СОДЕРЖАНИЕ

ВВЕДЕНИЕ 2
1 Анализ требований 3
2 Подготовка датасета 4
2.1 Выбор источника размеченных фотографий 4
2.2 Генерация искаженных изображений 6
2.3 Разметка датасета 10
3 Разработка нейронной сети 12
3.1 Выбор архитектуры 12
3.2 Реализация нейронной сети 14
3.2.1 Первое поколение 14
3.2.2 Второе поколение 17
ЗАКЛЮЧЕНИЕ 22
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 23
ПРИЛОЖЕНИЕ А ИСХОДНЫЙ ТЕКСТ ПРОГРАММЫ 24

ВВЕДЕНИЕ

В рамках преддипломной практики проводилась работа над дипломным проектом «Программная система для редактирования изображений с автоматическим отбором фотографий». Разрабатываемая система представляет собой приложение-редактор, дополненное модулем интеллектуальной оценки технического качества снимков. Назначение модуля - автоматизировать первичный отбор удачных кадров и сократить время пользователя на просмотр и сортировку фотоматериала.

Ежедневно пользователи делают большое количество фотографий, однако значимая часть кадров оказывается технически неудачной. При этом ручная оценка качества требует времени. В связи с этим актуальной является разработка решения, которое сможет определять распространённые технические дефекты.

Цель преддипломной практики - разработать нейронную сеть, которая автоматически выявляет технические дефекты фотографий.

Для достижения поставленной цели были сформулированы следующие задачи:

1) изучить предметную область автоматической оценки технического качества изображений и определить перечень целевых дефектов;

2) подготовить датасет для обучения модели;

3) спроектировать и реализовать архитектуру нейронной сети;

4) подобрать параметры обучения и метрики оценки;

5) провести валидацию качества и проанализировать результаты.

1 Анализ требований

Разрабатываемая в рамках дипломного проекта система предназначена для редактирования изображений и автоматического отбора фотографий по техническому качеству. Следовательно, ключевое требование к нейросетевой части системы - способность устойчиво определять, является ли снимок технически удачным, без учёта содержания сцены. Оценка должна быть применима к повседневным пользовательским фотографиям.

В контексте задачи отбора важно, чтобы результат работы нейронной сети был интерпретируемым и пригодным для использования в логике приложения: например, позволять настраивать пороги чувствительности. Также система должна быть рассчитана на практическое применение, поэтому требования включают стабильность работы на изображениях в распространённых форматах.

С учётом назначения приложения были выделены дефекты, которые наиболее часто встречаются в пользовательской съёмке, заметно снижают техническое качество и при этом могут быть формализованы для автоматического определения. Кроме того, выбранные дефекты должны быть релевантны для последующей интеграции в редактор и принятия решения об отбраковке кадра.

В результате анализа требований к системе в качестве целевых технических дефектов выбраны:

1) нерезкость (смаз/размытие) - в том числе из-за промаха фокуса и движения камеры/объекта;

2) недосвет - изображение слишком тёмное, с потерей деталей в тенях;

3) пересвет - слишком яркое изображение, с потерей деталей в светлых участках изображения.

Именно эти три класса дефектов используются в дальнейшем при подготовке данных, проектировании нейронной сети и оценке качества её работы.

2 Подготовка датасета

2.1 Выбор источника размеченных фотографий

Для обучения нейронной сети, определяющей технические дефекты фотографий, требуется набор данных с большим количеством изображений и корректной разметкой по целевым дефектам. При этом важно, чтобы разметка отражала именно техническое качество (нерезкость, недосвет, пересвет), а не смысл или «удачность» сцены. То есть разметка должна быть результатов не субъективной оценки «нравится»/«не нравится» опрошенных людей. На практике получить такой набор данных непросто: ручная разметка тысяч фотографий требует значительных трудозатрат и остаётся частично субъективной, особенно в пограничных случаях.

