**ОГЛАВЛЕНИЕ**

[ВВЕДЕНИЕ 2](#_Toc192805827)

[1. ТЕХНОЛОГИИ СВЁРТОЧНЫХ НЕЙРОННЫХ СЕТЕЙ 5](#_Toc192805828)

[1.1. Понятие свёрточных нейронных сетей 5](#_Toc192805829)

[1.2. Трансферное обучение 8](#_Toc192805830)

[1.2.1. Архитектура VGG 9](#_Toc192805831)

[1.2.2. Архитектура ResNet 10](#_Toc192805832)

[1.2.3 Архитектура MobileNet 12](#_Toc192805833)

[1.2.4. Архитектура EfficientNet 14](#_Toc192805834)

[2. ОБУЧЕНИЕ НЕЙРОННЫХ СЕТЕЙ 17](#_Toc192805835)

[2.1. Набор данных 17](#_Toc192805836)

[2.2. Реализация нейронных сетей 20](#_Toc192805837)

[3. АНАЛИЗ РЕЗУЛЬТАТОВ И РАЗРАБОТКА ИНТЕРФЕЙСА 22](#_Toc192805838)

[3.1. Анализ результатов 22](#_Toc192805839)

[СПИСОК ЛИТЕРАТУРЫ 24](#_Toc192805840)

[ПРИЛОЖЕНИЕ А 25](#_Toc192805841)

[ПРИЛОЖЕНИЕ Б 26](#_Toc192805842)

ВВЕДЕНИЕ

Современная авиация играет ключевую роль в глобальной транспортной системе, обеспечивая быстрое перемещение людей и грузов на большие расстояния. С развитием технологий разнообразие воздушных судов непрерывно растёт: от пассажирских лайнеров и грузовых самолётов до специализированных моделей для военных и исследовательских задач. Точная идентификация типа воздушного судна по внешним признакам критически важна для решения таких задач, как управление воздушным движением, мониторинг безопасности, логистическое планирование и анализ спутниковых данных.

В последние годы значительно возрос интерес к использованию методов компьютерного зрения для автоматического анализа аэрокосмических изображений. Такие технологии применяются в гражданской и военной сферах, включая мониторинг аэропортов, контроль воздушного движения, автоматическое обнаружение объектов на спутниковых снимках и идентификацию моделей воздушных судов. Современные алгоритмы машинного обучения, такие как свёрточные нейронные сети, демонстрируют высокую эффективность [1] в задачах классификации изображений, но требуют дальнейшего совершенствования при применении к сложным сценам с высоким уровнем шума и разнообразием классов объектов.

Актуальность разработки надёжных методов машинного обучения для классификации воздушных судов обусловлена растущим объёмом спутниковых данных и необходимостью их оперативного анализа. Основная сложность заключается в вариативности ракурсов съёмки, влиянии погодных условий, наличии помех и высокой степени схожести между различными моделями воздушных судов. Совершенствование методов автоматической классификации позволит повысить точность идентификации и сократить нагрузку на специалистов, работающих с аэрокосмическими данными.

Целью данной работы является разработка системы классификации типов воздушных судов по спутниковым изображениям на основе свёрточной нейронной сети.

Для реализации поставленной цели были решены следующие задачи:

1. Определить наиболее подходящие модели свёрточных нейронных сетей для реализации трансферного обучения;
2. Определить метрики для объективного сравнения моделей;
3. Обучить модели на наборе данных спутниковых изображений воздушных судов;
4. Определить наиболее оптимальную модель для заданной цели;
5. Реализовать пользовательский интерфейс.

Объектом исследования является процесс автоматической классификации воздушных судов по изображениям; предметом исследования является применение свёрточных нейронных сетей для решения задачи классификации типов воздушных судов.

Методологической основой работы является использование методов глубокого обучения, в частности, свёрточных нейронных сетей и обучения с переносом для обработки и классификации изображений. В качестве инструментов использовались язык программирования Python, в частности библиотеки TensorFlow и Keras для реализации моделей машинного обучения, а также набор данных спутниковых снимков воздушных судов. Для разработки пользовательского интерфейса применялась библиотека PythonQt, которая позволила создать удобное и эффективное графическое представление результатов классификации.

Практическая значимость работы заключается в разработке системы, способной автоматически идентифицировать типы воздушных судов, что может быть применимо в задачах мониторинга, обороны и авиационной аналитики.

Структура работы включает введение, теоретический обзор современных методов классификации изображений с применением глубокого обучения, описание используемых моделей и методологии экспериментов, результаты тестирования разработанной системы, анализ их эффективности и заключение.

1. ТЕХНОЛОГИИ СВЁРТОЧНЫХ НЕЙРОННЫХ СЕТЕЙ

1.1. Понятие свёрточных нейронных сетей

Свёрточные нейронные сети (далее - СНС) являются классом алгоритмов машинного обучения, в основе которых лежит способность автоматически извлекать признаки из входных данных, что особенно эффективно при обработке изображений и видео. Это достигается благодаря применению свёрточных слоев, которые фильтруют информацию и сохраняют только наиболее значимые признаки [2].

Несмотря на непрерывную эволюцию СНС, набирающую все большие темпы за последнее десятилетие, основные части алгоритма остаются неизменными. Первая модель свёрточной нейронной сети – LeNet-5, разработанная в 1998 году для задачи распознавания рукописных цифр и букв с помощью классификации [3], использует в своей основе те же (но модифицированные) основные компоненты, как и выпущенная в 2021 году модель СНС EfficientNetV2-L с более чем 120 миллионами параметров [4].

Архитектуру простейшей СНС можно разбить на 2 основные части, связанные друг с другом и идущие последовательно [3]:

1) Модуль извлечения признаков, состоящий из набора последовательных слоёв свёртки, слоёв активации и слоёв подвыборки.

2) Обучаемый классификационный модуль, представляющий из себя полносвязную нейронную сеть.

Свёрточные слои — основной компонент СНС, обрабатывающий входные данные с помощью фильтров (ядер свёртки). Ядра свертки - матрицы весов небольшого размера, как правило, 3x3. Свёртка позволяет сети выявлять локальные признаки, такие как края, текстуры и другие элементы изображения. Фильтр перемещается по входному признаку, например, градиенту яркости пикселей изображения, и вычисляет свёртку, то есть скалярное произведение значений входных пикселей и весов ядра. В результате получается карта признаков, содержащая информацию о выявленных структурах. Во время обучения нейронной сети градиентный спуск изменяет веса фильтров, чтобы сеть лучше выделяла значимые признаки [2]. На рисунке 1 представлено визуальное представление работы ядра слоя свёртки.

