D)

 conv2d_1 (Conv2D)           (None, 98, 98, 64)        8256

 max_pooling2d_1 (MaxPoolin  (None, 49, 49, 64)        0
 g2D)

 conv2d_2 (Conv2D)           (None, 48, 48, 128)       32896

 max_pooling2d_2 (MaxPoolin  (None, 24, 24, 128)       0
 g2D)

 flatten (Flatten)           (None, 73728)             0

 dense_12 (Dense)            (None, 256)               18874624

 dropout (Dropout)           (None, 256)               0

 dense_13 (Dense)            (None, 6)                 1542

=================================================================
Total params: 18917734 (72.17 MB)
Trainable params: 18917734 (72.17 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

ПРИЛОЖЕНИЕ Г

```
train_datagen = ImageDataGenerator(rescale=1./255,
                                   rotation_range=20,
                                   width_shift_range=0.1,
                                   height_shift_range=0.1,
                                   horizontal_flip=True)
test_datagen = ImageDataGenerator(rescale=1./255)


train_generator = train_datagen.flow_from_directory(train_dir,
                                                    target_size=(200, 200),
                                                    batch_size=32,
                                                    class_mode='categorical',
                                                    shuffle=True)
valid_generator = test_datagen.flow_from_directory(valid_dir,
```

```
                                                target_size=(200, 200),
                                                batch_size=32,
                                                class_mode='categorical',
                                                shuffle=False)

test_generator = test_datagen.flow_from_directory(test_dir,
                                                target_size=(200, 200),
                                                batch_size=32,
                                                class_mode='categorical',
                                                shuffle=False)


sequential_model = Sequential([ Conv2D(32, (2, 2), activation='relu',
input_shape=(200, 200, 3)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (2, 2), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(128, (2, 2), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(256, activation='relu'),
    Dropout(0.2),
    Dense(6 ,activation='softmax')])


sequential_model.compile(optimizer="adam",loss="categorical_crossentropy",
metrics=["accuracy", "F1Score", "precision", "recall"])


sequential_history = sequential_model.fit(train_generator,
                    epochs=20,
                    batch_size=32,
                    validation_data=valid_generator)
```

ПРИЛОЖЕНИЕ Д

```
train_datagen = ImageDataGenerator(rescale=1./255,
                                rotation_range=20,
                                width_shift_range=0.1,
                                height_shift_range=0.1,
                                horizontal_flip=True)
test_datagen = ImageDataGenerator(rescale=1./255)


train_generator = train_datagen.flow_from_directory(train_dir,
                                                target_size=(200, 200),
                                                batch_size=32,
                                                class_mode='categorical',
                                                shuffle=True)
```

```
valid_generator = test_datagen.flow_from_directory(valid_dir,
                                                   target_size=(200, 200),
                                                   batch_size=32,
                                                   class_mode='categorical',
                                                   shuffle=False)

test_generator = test_datagen.flow_from_directory(test_dir,
                                                   target_size=(200, 200),
                                                   batch_size=32,
                                                   class_mode='categorical',
                                                   shuffle=False)


from tf_keras.applications.vgg16 import VGG16
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Activation,
Flatten, Dense, Dropout

base_model = VGG16(include_top = False , weights = 'imagenet', input_shape
= (200,200,3), pooling= 'max')

vgg_model = Sequential()
vgg_model.add(base_model)
vgg_model.add(Dense(units = 128, activation='relu'))
vgg_model.add(Dense(units = 6, activation = 'relu'))

vgg_model.build(input_shape=(None, 200, 200, 3))



vgg_model.compile(optimizer="adam",loss="categorical_crossentropy",metrics
=["accuracy"])


vgg_history = vgg_model.fit(train_generator,
                    epochs=5,
                    batch_size=32,
                    validation_data=valid_generator)
```