При формировании данных возможны два подхода. Первый - использование реальных датасетов с оценками качества. Такие наборы ближе к реальным пользовательским условиям, однако часто содержат только общую оценку качества без явного указания типа дефекта. Кроме того, часть датасетов ориентирована на «эстетическую» (т.е. субъктивную) оценку, что усложняет выделение конкретных технических причин брака.

Второй подход - формирование синтетического датасета на основе исходных качественных изображений, когда к «чистым» фотографиям программно добавляются контролируемые искажения (размытие, изменение яркости и т.п.). В этом случае разметка формируется автоматически и не зависит от субъективной оценки, можно получить достаточное число примеров каждого дефекта, управлять силой дефекта через уровни искажений и обеспечивать воспроизводимость экспериментов.

С учётом ограничений преддипломной практики (необходимость быстро получить достаточный объём данных с однозначной разметкой) более рациональным является второй подход - обучение на синтетических искажениях.

В качестве источника данных выбран датасет KADIS-700K. Он содержит исходные изображения и большое число их версий с искусственно добавленными искажениями различных типов и уровней. Это позволяет сформировать выборку, напрямую соответствующую поставленной задаче: выделить только те типы искажений, которые моделируют нерезкость (размытие/смаз), недосвет и пересвет, и автоматически сформировать разметку для обучения нейронной сети.

KADIS-700K — крупный синтетический набор для оценки качества изображений. Он включает около 140 000 исходных (неискажённых) изображений и примерно 700 000 искажённых: для каждого исходного изображения подготовлено несколько вариантов, полученных добавлением искажений, выбираемых из набора 25 типов с 5 уровнями выраженности. Набор искажений формируется с помощью кода генерации искажений (Matlab), что позволяет точно знать тип и уровень добавленного дефекта.

В рамках дипломного проекта, конечно, рассматриваются только три дефекта, и поэтому в обучении не будут участвовать все 840 000 изображений, но на их базе можно сформировать достаточную выборку изображений с нужными дефектами. Из всего спектра искажений KADIS-700K используются только следующие типы:

1) нерезкость/смаз: lens blur (dist_type = 2) и motion blur (dist_type = 3);

2) пересвет: brighten (dist_type = 16);

3) недосвет: darken (dist_type = 17).

2.2 Генерация искаженных изображений

Для формирования обучающей выборки на основе KADIS-700K использовался исходный скрипт генерации, поставляемый вместе с кодом добавления искажений (code_imdistort) и функцией imdist_generator. В оригинальном варианте скрипт последовательно проходил по списку исходных изображений из файла kadis700k_ref_imgs.csv, считывал каждое изображение из каталога ref_imgs/ и для заданного типа искажения генерировал все пять уровней (от 1 до 5). Исходный код добавления искажений представлен в листинге 1.

Листинг 1 — Исходный код добавления искажений

```
%% setup
clear; clc;
addpath(genpath('code_imdistort'));


%% read the info of pristine images

tb = readtable('kadis700k_ref_imgs.csv');
tb = table2cell(tb);

%% generate distorted images in dist_imgs folder

for i = 1:size(tb,1)
    ref_im = imread(['ref_imgs/' tb{i,1}]);
    dist_type = tb{i,2};

    for dist_level = 1:5
        [dist_im] = imdist_generator(ref_im, dist_type,
dist_level);
        strs = split(tb{i,1},'.');
        dist_im_name = [strs{1}  '_' num2str(tb{i,2},'%02d')
'_' num2str(dist_level,'%02d') '.bmp'];
        disp(dist_im_name);
        imwrite(dist_im, ['dist_imgs/' dist_im_name]);
    end

end
```

В рамках преддипломной практики данный скрипт был адаптирован под задачи дипломного проекта. Во-первых, из полного набора искажений KADIS-700K были оставлены только те, которые соответствуют целевым

дефектам. Для этого после чтения таблицы выполнялась фильтрация по dist_type, и дальнейшая генерация выполнялась только для типов 2, 3, 16 и 17.