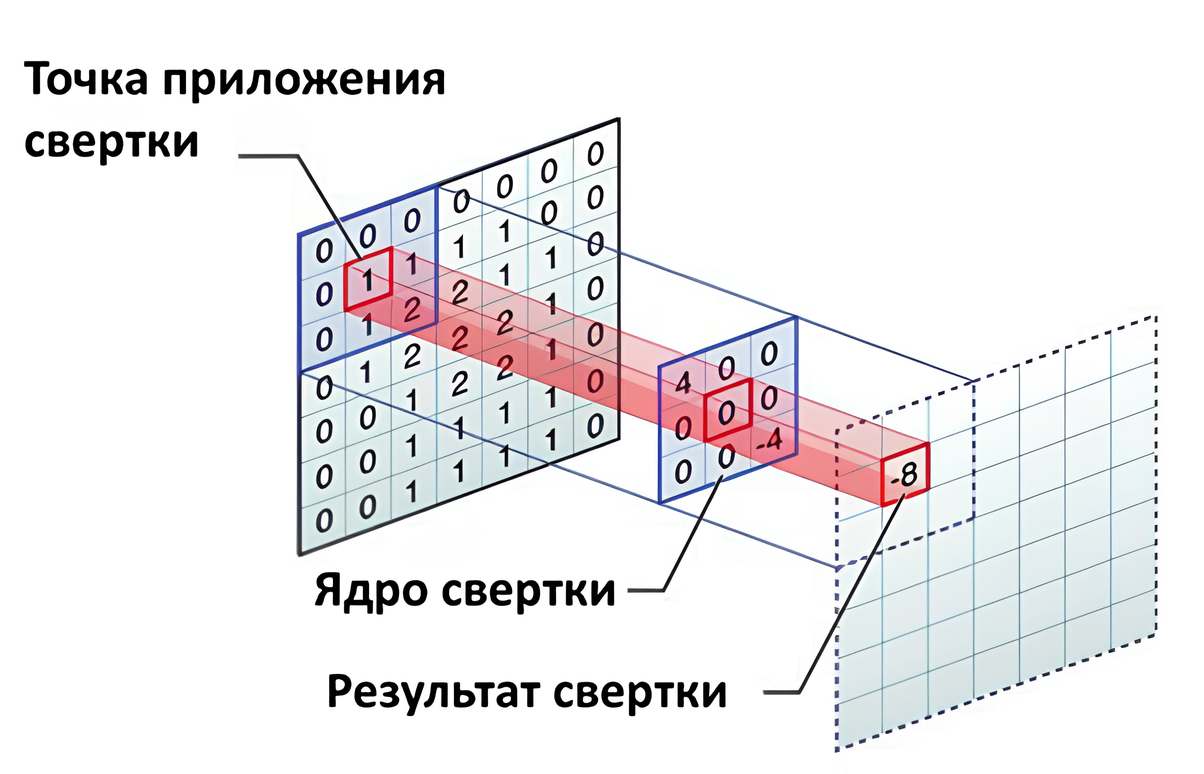


Рисунок 1 – Визуальное представление работы ядра свертки

После применения свёрточных слоёв в большинстве случаев применяется слой активации для введения нелинейности в модель. Это позволяет сети моделировать сложные зависимости между признаками и улучшает её способность к обобщению. Одной из наиболее распространённых функций активации является ReLU (Rectified Linear Unit), которая заменяет все отрицательные значения на ноль, сохраняя положительные значения без изменений [5]. Математическое представление функции активации ReLu:

Другие популярные функции активации включают сигмоидную функцию и гиперболический тангенс (tanh), каждая из которых имеет свои особенности и области применения [5].

Слои подвыборки, в свою очередь, позволяют снизить пространственное разрешение данных, уменьшая размерность и вычислительные затраты, а также способствуют инвариантности к небольшим изменениям в изображении. Существуют несколько разновидностей слоёв подвыборки, однако основные из них: максимальная подвыборка (max pooling) и средняя подвыборка (average pooling), где используется максимальное и среднее значение признаков соответственно [6]. На рисунке 2 представлено визуальное представление работы слоев максимальной и средней подвыборки.

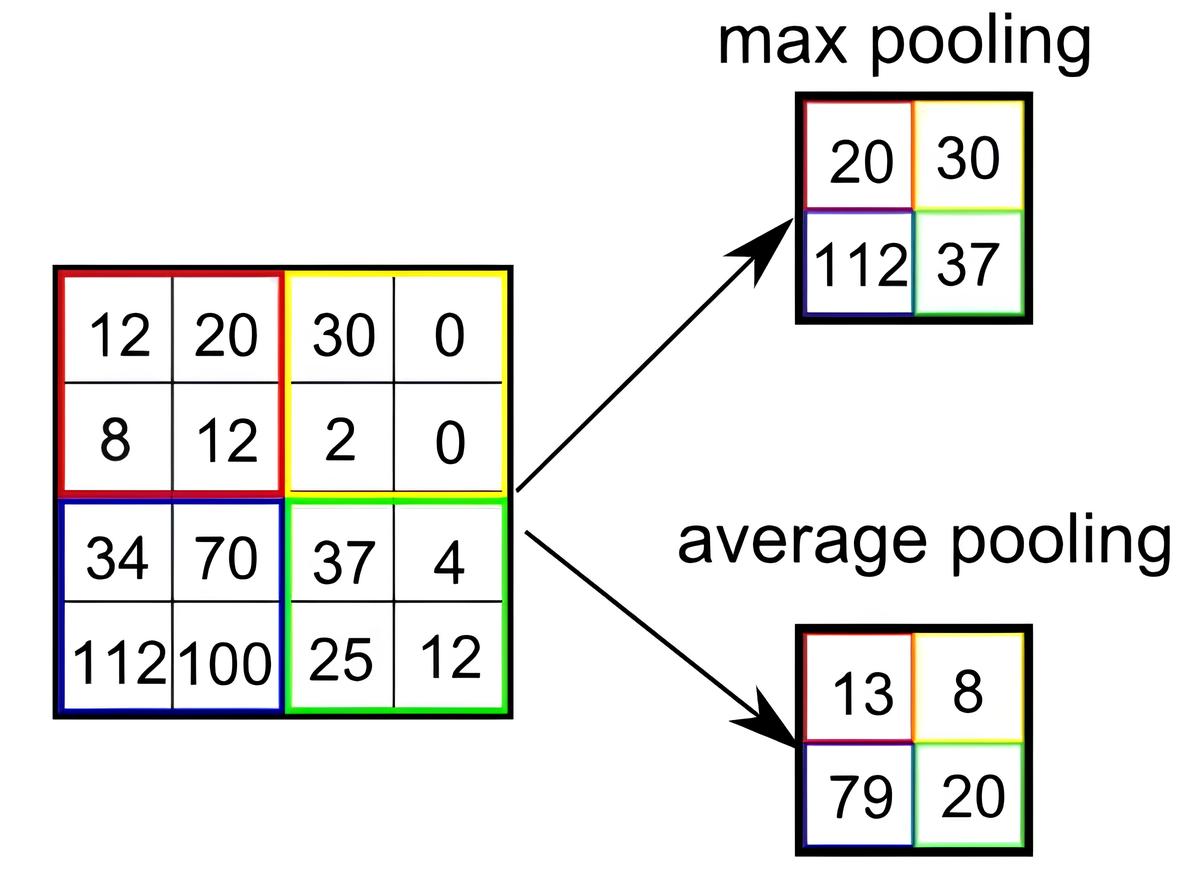


Рисунок 2 – Визуальное представление работы слоев максимальной и средней подвыборки

После выделения основных признаков модуль классификации выполняет интеграцию этих признаков для принятия окончательного решения о принадлежности объекта к определённому классу. Полносвязная нейронная сеть преобразует входной вектор выделенных признаков в выходной вектор вероятностей принадлежности классифицируемого объекта к определённому классу [1].