ПРИЛОЖЕНИЕ Е

```
import numpy as np
import pandas as pd
import os
import torch
import torchvision
from torchvision import datasets
from torchvision import transforms as T
from torch import nn, optim
```

```
from torch.nn import functional as F
from torch.utils.data import DataLoader, sampler, random_split
from torchvision import models

from tensorflow.keras.preprocessing.image import ImageDataGenerator,
load_img, img_to_array

import timm
from timm.loss import LabelSmoothingCrossEntropy

import sys
from tqdm import tqdm
import time
import copy



def get_classes(data_dir):
    all_data = datasets.ImageFolder(data_dir)
    return all_data.classes



def get_data_loaders(data_dir, batch_size, train = False, val = False):
    if train:
        #train
        transform = T.Compose([
            T.RandomHorizontalFlip(),
            T.RandomVerticalFlip(),
            T.RandomApply(torch.nn.ModuleList([T.ColorJitter()]), p=0.25),
            T.Resize(256),
            T.CenterCrop(224),
            T.ToTensor(),
            T.Normalize(timm.data.IMAGENET_DEFAULT_MEAN,
timm.data.IMAGENET_DEFAULT_STD),
            T.RandomErasing(p=0.1, value='random')
        ])
        train_data = datasets.ImageFolder(os.path.join(data_dir,
"train/"), transform = transform)
        train_loader = DataLoader(train_data, batch_size=batch_size,
shuffle=True, num_workers=4)
        return train_loader, len(train_data)
    elif val:
        # val/test
        transform = T.Compose([
            T.Resize(256),
            T.CenterCrop(224),
            T.ToTensor(),
            T.Normalize(timm.data.IMAGENET_DEFAULT_MEAN,
timm.data.IMAGENET_DEFAULT_STD),
        ])
```

```
        val_data = datasets.ImageFolder(os.path.join(data_dir, "valid/"),
transform=transform)

        val_loader = DataLoader(val_data, batch_size=batch_size,
shuffle=True, num_workers=4)

        return val_loader, len(val_data)
    else:
        transform = T.Compose([
            T.Resize(256),
            T.CenterCrop(224),
            T.ToTensor(),
            T.Normalize(timm.data.IMAGENET_DEFAULT_MEAN,
timm.data.IMAGENET_DEFAULT_STD),
        ])
        test_data = datasets.ImageFolder(os.path.join(data_dir, "test/"),
transform=transform)
        test_loader = DataLoader(test_data, batch_size=batch_size,
shuffle=True, num_workers=4)
        return test_loader, len(test_data)



(train_loader, train_data_len) = get_data_loaders(train_dir, 128,
train=True)
(val_loader, valid_data_len) = get_data_loaders(valid_dir, 32, val=True)
(test_loader, test_data_len) = get_data_loaders(valid_dir, 32)



dataloaders = {
    "train": train_loader,
    "val": val_loader
}
dataset_sizes = {
    "train": train_data_len,
    "val": valid_data_len
}



HUB_URL = "SharanSMenon/swin-transformer-hub:main"
MODEL_NAME = "swin_tiny_patch4_window7_224"
model = torch.hub.load(HUB_URL, MODEL_NAME, pretrained=True)



for param in model.parameters():
    param.requires_grad = False

n_inputs = model.head.in_features
```

```
model.head = nn.Sequential(
    nn.Linear(n_inputs, 512),
    nn.ReLU(),
    nn.Dropout(0.3),
    nn.Linear(512, len(class_names))
)
model = model.to(device)
print(model.head)



criterion = LabelSmoothingCrossEntropy()
criterion = criterion.to(device)
optimizer = optim.AdamW(model.head.parameters(), lr=0.001)