Во-вторых, вместо перебора всех уровней искажения для каждого изображения уровни выбирались случайно из заранее заданных допустимых наборов, что позволило сконцентрировать данные на более заметных дефектах и избежать слабых искажений, которые хуже соответствуют практическим ситуациям. Использовались следующие диапазоны: для dist_type = 2 уровни [2, 3, 4], для dist_type = 3 уровни [4, 5], для dist_type = 16 уровни [4, 5], для dist_type = 17 уровни [4, 5].

В-третьих, изменён формат сохранения искажённых изображений. Вместо исходного расширения .bmp результаты сохранялись в .jpg с параметром качества 95. Такой выбор сделан для приближения данных к реальным пользовательским фотографиям (в большинстве случаев они хранятся в JPEG), а также для ускорения чтения с диска и уменьшения занимаемого объёма памяти. Имена файлов формировалисьпо схеме name_XX_YY.jpg, где к базовому имени исходного файла добавлялись код типа искажения XX и уровень YY, записанные в двухзначном формате.

Дополнительно в скрипт были внесены проверки, повышающие устойчивость генерации. Если целевой файл уже существует в dist_imgs/, генерация пропускается, что позволяет безопасно перезапускать процесс без перезаписи уже сформированных данных. К тому же это позволило снизить избыточную корреляцию в данных: если сохранять множество вариантов одной и той же сцены, модель может переобучаться на характерные детали конкретного изображения и демонстрировать завышенные метрики, вместо устойчивого распознавания дефектов на разных сценах.

Модифицированный код представлен в листинге 2 и в приложении А.

Листинг 2 — Модифицированный код добавления искажений

```
%% setup
clear; clc;
addpath(genpath('code_imdistort'));

fid = fopen('labels.csv','w');
```

Продолжение листинга 2

```
fprintf(fid,
'path,blur,under,over,dist_type,dist_level,ref\n');

tb = readtable('kadis700k_ref_imgs.csv');

% --- ОСТАВЛЯЕМ ТОЛЬКО НУЖНЫЕ ИСКАЖЕНИЯ ---
types = [2 3 16 17];            % gaussian/lens/motion blur +
brighten + darken
tb = tb(ismember(tb{:,2}, types), :);  % 2-й столбец =
dist_type

tb = table2cell(tb);

ref_list = unique(tb(:,1));

for r = 1:numel(ref_list)
    ref_name = ref_list{r};
    ref_path = fullfile('ref_imgs', ref_name);

    % дефектов нет
    blur = 0; under = 0; over = 0;
    dist_type = 0; dist_level = 0;

    fprintf(fid, '%s,%d,%d,%d,%d,%d,%s\n', ref_path, blur,
under, over, dist_type, dist_level, ref_name);
end


%% generate distorted images in dist_imgs folder

for i = 1:size(tb,1)
    ref_im = imread(fullfile('ref_imgs', tb{i,1}));
    dist_type = tb{i,2};

    if dist_type == 2
        allowed_levels = [2 3 4];
    elseif dist_type == 3
        allowed_levels = [4 5];
    elseif dist_type == 16
        allowed_levels = [4 5];
    elseif dist_type == 17
        allowed_levels = [4 5];
    else
        continue;
    end
```

Продолжение листинга 2

```
  %%  for k = 1:numel(keep_levels)
        dist_level =
allowed_levels(randi(numel(allowed_levels)));

        dist_im = imdist_generator(ref_im, dist_type,
dist_level);

        % имя файла остаётся совместимым: _тип_уровень.bmp
        strs = split(tb{i,1},'.');

        dist_im_name = [strs{1} '_' num2str(dist_type,'%02d')
'_' num2str(dist_level,'%02d') '.jpg'];
        out_path = fullfile('dist_imgs', dist_im_name);

        if exist(out_path, 'file')
            continue;
        end

        imwrite(dist_im, out_path, 'Quality', 95);

        % вычисляем метки по dist_type
        blur = 0; under = 0; over = 0;
        if dist_type == 2 || dist_type == 3
            blur = 1;
        elseif dist_type == 17
            under = 1;
        elseif dist_type == 16
            over = 1;
        end

        fprintf(fid, '%s,%d,%d,%d,%d,%d,%s\n', out_path, blur,
under, over, dist_type, dist_level, tb{i,1});

 %%   end
end

fclose(fid);
```

В результате выполненной модификации был реализован воспроизводимый процесс генерации искажённых изображений, который формирует только те варианты, что соответствуют выбранным дефектам дипломного проекта, и сохраняет их в формате, близком к условиям практического использования в приложении.