В задачах классификации на последнем слое часто используется функция активации softmax, которая нормализует выходные значения в диапазон от 0 до 1, представляя собой вероятности принадлежности объекта к определенному классу. Математическое представление функции softmax:

,

где *z* – выходной вектор полносвязного слоя;

*zi –* выходное значение нейрона *i* полносвязного слоя;

*N* – общее количество классов [7].

На рисунке 3 представлена стандартная архитектура простейшей СНС, где «скрытые слои» - модуль выделения признаков, а «классификация» - полносвязная нейронная сеть – классификатор.

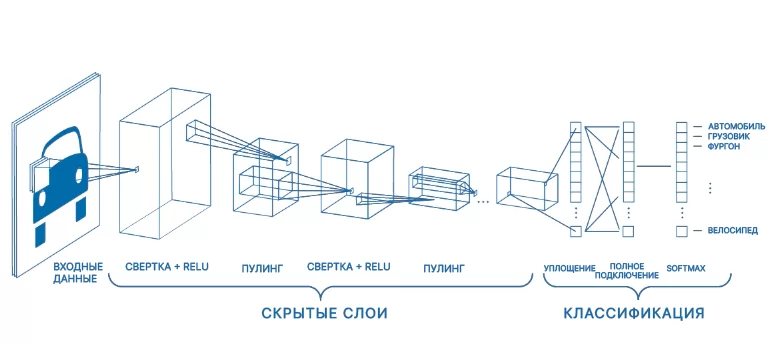


Рисунок 3 – Стандартная архитектура простейшей СНС

1.2. Трансферное обучение

Современные СНС обладают высокой эффективностью в задачах классификации изображений, однако их обучение «с нуля», а именно обучение как модуля извлечения признаков, так и классификационного модуля, требует значительных вычислительных ресурсов и больших объемов данных. Например, обучение модели AlexNet, которая произвела революцию в области машинного обучения в 2012 году, потребовало использования нескольких графических процессоров и заняло значительное время [8].

Для преодоления этих ограничений применяется метод трансферного обучения. Этот подход позволяет использовать веса, полученные моделью при решении одной задачи, для ускорения и улучшения обучения в другой, схожей задаче [9]. В частности, модели, обученные на обширном наборе данных ImageNet Large Scale Visual Recognition Challenge (далее - ILSVRC), содержащем 1,461,406 размеченных изображений [10], широко используются в качестве основы для решения специализированных задач.

В данной работе применяется метод трансферного обучения, при котором параметры базовой модели, предварительно обученной на наборе данных ILSVRC, фиксируются и не обновляются в ходе дальнейшего обучения. Адаптация модели осуществляется исключительно на уровне классификационного слоя, использующего заранее извлечённые признаки.

В следующих разделах рассматриваются архитектуры базовых моделей, к которым была применена методика трансферного обучения для последующего выбора наиболее оптимальной модели для решения задачи классификации воздушных средств на спутниковых изображениях.

1.2.1. Архитектура VGG

Архитектура VGG (Visual Geometry Group), представленная в 2014 году характеризуется последовательным увеличением глубины сети за счёт использования небольших свёрточных фильтров размером 3×3, что позволило значительно улучшить качество классификации изображений по сравнению с более ранними моделями, такими как AlexNet [11].

Модель была представлена в пяти конфигурациях, различающихся глубиной и структурой свёрточных слоёв. Среди них наибольшее распространение получили VGG-16 и VGG-19, содержащие 16 и 19 весовых слоёв соответственно. Эти версии активно используются в задачах компьютерного зрения, например для распознавания промышленных дефектов или для классификации видов растений, благодаря способности извлекать сложные иерархические признаки из изображений [12,13]. На рисунке 4 представлено визуальное представление архитектуры модели VGG-16.

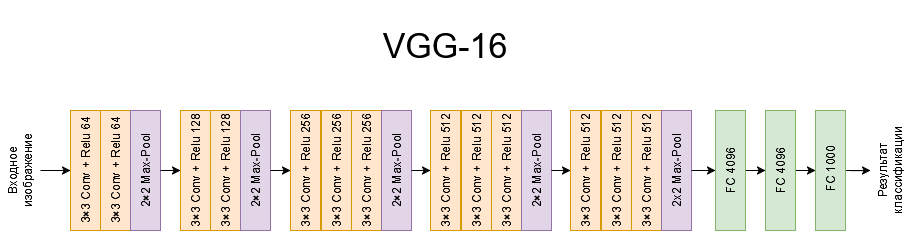


Рисунок 4 - Визуальное представление архитектуры модели VGG-16

1.2.2. Архитектура ResNet

Архитектура Residual Network (ResNet) была разработана в 2015 году группой исследователей под руководством Кайминга Хэ с целью решения проблемы затухания градиентов, возникающей при обучении глубоких нейронных сетей. Затухание градиентов приводит к тому, что градиенты ошибки становятся чрезвычайно малыми при обратном распространении ошибки, что затрудняет обновление весов в слоях сети и замедляет процесс обучения [14].

В основе архитектуры ResNet лежит концепция остаточного обучения (residual learning), согласно которой вместо непосредственного приближения некоторой целевой функции , каждый блок слоёв сети обучается аппроксимировать остаточную функцию:

,

где - это желаемая функция, которую сеть должна аппроксимировать, то есть преобразование входного сигнала в более информативное представление, а - это разница между и входными данными [14].

Для реализации остаточного обучения в архитектуре ResNet применяются так называемые пропускные соединения (shortcut connections), которые позволяют входным данным слоя напрямую передаваться в следующие слои, минуя один или несколько промежуточных слоёв. В простейшем случае пропускные соединения реализуются посредством прямого копирования данных без изменений, не увеличивая вычислительную сложность сети. Данная методика позволяет эффективно обучать очень глубокие нейронные сети, избегая проблем с затуханием градиентов. Авторы ResNet показали, что предложенная архитектура позволяет эффективно обучать сверхглубокие сети, например, сеть с 1202 слоями [14].

В оригинальной статье также представлены несколько версий моделей на основе ResNet, различающихся по глубине сети: ResNet-18, ResNet-34, ResNet-50, ResNet-101 и ResNet-152. Число после "ResNet-" указывает на количество слоёв в модели [14].

На рисунке 5 изображено визуальное представление архитектуры модели ResNet-34 [14].