exp_lr_scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=3,
gamma=0.97)



def train_model(model, criterion, optimizer, scheduler, num_epochs=10):
    since = time.time()
    best_model_wts = copy.deepcopy(model.state_dict())
    best_acc = 0.0

    for epoch in range(num_epochs):
        print(f'Epoch {epoch}/{num_epochs - 1}')
        print("-"*10)

        for phase in ['train', 'val']:
            if phase == 'train':
                model.train()
            else:
                model.eval()
            running_loss = 0.0
            running_corrects = 0.0

            for inputs, labels in tqdm(dataloaders[phase]):
                inputs = inputs.to(device)
                labels = labels.to(device)

                optimizer.zero_grad()

                with torch.set_grad_enabled(phase == 'train'):
                    outputs = model(inputs)
                    _, preds = torch.max(outputs, 1)
                    loss = criterion(outputs, labels)

                    if phase == 'train':
```

```
                        loss.backward()
                        optimizer.step()
                running_loss += loss.item() * inputs.size(0)
                running_corrects += torch.sum(preds == labels.data)

            if phase == 'train':
                scheduler.step()

            epoch_loss = running_loss / dataset_sizes[phase]
            epoch_acc =  running_corrects.double() / dataset_sizes[phase]

            print("{} Loss: {:.4f} Acc: {:.4f}".format(phase, epoch_loss,
epoch_acc))

            if phase == 'val' and epoch_acc > best_acc:
                best_acc = epoch_acc
                best_model_wts = copy.deepcopy(model.state_dict())
        print()
    time_elapsed = time.time() - since
    print('Training complete in {:.0f}m {:.0f}s'.format(time_elapsed //
60, time_elapsed % 60))
    print("Best Val Acc: {:.4f}".format(best_acc))

    model.load_state_dict(best_model_wts)
    return model



model_ft = train_model(model, criterion, optimizer, exp_lr_scheduler,
num_epochs=7)



test_loss = 0.0
class_correct = list(0 for i in range(len(classes)))
class_total = list(0 for i in range(len(classes)))
model_ft.eval()

for data, target in tqdm(test_loader):
    data, target = data.to(device), target.to(device)
    with torch.no_grad():
        output = model_ft(data)
        loss = criterion(output, target)
    test_loss = loss.item() * data.size(0)
    _, pred = torch.max(output, 1)
    correct_tensor = pred.eq(target.data.view_as(pred))
    correct = np.squeeze(correct_tensor.cpu().numpy())
    if len(target) == 32:
        for i in range(32):
            label = target.data[i]
            class_correct[label] += correct[i].item()
```

```
            class_total[label] += 1

test_loss = test_loss / test_data_len
print('Test Loss: {:.4f}'.format(test_loss))
for i in range(len(classes)):
    if class_total[i] > 0:
        print("Test Accuracy of %5s: %2d%% (%2d/%2d)" % (
            classes[i], 100*class_correct[i]/class_total[i],
np.sum(class_correct[i]), np.sum(class_total[i])
        ))
    else:
        print("Test accuracy of %5s: NA" % (classes[i]))
print("Test Accuracy of %2d%% (%2d/%2d)" % (
            100*np.sum(class_correct)/np.sum(class_total),
np.sum(class_correct), np.sum(class_total)
        ))



print(model_ft)
```