2.3 Разметка датасета

После генерации искажённых изображений был сформирован единый файл разметки labels.csv, содержащий пути к изображениям и целевые метки дефектов. Для дальнейшего обучения нейронной сети требовалось разделить данные на обучающую и валидационную выборки так, чтобы оценка качества была корректной и не завышалась из-за попадания в обе выборки разных версий одной и той же сцены.

Разделение «по строкам» в данном случае является нежелательным, так как в датасете присутствуют несколько вариантов одного исходного изображения (исходное и искажённые версии). Если часть вариантов одной сцены окажется в обучающей выборке, а другая часть — в валидационной, нейронная сеть может демонстрировать завышенные метрики за счёт узнавания особенностей конкретной сцены, а не за счёт устойчивого распознавания дефектов. Чтобы избежать такой утечки, разделение выполнялось не по отдельным файлам, а по идентификатору исходного изображения.

В разметке для этого используется поле ref, которое хранит имя исходного файла и тем самым группирует все варианты одной сцены. В скрипте сначала загружается labels.csv, затем выполняется разделение по уникальным значениям ref. Множество исходных сцен случайно разбивается на две части в пропорции 80/20. После этого строки, относящиеся к выбранным ref, собираются в train_df, а остальные — в val_df. Полученные таблицы сохраняются в файлы train.csv и val.csv.

Скрипт, разделяющий датасет на обучающую и валидационную выборки содержится в 3 и в приложении А.

Листинг 3 — Скрипт, разделяющий датасет на обучающую и валидационную выборки

```
import os
import pandas as pd
from sklearn.model_selection import train_test_split

LABELS_PATH = "labels.csv"
```

Продолжение листинга 3

```
TRAIN_OUT = "train.csv"
VAL_OUT = "val.csv"
VAL_RATIO = 0.2
SEED = 42

df = pd.read_csv(LABELS_PATH)

df["path"] = df["path"].astype(str).str.replace("\\",
os.sep).str.replace("/", os.sep)

# Сплит по ref (чтобы не было утечки)
refs = df["ref"].astype(str).unique()
train_refs, val_refs = train_test_split(refs,
test_size=VAL_RATIO, random_state=SEED, shuffle=True)

train_df =
df[df["ref"].astype(str).isin(train_refs)].reset_index(drop=True)
val_df   =
df[df["ref"].astype(str).isin(val_refs)].reset_index(drop=True)

train_df.to_csv(TRAIN_OUT, index=False)
val_df.to_csv(VAL_OUT, index=False)

print("Saved:", TRAIN_OUT, len(train_df))
print("Saved:", VAL_OUT, len(val_df))
print("Unique refs train/val:", len(train_refs),
len(val_refs))
```

В результате такой процедуры обеспечивается отсутствие пересечения сцен между выборками: одна и та же исходная фотография и её искажённые версии могут принадлежать только обучающей или только валидационной части. Это позволяет получить более честную оценку качества нейронной сети и избежать завышения метрик из-за утечки данных. В результате мы получили готовый разделенный датасет, включающий в себя как данные, так и разметку.

3 Разработка нейронной сети

3.1 Выбор архитектуры

Для задачи автоматического отбора фотографий требовалось решение, которое одновременно устойчиво выделяет технические дефекты на разных сценах, достаточно лёгкое для интеграции в приложение и выдаёт раздельные оценки по каждому дефекту, чтобы затем можно было настраивать пороги и логику отбора изображений.