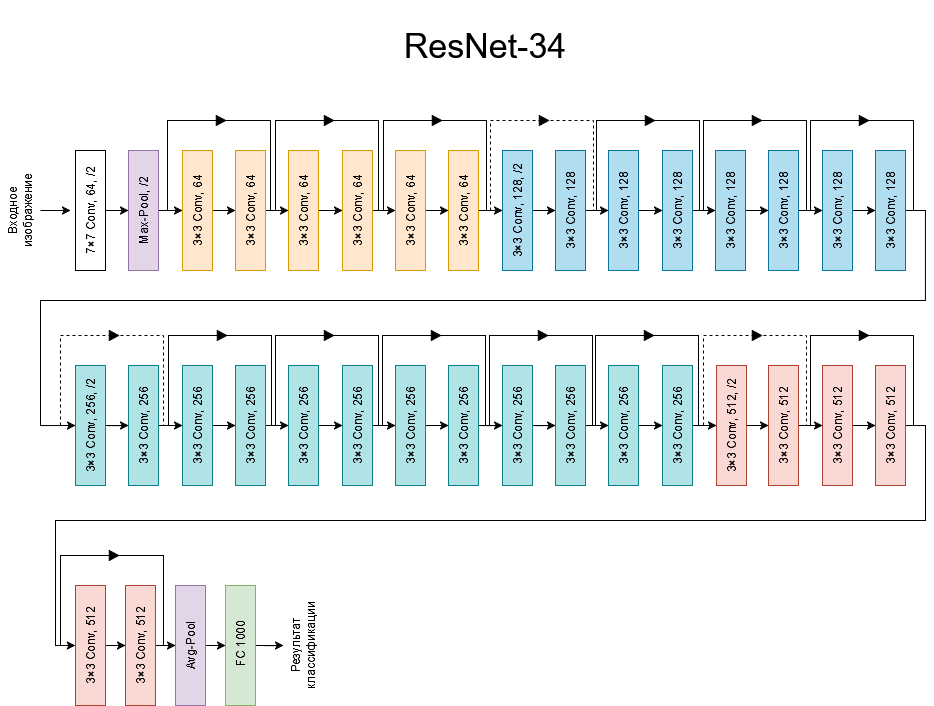


Рисунок 5 – Визуальное отображение архитектуры модели ResNet-34

1.2.3 Архитектура MobileNet

Архитектура MobileNet была представлена в 2017 году группой исследователей из компании Google. MobileNet проектировалась для работы в мобильных устройствах и встроенных системах и, как следствие, большой акцент при разработке уделялся производительности и эффективности. Ключевой особенностью MobileNet является использование DS Conv (depthwise separable convolutions) блоков свёртки, которые позволяют значительно уменьшить количество параметров и вычислительные затраты без существенной потери точности [15].

В стандартных слоях свёртки, применяемых, например, в VGG-19 каждый фильтр применяется ко всем каналам входного изображения, что приводит к значительным вычислительным затратам. В блоках DS Conv используется особый подход, заключающийся в разделении операции свёртки на два этапа [15]:

1. Cвёртка по глубине (depthwise convolution) - каждый фильтр применяется отдельно к каждому каналу входного изображения.

2. Точечная свёртка (pointwise convolution) - 1×1 свёртка, объединяющая признаки, выделенные свёрткой по глубине, и смешивающая информацию между каналами.

Использование DS Conv блоков свёртки сокращает количество операций умножения и сложения примерно в 8–9 раз по сравнению с традиционной свёрточной архитектурой, при этом обеспечивая лишь небольшое снижение точности [15].

На рисунке 6 представлено визуальное представление архитектуры модели на основе MobileNet, а также архитектура блока свёртки DS Conv [15].

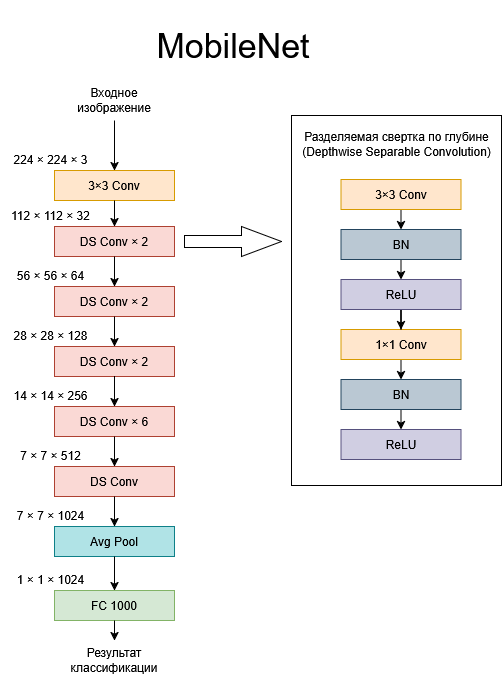


Рисунок 6 - Визуальное представление модели на основе архитектуры MobileNet и блока свёртки DS Conv

В дальнейшем были представлены вторая версия архитектуры в 2018 году (MobileNetV2) и третья версия в 2019 (MobileNetV3). Особенностью MobileNetV2 является использование обратных остаточных блоков свёртки (inverted residuals), которые являются модификацией ResNet блоков, и линейных узких слоев (linear bottlenecks). MobileNetV3, представляя собой прямую эволюцию MobileNetV2, является одной из первых архитектур СНС, при создании которой были использованы методы автоматического поиска архитектуры, такие как NAS (Neural Architecture Search) и NetAdapt. Также в MobileNetV3 была использована функция активации h-swish, пришедшая на замену ReLU, и модифицированные свёрточные блоки, включающие в себя слой squeeze-and-excitation (SE) модули, которые адаптивно изменяют весовые коэффициенты каналов, усиливая наиболее значимые признаки [16,17].

1.2.4. Архитектура EfficientNet

Архитектура EfficientNet, разработанная в 2019 году, отличается использованием метода комплексного масштабирования, позволяющего одновременно увеличивать глубину (количество слоев), ширину (число каналов в каждом слое) и разрешение (размер входных данных) сети с помощью единого коэффициента. В отличие от традиционных методов, где масштабируется только один параметр, такой подход позволяет добиться лучшего баланса между точностью и вычислительной эффективностью модели [18].

Помимо использования метода комплексного масштабирования, данную архитектуру выделяют блоки свёртки – MBConv. Данные блоки являются модификацией обратных остаточных блоков свёртки (используемых в архитектуре MobileNetV2) с применением механизма squeeze-and-excitation (SE). В модели EfficientNet блоки MBConv представлены в двух вариациях [18]:

1) MBConv1 – базовый вариант блока без расширяющего слоя (expansion layer).

2) MBConv6 - расширенный вариант блока, в котором используется коэффициент расширения 6, т.е. количество каналов во входном тензоре увеличивается в 6 раз. Это достигается за счёт применения 1×1 свёртки, которая расширяет пространство признаков, позволяя модели извлекать более сложные и выразительные признаки.

На рисунке 7 представлено визуальное представление модели на основе архитектуры EfficientNet, а также архитектуры блоков свёртки MBConv1 и MBConv6 [18].

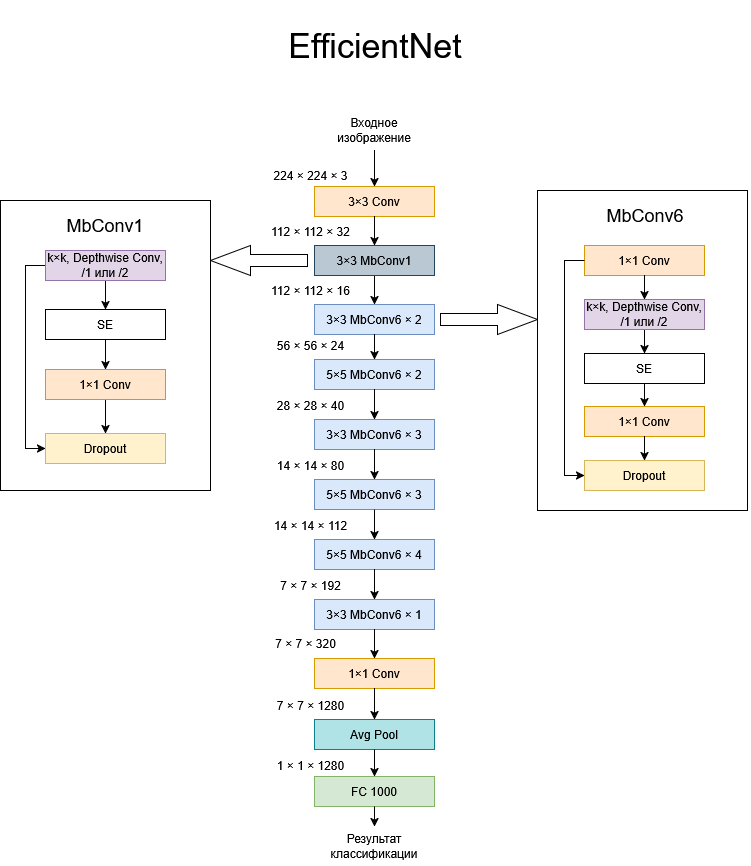
**

Рисунок 7 – Визуальное представление модели на основе архитектуры EfficientNet и архитектуры блоков MbConv1 и MbConv6

В 2021 году была разработана усовершенствованная версия данной архитектуры - EfficientNetV2, особенностью которой было использование нового поколения свёрточных блоков - Fused-MBConv, а также автоматической генерации архитектуры с помощью автоматического алгоритма поиска архитектуры NAS [4].