ПРИЛОЖЕНИЕ Ж

```
import numpy as np
import pandas as pd
import tensorflow as tf
import tensorflow.keras.layers as L
import glob, random, os, warnings
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns
import glob
from tensorflow.keras.preprocessing.image import ImageDataGenerator,
load_img, img_to_array

print('TensorFlow Version ' + tf.__version__)

def seed_everything(seed = 0):
    random.seed(seed)
    np.random.seed(seed)
    tf.random.set_seed(seed)
    os.environ['PYTHONHASHSEED'] = str(seed)
    os.environ['TF_DETERMINISTIC_OPS'] = '1'

seed_everything()
warnings.filterwarnings('ignore')
```

```
image_size = 200
batch_size = 32
n_classes = 6

train_dir = "/content/drive/MyDrive/train"
test_dir = "/content/drive/MyDrive/test"
valid_dir = "/content/drive/MyDrive/valid"


test_images = glob.glob(test_dir + '/*.jpg')
df_test = pd.DataFrame(test_images, columns = ['image_path'])



def data_augment(image):
    p_spatial = tf.random.uniform([], 0, 1.0, dtype = tf.float32)
    p_rotate = tf.random.uniform([], 0, 1.0, dtype = tf.float32)

    image = tf.image.random_flip_left_right(image)
    image = tf.image.random_flip_up_down(image)

    if p_spatial > .75:
        image = tf.image.transpose(image)


    if p_rotate > .75:
        image = tf.image.rot90(image, k = 3)
    elif p_rotate > .5:
        image = tf.image.rot90(image, k = 2)
    elif p_rotate > .25:
        image = tf.image.rot90(image, k = 1)
    return image



train_datagen = ImageDataGenerator(rescale=1./255,
                                   rotation_range=20,
                                   width_shift_range=0.1,
                                   height_shift_range=0.1,
                                   horizontal_flip=True)
test_datagen = ImageDataGenerator(rescale=1./255)

train_gen = train_datagen.flow_from_directory(train_dir,
                                              target_size=(200, 200),
                                              batch_size=32,
                                              class_mode='categorical',
                                              shuffle=True)
valid_gen = test_datagen.flow_from_directory(valid_dir,
                                              target_size=(200, 200),
                                              batch_size=32,
```

```
                                                class_mode='categorical',
                                                shuffle=False)

test_gen = test_datagen.flow_from_directory(test_dir,
                                                target_size=(200, 200),
                                                batch_size=32,
                                                class_mode='categorical',
                                                shuffle=False)



learning_rate = 0.001
weight_decay = 0.0001
num_epochs = 1

patch_size = 7  num_patches = (image_size // patch_size) ** 2
projection_dim = 64
num_heads = 4
transformer_units = [
    projection_dim * 2,
    projection_dim,
]
transformer_layers = 8
mlp_head_units = [56, 28]



def mlp(x, hidden_units, dropout_rate):
    for units in hidden_units:
        x = L.Dense(units, activation = tf.nn.gelu)(x)
        x = L.Dropout(dropout_rate)(x)
    return x



class Patches(L.Layer):
    def __init__(self, patch_size):
        super(Patches, self).__init__()
        self.patch_size = patch_size

    def call(self, images):
        batch_size = tf.shape(images)[0]
        patches = tf.image.extract_patches(
            images = images,
            sizes = [1, self.patch_size, self.patch_size, 1],
            strides = [1, self.patch_size, self.patch_size, 1],
            rates = [1, 1, 1, 1],
            padding = 'VALID',
        )
        patch_dims = patches.shape[-1]
        patches = tf.reshape(patches, [batch_size, -1, patch_dims])
```

```
        return patches


class PatchEncoder(L.Layer):
    def __init__(self, num_patches, projection_dim):
        super(PatchEncoder, self).__init__()
        self.num_patches = num_patches
        self.projection = L.Dense(units = projection_dim)
        self.position_embedding = L.Embedding(
            input_dim = num_patches, output_dim = projection_dim
        )

    def call(self, patch):
        positions = tf.range(start = 0, limit = self.num_patches, delta =
1)
        encoded = self.projection(patch) +
self.position_embedding(positions)
        return encoded



def vision_transformer():
    inputs = L.Input(shape = (image_size, image_size, 3))


    patches = Patches(patch_size)(inputs)


    encoded_patches = PatchEncoder(num_patches, projection_dim)(patches)


    for _ in range(transformer_layers):


        x1 = L.LayerNormalization(epsilon = 1e-6)(encoded_patches)


        attention_output = L.MultiHeadAttention(
            num_heads = num_heads, key_dim = projection_dim, dropout = 0.1
        )(x1, x1)


        x2 = L.Add()([attention_output, encoded_patches])


        x3 = L.LayerNormalization(epsilon = 1e-6)(x2)


        x3 = mlp(x3, hidden_units = transformer_units, dropout_rate = 0.1)
```

```
        encoded_patches = L.Add()([x3, x2])


    representation = L.LayerNormalization(epsilon = 1e-6)(encoded_patches)
    representation = L.Flatten()(representation)
    representation = L.Dropout(0.5)(representation)


    features = mlp(representation, hidden_units = mlp_head_units,
dropout_rate = 0.5)


    logits = L.Dense(n_classes)(features)


    model = tf.keras.Model(inputs = inputs, outputs = logits)

    return model



decay_steps = train_gen.n // train_gen.batch_size
initial_learning_rate = learning_rate

lr_decayed_fn =
tf.keras.optimizers.schedules.CosineDecay(initial_learning_rate,
decay_steps)

lr_scheduler = tf.keras.callbacks.LearningRateScheduler(lr_decayed_fn)



optimizer = tf.keras.optimizers.Adam(learning_rate = learning_rate)

model = vision_transformer()

model.compile(optimizer = optimizer,
              loss =
tf.keras.losses.CategoricalCrossentropy(label_smoothing = 0.1),
              metrics = ['accuracy'])


STEP_SIZE_TRAIN = train_gen.n // train_gen.batch_size
STEP_SIZE_VALID = valid_gen.n // valid_gen.batch_size

earlystopping = tf.keras.callbacks.EarlyStopping(monitor = 'val_accuracy',
                                                 min_delta = 1e-4,
                                                 patience = 5,
                                                 mode = 'max',
```

```
                                                    restore_best_weights =
True,
                                                    verbose = 1)


callbacks = [earlystopping, lr_scheduler]

model.fit(x = train_gen,
          steps_per_epoch = STEP_SIZE_TRAIN,
          validation_data = valid_gen,
          validation_steps = STEP_SIZE_VALID,
          epochs = 20)
```