С учётом этого была выбрана архитектура по схеме «общая основа + несколько выходов». Общая часть (основа) извлекает универсальные признаки изображения, а далее идут отдельные выходные блоки, каждый из которых отвечает за свой дефект. Такой подход особенно удобен в задаче, где дефекты могут встречаться независимо друг от друга: фотография может быть одновременно нерезкой и недоэкспонированной, либо иметь только один дефект. Поэтому вместо выбора одной оценки хорошо/плохо используется многометочная постановка: сеть выдаёт три независимые оценки - по нерезкости, недосвету и пересвету.

На практике входное изображение подаётся в основу EfficientNetB0 (предобученную на ImageNet, без верхней классификационной части), далее применяется глобальное агрегирование признаков и регуляризация (dropout). Затем от общего вектора признаков отходят три параллельных выхода, каждый из которых предсказывает один логит (одно число) - отдельно для нерезкости, недосвета и пересвета. Таким образом, начальная версия нейронной сети представляла собой EfficientNetB0 + 3 выхода по одному нейрону. На выходе сеть формирует три значения, каждое из которых отражает наличие или отсутствие соответствующего дефекта.

Итоговая архитектура схематично представлена на рисунке 1.

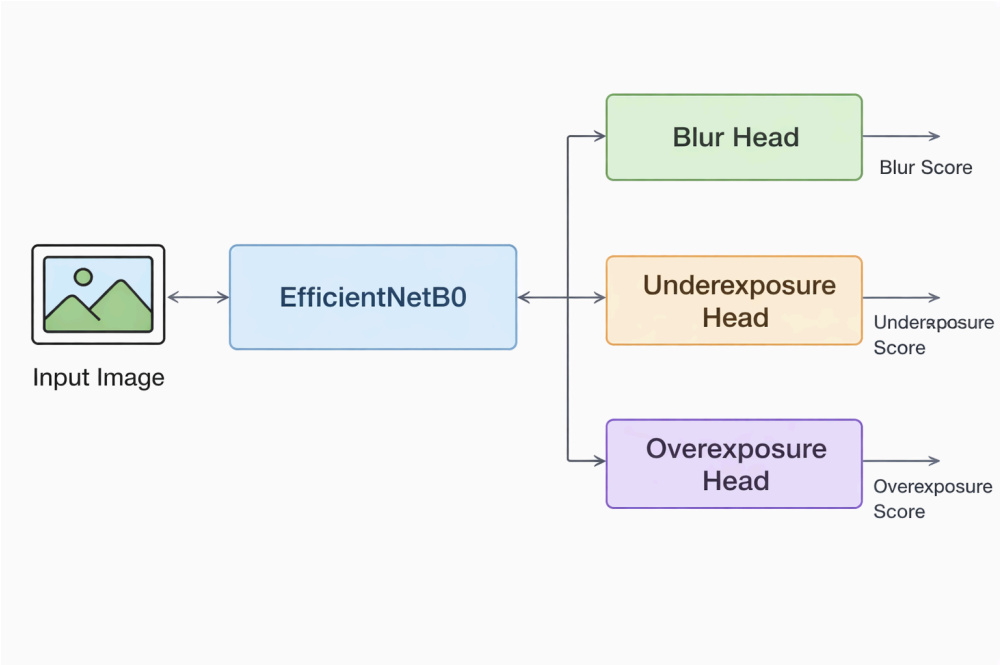

Рисунок 1 — Архитектура нейронной сети

Выбор EfficientNetB0 как основы был сделан на базе результатов научно-исследовательской работы, выполненной в 7 семестре. В рамках НИРа сравнивались несколько популярных архитектур по трём критериям: число параметров, вычислительная сложность и итоговое качество. Рассматривались, в частности, ResNet18, MobileNetV2, EfficientNet-B0 и более тяжёлая архитектура на трансформерах (TinyViT-21M). Сравнение показало, что тяжёлые модели действительно могут давать более высокую точность, но требуют существенно больше вычислений и ресурсов. При этом EfficientNet-B0 продемонстрировала лучший баланс: по качеству она заметно превосходила более простые варианты при умеренном размере и вычислительной нагрузке. Для задачи преддипломной практики этот компромисс оказался оптимальным: модель остаётся достаточно компактной для обучения, экспериментов и встраивания в десктопное приложение, но при этом обеспечивает хороший уровень точности и устойчивости.