2. ОБУЧЕНИЕ НЕЙРОННЫХ СЕТЕЙ

2.1. Набор данных

Качество набора данных играет решающую роль в задаче реализации модели свёрточной нейронной сети для классификации воздушных судов по спутниковым изображениям, поскольку от него напрямую зависит точность, обобщаемость и эффективность работы свёрточных нейронных сетей.

Высококачественный набор данных должен обладать такими характеристиками, как полнота (наличие достаточного количества изображений для всех классов), аннотированная точность (корректность меток) и сбалансированность классов (равномерное распределение данных по категориям). Недостатки в этих аспектах могут привести к снижению точности модели, появлению смещений в предсказаниях и ухудшению способности нейросети адаптироваться к новым данным [19].

Для реализации задачи классификации воздушных судов использовался набор данных Multi-Type Aircraft Remote Sensing Images (далее - MTARSI). Приведенный набор данных включает 9385 спутниковых изображений самолетов 20 различных типов, что делает его актуальным и репрезентативным для задач автоматического распознавания воздушных судов. Изображения в MTARSI получены с различных спутниковых снимков, имеют сложные фоновые условия, варьируют по разрешению, освещению и времени съемки, что способствует созданию более устойчивых и обобщаемых моделей машинного обучения. Каждое изображение содержит ровно один самолет, а аннотации были выполнены семью экспертами в области интерпретации аэрокосмических данных. Пространственное разрешение изображений находится в диапазоне от 0,3 до 1,0 м, что обеспечивает высокую детализацию объектов [20].

Структура классов в наборе данных представлена в таблице 1 [20].

Таблица 1 – Структура классов в наборе данных MTARSI

|  |  |  |  |
| --- | --- | --- | --- |
| Тип воздушного судна | Количество изображений | Тип воздушного судна | Количество изображений |
| B-1 | 513 | C-130 | 763 |
| B-2 | 619 | C-135 | 526 |
| B-29 | 321 | C-17 | 480 |
| B-52 | 548 | C-5 | 499 |
| Boeing | 605 | E-3 | 452 |
| F-16 | 372 | F-22 | 846 |
| KC-10 | 554 | C-21 | 491 |
| U-2 | 362 | T-6 | 248 |
| P-63 | 305 | T-43 | 306 |

На рисунке 8 представлены примеры изображений из набора данных MTARSI с подписями соответствующих им классам.



Рисунок 8 – Примеры изображений из набора данных MTARSI

Используя подпрограмму, код которой представлен в приложении А, был получен средний размер изображений в наборе данных - 218**×**206 пикселей (ширина × высота). Для обучения нейронных сетей и дальнейшего использования в приложении использовался приближенный к данному значению размер изображений - 200**×**200 пикселей.

Для обучения нейронных сетей набор данных был разбит в пропорции 60% от общего числа изображений для обучения, 15% для валидации и 25% для тестирования.

2.2. Реализация нейронных сетей

Для создания, компиляции, обучения и последующей оценки свёрточных нейронных сетей использовались библиотеки для языка программирования Python – TensorFlow и Keras.

TensorFlow и Keras являются ключевыми инструментами в области глубокого обучения, широко используемыми для разработки и обучения нейронных сетей. TensorFlow, представленный в 2016 году, представляет собой систему машинного обучения, способную работать в различных масштабируемых средах, обеспечивая поддержку как обучения, так и вывода для глубоких нейронных сетей. Keras, в свою очередь, является высокоуровневым API, интегрированным в TensorFlow, который упрощает процесс создания и обучения моделей глубокого обучения [21,22].

Для загрузки и компиляции моделей была разработана функция load\_and\_compile\_model, позволяющая загружать предобученные модели, такие как VGG16, ResNet50, EfficientNetV2S и другие, с исключением верхних слоев и заданной формы входных данных (200**×**200**×**3 – ширина **×** высота **×** количество каналов). После загрузки базовой модели ее веса фиксировались, предотвращая их обновление во время обучения. Затем к базовой модели добавлялись слои глобального усреднения для уменьшения размерности и полносвязный слой с функцией активации softmax для классификации на 20 классов. Модели компилировались с использованием оптимизатора Adam и функции потерь категориальной кросс-энтропии.

Для обучения моделей была разработана функция train\_model. Процесс обучения состоял из 50 эпох, что обеспечило достаточное число итераций для сходимости модели. В качестве меры контроля за переобучением применялась ранняя остановка обучения в виде сохранения наилучшей версии модели (метод ModelCheckpoint), которая определялась минимальной функцией потерь на валидационном наборе данных. Также для повышения стабильности обучения использовался метод уменьшения скорости обучения (ReduceLROnPlateau), который снижал значение коэффициента скорости обучения (learning rate) в 2 раза при отсутствии улучшений в течение 3 эпох.

Для последующего анализа и оценки эффективности модели в ходе обучения сохранялись метрики топ-1 точности, топ-2 точности и топ-3 точности, которые характеризуют долю корректных предсказаний в пределах первого, второго и третьего наиболее вероятных классов соответственно.

После завершения обучения производилась финальная оценка модели на тестовом наборе данных с помощью функции evaluate\_model. Помимо метрик точности (топ-1, топ-2 и топ-3) и функции потери (loss) на тестовом наборе данных вычислялось среднее время инференса, характеризующее скорость обработки одного изображения. Полученные результаты фиксировались в файл evaluation\_results.csv, что позволило произвести детальный сравнительный анализ производительности различных архитектур нейронных сетей.

Полный код программы для создания, компиляции, обучения и оценки моделей представлен в приложении Б.

3. АНАЛИЗ РЕЗУЛЬТАТОВ И РАЗРАБОТКА ИНТЕРФЕЙСА

3.1. Анализ результатов

На основе метрик, зафиксированных в ходе обучения моделей, были построены графики зависимости точности на валидационном наборе данных от количества эпох (рисунок 9), а также графики изменения значения функции потерь (loss) на тех же данных в процессе обучения (рисунок 10).

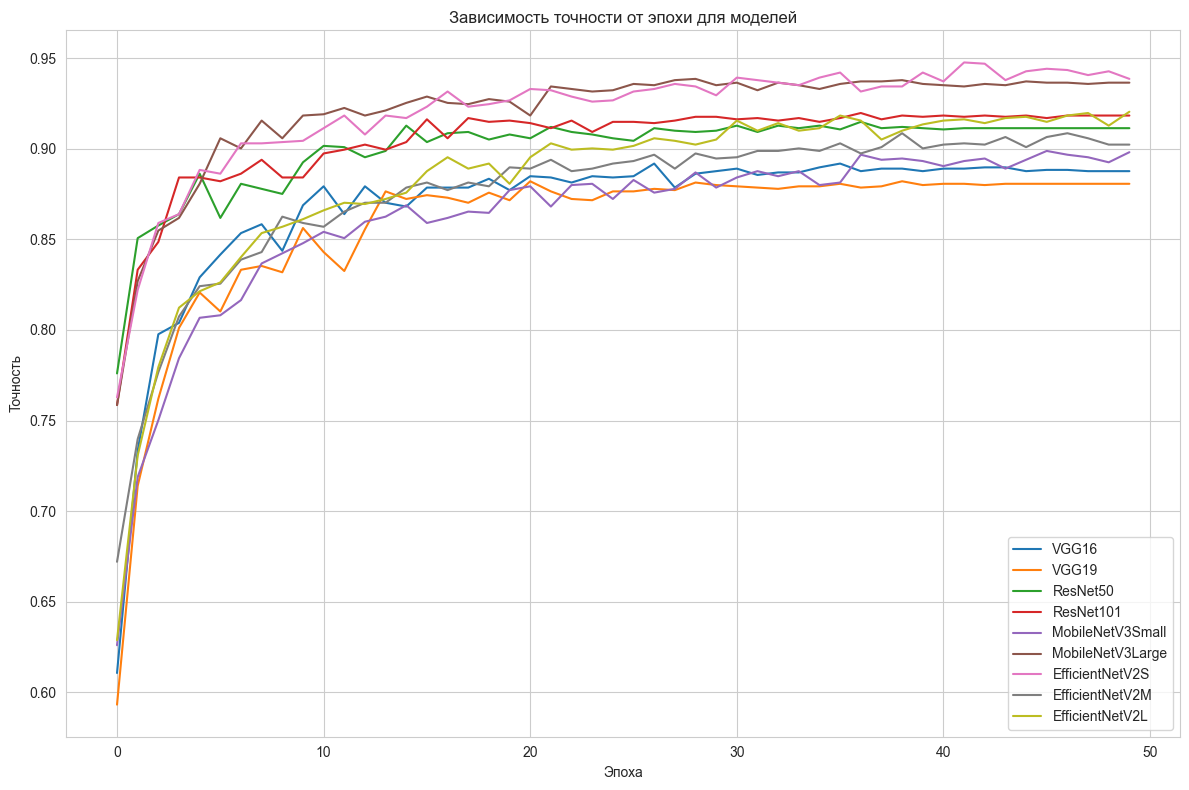


Рисунок 9 – Графики зависимости топ-1 точности от эпохи

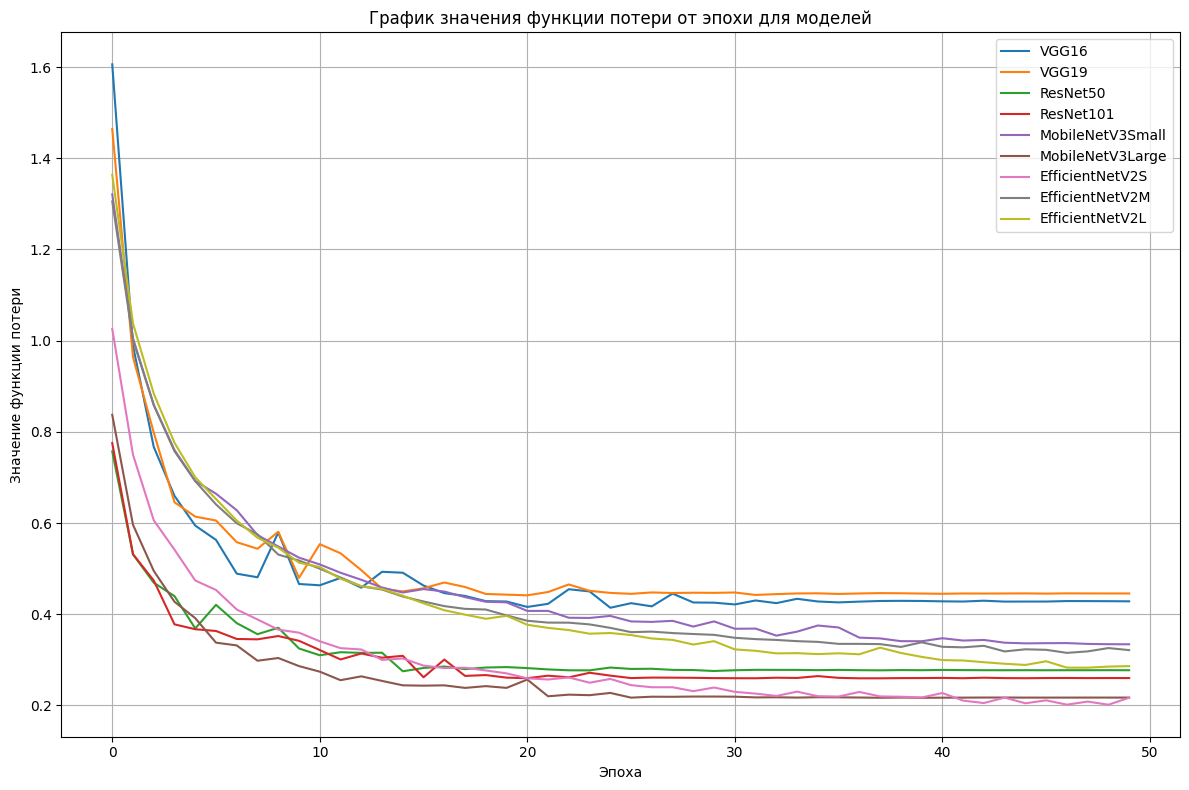


Рисунок 10 – Графики зависимости значения функции потерь от эпохи

Из графиков на рисунках 9 и 10 можно сделать ряд наблюдений относительно динамики обучения различных моделей. На графике зависимости точности от количества эпох видно, что большинство моделей демонстрируют быстрый рост точности на начальных этапах обучения, после чего темп улучшений замедляется. Некоторые архитектуры, такие как EfficientNetV2S и EfficientNetV2M, показывают более стабильное повышение точности и достигают лучших результатов по сравнению с другими моделями.

Анализ графиков зависимости функции потерь подтверждает данную тенденцию. Для всех моделей наблюдается значительное снижение значения loss на первых эпохах, после чего значения функции потерь выходят на плато. При этом можно отметить, что модели семейства EfficientNet демонстрируют более плавное уменьшение loss и достигают меньших финальных значений функции потерь, что может свидетельствовать о лучшем качестве обучения. В то же время модели, такие как VGG16 и VGG19, демонстрируют более выраженные колебания, что может указывать на менее стабильный процесс обучения.

На основе данных, полученных из оценки моделей на тестовом датасете, были построены графики, отображающие ключевые метрики, включая точность предсказаний моделей (рисунок 11), значение функции потерь (рисунок 12), а также среднее время обработки одного изображения (рисунок 13).

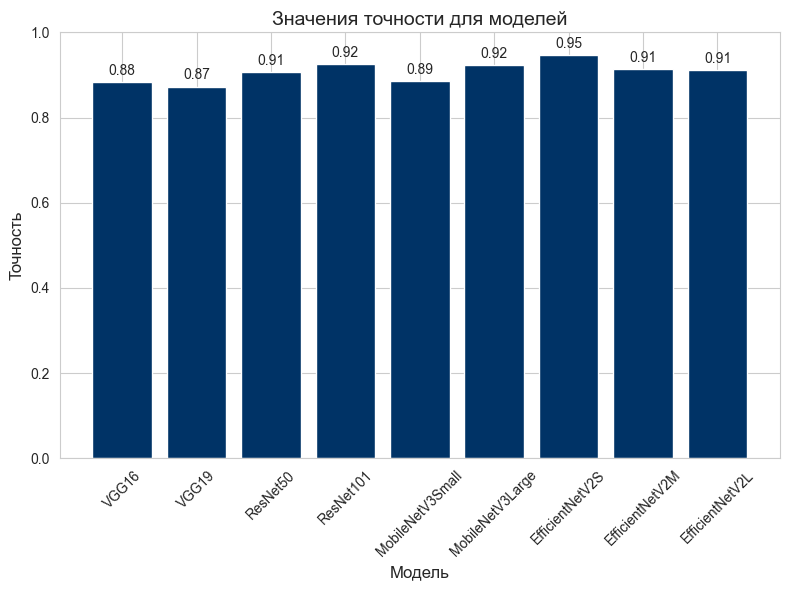


Рисунок 11 – График значений точности для каждой модели



Рисунок 12 – График значений функции потерь для каждой модели

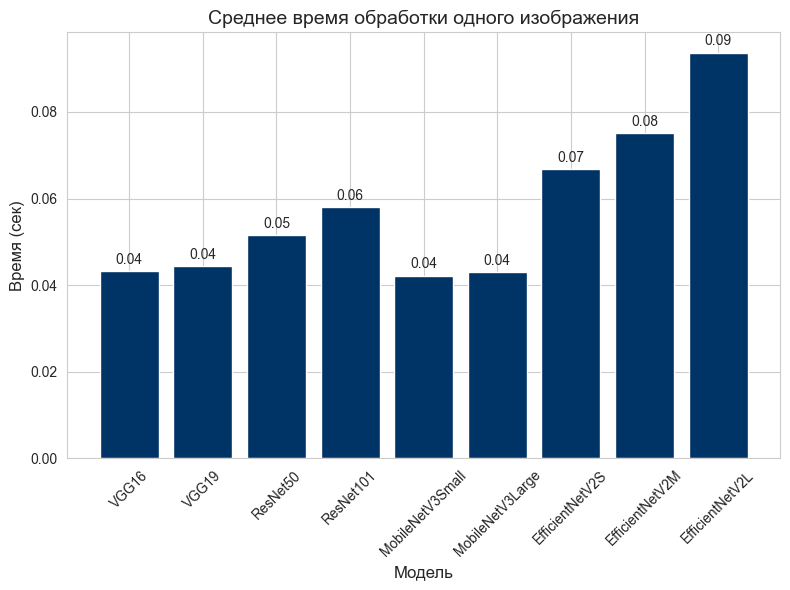


Рисунок 13 – График значений среднего времени обработки одно потерь для каждой модели

СПИСОК ЛИТЕРАТУРЫ

1. Сикорский О.С. Обзор свёрточных нейронных сетей для задачи классификации изображений // Новые информационные технологии в автоматизированных системах. 2017. № 20. P. 37–42.

2. Маршалко Д.В., Кубанских О.В. Архитектура свёрточных нейронных сетей // Ученые записки Брянского государственного университета. 2019. № 4 (16). P. 10–13.

3. Lecun Y. et al. Gradient-based learning applied to document recognition // Proc. IEEE. 1998. Vol. 86, № 11. P. 2278–2324.

4. Tan M., Le Q.V. EfficientNetV2: Smaller Models and Faster Training: arXiv:2104.00298. arXiv, 2021.

5. Parhi R., Nowak R.D. The Role of Neural Network Activation Functions // IEEE Signal Process. Lett. 2020. Vol. 27. P. 1779–1783.

6. Zafar A. et al. A Comparison of Pooling Methods for Convolutional Neural Networks: 17 // Applied Sciences. Multidisciplinary Digital Publishing Institute, 2022. Vol. 12, № 17. P. 8643.

7. Нгуен К. Т, Боровик И. Г. Сравнение функций активации softmax и сигмоида в контексте многоклассовой классификации распознавания объектов // Международный научно-исследовательский журнал. 2024. № 6 (144).

8. Krizhevsky A., Sutskever I., Hinton G.E. ImageNet classification with deep convolutional neural networks // Commun. ACM. 2017. Vol. 60, № 6. P. 84–90.

9. Федотов И.с. Трансферное обучение в компьютерном зрении: применение и перспективы // Вестник науки. 2023. Vol. 1, № 9 (66). P. 135–137.

10. Russakovsky O. et al. ImageNet Large Scale Visual Recognition Challenge: arXiv:1409.0575. arXiv, 2015.

11. Simonyan K., Zisserman A. Very Deep Convolutional Networks for Large-Scale Image Recognition: arXiv:1409.1556. arXiv, 2015.

12. Siddharth T., Kirar B.S., Agrawal D.K. Plant Species Classification Using Transfer Learning by Pretrained Classifier VGG-19: arXiv:2209.03076. arXiv, 2022.

13. Apostolopoulos I.D., Tzani M. Industrial object, machine part and defect recognition towards fully automated industrial monitoring employing deep learning. The case of multilevel VGG19 // J Ambient Intell Human Comput. 2023. Vol. 14, № 8. P. 10263–10276.

14. He K. et al. Deep Residual Learning for Image Recognition: arXiv:1512.03385. arXiv, 2015.

15. Howard A.G. et al. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications: arXiv:1704.04861. arXiv, 2017.

16. Sandler M. et al. MobileNetV2: Inverted Residuals and Linear Bottlenecks: arXiv:1801.04381. arXiv, 2019.

17. Howard A. et al. Searching for MobileNetV3: arXiv:1905.02244. arXiv, 2019.

18. Tan M., Le Q.V. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks: arXiv:1905.11946. arXiv, 2020.

19. Gong Y. et al. A survey on dataset quality in machine learning // Information and Software Technology. 2023. Vol. 162. P. 107268.

20. Wu Z.-Z. et al. A benchmark data set for aircraft type recognition from remote sensing images // Applied Soft Computing. 2020. Vol. 89. P. 106132.

21. Abadi M. et al. TensorFlow: A system for large-scale machine learning: arXiv:1605.08695. arXiv, 2016.

22. Павлов Е.М., Рыжов А.В. Применение нейронных сетей для распознавания образов // Бюллетень науки и практики. 2023. Vol. 9, № 12. P. 52–58.

ПРИЛОЖЕНИЕ А

Код программы для получения среднего размера изображений в наборе данных

def calculate\_average\_image\_size(dataset\_path):

    total\_width, total\_height, image\_count = 0, 0, 0

    for class\_folder in os.listdir(dataset\_path):

        class\_path = os.path.join(dataset\_path, class\_folder)

        if os.path.isdir(class\_path):

            for image\_name in os.listdir(class\_path):

                image\_path = os.path.join(class\_path, image\_name)

                try:

                    with Image.open(image\_path) as img:

                        width, height = img.size

                        total\_width += width

                        total\_height += height

                        image\_count += 1

                except Exception as e:

                    print(f"Ошибка при обработке файла {image\_path}: {e}")

    if image\_count == 0:

        return (0, 0)  # Нет изображений

    average\_width = total\_width / image\_count

    average\_height = total\_height / image\_count

    return average\_width, average\_height

# Путь к папке

dataset\_path = "MTARSI\_Dataset"

average\_size = calculate\_average\_image\_size(dataset\_path)

print(f"Средний размер изображений: {average\_size[0]:.2f}x{average\_size[1]:.2f}")

ПРИЛОЖЕНИЕ Б

Код программы для создания, компиляции, обучения и оценки моделей

TARGET\_SIZE = (200, 200)

# Генератор без нормализации

datagen\_default = ImageDataGenerator()

# Генераторы для стандартной загрузки данных

train\_generator = datagen\_default.flow\_from\_directory(

    directory=r"processed\_dataset/train",

    target\_size=TARGET\_SIZE,

    color\_mode="rgb",

    batch\_size=16,

    class\_mode="categorical",

    seed=42

)

validation\_generator = datagen\_default.flow\_from\_directory(

    directory=r"processed\_dataset/validation",

    target\_size=TARGET\_SIZE,

    color\_mode="rgb",

    batch\_size=16,

    class\_mode="categorical",

    seed=42

)

test\_generator = datagen\_default.flow\_from\_directory(

    directory=r"processed\_dataset/test",

    target\_size=TARGET\_SIZE,

    color\_mode="rgb",

    batch\_size=1,

    class\_mode="categorical",

    shuffle=False,

    seed=42

)

# Функция загрузки и компиляции модели

def load\_and\_compile\_model(model\_name, input\_shape=(200, 200, 3), num\_classes=20):

    if model\_name == 'VGG16':

        base\_model = VGG16(weights='imagenet', include\_top=False, input\_shape=input\_shape)

    elif model\_name == 'VGG19':

        base\_model = VGG19(weights='imagenet', include\_top=False, input\_shape=input\_shape)

    elif model\_name == 'ResNet50':

        base\_model = ResNet50(weights='imagenet', include\_top=False, input\_shape=input\_shape)

    elif model\_name == 'ResNet101':

        base\_model = ResNet101(weights='imagenet', include\_top=False, input\_shape=input\_shape)

    elif model\_name == 'MobileNetV3Small':

        base\_model = MobileNetV3Small(weights='imagenet', include\_top=False, input\_shape=input\_shape)

    elif model\_name == 'MobileNetV3Large':

        base\_model = MobileNetV3Large(weights='imagenet', include\_top=False, input\_shape=input\_shape)

    elif model\_name == 'EfficientNetV2S':

        base\_model = EfficientNetV2S(weights='imagenet', include\_top=False, input\_shape=input\_shape)

    elif model\_name == 'EfficientNetV2M':

        base\_model = EfficientNetV2M(weights='imagenet', include\_top=False, input\_shape=input\_shape)

    elif model\_name == 'EfficientNetV2L':

        base\_model = EfficientNetV2L(weights='imagenet', include\_top=False, input\_shape=input\_shape)

    base\_model.trainable = False

    model = tf.keras.Sequential([

        base\_model,

        tf.keras.layers.GlobalAveragePooling2D(),

        tf.keras.layers.Dense(num\_classes, activation='softmax')

    ])

    # Компиляция модели с метриками Top-1, Top-2 и Top-3 Accuracy

    model.compile(optimizer=Adam(),

                  loss=CategoricalCrossentropy(),

                  metrics=[

                      tf.keras.metrics.TopKCategoricalAccuracy(k=1, name='top\_1\_accuracy'),

                      tf.keras.metrics.TopKCategoricalAccuracy(k=2, name='top\_2\_accuracy'),

                      tf.keras.metrics.TopKCategoricalAccuracy(k=3, name='top\_3\_accuracy')

                  ])

    return model

# Функция обучения модели

def train\_model(model, train\_generator, validation\_generator, model\_name):

    csv\_logger = CSVLogger(f'Training\_logs\{model\_name}\_imagenet\_training\_log.csv', append=True)

    checkpoint = ModelCheckpoint(

        filepath=f'{model\_name}\_best\_model.h5',

        monitor='val\_loss',

        save\_best\_only=True,

        mode='min',

        verbose=1

    )

    reduce\_lr = ReduceLROnPlateau(

        monitor='val\_loss',

        factor=0.5,

        patience=3,

        verbose=1,

        min\_delta=1e-4,

        cooldown=2,

        min\_lr=1e-7

    )

    history = model.fit(

        train\_generator,

        epochs=50,

        validation\_data=validation\_generator,

        callbacks=[csv\_logger, checkpoint, reduce\_lr]

    )

    return history

# Функция для вычисления метрик

def evaluate\_model(model, test\_generator, model\_name):

    # Рассчитываем среднее время инференса

    inference\_times = []

    for \_ in range(len(test\_generator)):

        start\_time = time.time()

        X\_test\_batch, \_ = test\_generator.next()  # Получаем следующий батч

        model.predict(X\_test\_batch, verbose=0)

        inference\_times.append(time.time() - start\_time)

    avg\_inference\_time = np.mean(inference\_times)

    # Оценка модели и извлечение метрик

    results = model.evaluate(test\_generator, verbose=0)

    metric\_names = model.metrics\_names

    loss = results[0]

    top\_1 = results[metric\_names.index('top\_1\_accuracy')]

    top\_2 = results[metric\_names.index('top\_2\_accuracy')]

    top\_3 = results[metric\_names.index('top\_3\_accuracy')]

    return {

        'Model': model\_name,

        'Loss': loss,

        'Top-1 Accuracy': top\_1,

        'Top-2 Accuracy': top\_2,

        'Top-3 Accuracy': top\_3,

        'Avg Inference Time (s)': avg\_inference\_time

    }

# Основной процесс

def main():

    # Список используемых моделей

    models = [

        'VGG16', 'VGG19', 'ResNet50', 'ResNet101',

        'MobileNetV3Small', 'MobileNetV3Large',

        'EfficientNetV2S', 'EfficientNetV2M', 'EfficientNetV2L'

    ]

    csv\_filename = 'evaluation\_results.csv'

    for model\_name in models:

        print(f"Training {model\_name}")

        model = load\_and\_compile\_model(model\_name)

        train\_model(model, train\_generator, validation\_generator, model\_name)

        # Загрузка лучшей версии модели

        best\_model = tf.keras.models.load\_model(f'{model\_name}\_best\_model.h5')

        # Оценка модели

        print(f"Evaluating {model\_name}")

        metrics = evaluate\_model(best\_model, test\_generator, model\_name)

        # Вывод результатов в консоль

        for key, value in metrics.items():

            print(f"{key}: {value}")

        # Сохранение результатов после каждой модели (результаты добавляются, если файл существует)

        metrics\_df = pd.DataFrame([metrics])

        if not os.path.isfile(csv\_filename):

            metrics\_df.to\_csv(csv\_filename, index=False, mode='w')

        else:

            metrics\_df.to\_csv(csv\_filename, index=False, header=False, mode='a')

        print(f"Results for {model\_name} saved to {csv\_filename}\n")

# Запуск основного процесса

main()