3.2 Реализация нейронной сети

3.2.1 Первое поколение

Обучение организовано через TensorFlow с использованием конвейера tf.data, так как разметка задаётся таблицами train.csv и val.csv, а не структурой папок вида класс/изображения. На этапе подготовки данных путь к файлу считывается из CSV, затем изображение загружается функцией tf.io.read_file и декодируется универсальным декодером tf.image.decode_image, который позволяет работать с форматами JPG/PNG/BMP. После декодирования изображение приводится к типу float32 и нормируется в диапазон [0; 1], а далее масштабируется до фиксированного размера 384×512 пикселей. Такой размер был выбран как компромисс между сохранением деталей дефектов (в первую очередь нерезкости) и ограничениями по ресурсам видеопамяти при обучении. Для обучающей выборки дополнительно применяется перемешивание, затем данные группируются в пакеты размером 16 изображений и подаются на обучение с предварительной подгрузкой, что уменьшает простои вычислений из-за ввода-вывода.

Выход основы (EfficientNet-B0) представляет собой пространственную карту признаков, которая преобразуется в вектор с помощью глобального усредняющего пуллинга (GlobalAveragePooling2D). Далее применяется Dropout с вероятностью 0.2: этот слой на обучении случайно «отключает» часть нейронов и тем самым снижает риск переобучения. После этого реализованы три независимые «головы» - по одному полносвязному слою на каждый дефект. Важно, что выходы голов являются логитами: логит - это значение до применения сигмоиды, а вероятность дефекта получается применением функции sigmoid(logit).

В качестве функции потерь используется бинарная кросс-энтропия с параметром from_logits=True. Это означает, что функция потерь сама применяет сигмоиду внутри себя и корректно работает с логитами. Для контроля качества в процессе обучения применяется метрика AUC, причём в многометочном варианте. Бинарной кросс-энтропии рассчитывается по формуле (1).

$$L(y, p) = -[y \log(p) + (1 - y) \log(1 - p)], \quad (1)$$

где $L(y, p)$ - значение функции потерь,

y - истинная метка ($yin\{0, 1\}$),

p - предсказанная моделью вероятность класса 1 ($0 < p < 1$).

Обучение первого поколения проводилось в два этапа. На первом этапе обучались только три выходные головы, а основа EfficientNetB0 была заморожена (backbone.trainable=False). Скорость обучения на этом этапе была выбрана 10^(-3). На втором этапе основа размораживалась полностью (backbone.trainable=True) и выполнялось дообучение всей сети с уменьшенной скоростью обучения 110^(-4). Уменьшение шага здесь принципиально: дообучение предобученной основы требует более аккуратных обновлений, иначе можно быстро получить деградацию признаков и нестабильность обучения. Размер пакета 16 выбран исходя из практического баланса: он достаточно велик для приемлемой стабильности градиентов, но при размере входа 384×512 не приводит к переполнению видеопамяти на видеокарте «домашнего» компьютера.

Динамика обучения показывает, что выбранная схема действительно даёт прирост качества. По итогам первого этапа (5-я эпоха обучения только голов) обучающая AUC достигла 0.6063 при значении функции потерь 0.4276, а на проверочной выборке AUC составила 0.7151 при val_loss = 0.4325. После перехода ко второму этапу и дообучения основы качество на проверке выросло существенно выше. В конце обучения достигнуты значения auc = 0.8048, loss = 0.3496, val_auc = 0.8627, val_loss = 0.3205. Таким образом, именно после дообучения основы модель значительно лучше отделяет дефектные изображения от качественных по всем трём меткам. Конкретные значения loss и auc в зависимости от эпох прежставлены на рисунке 2.

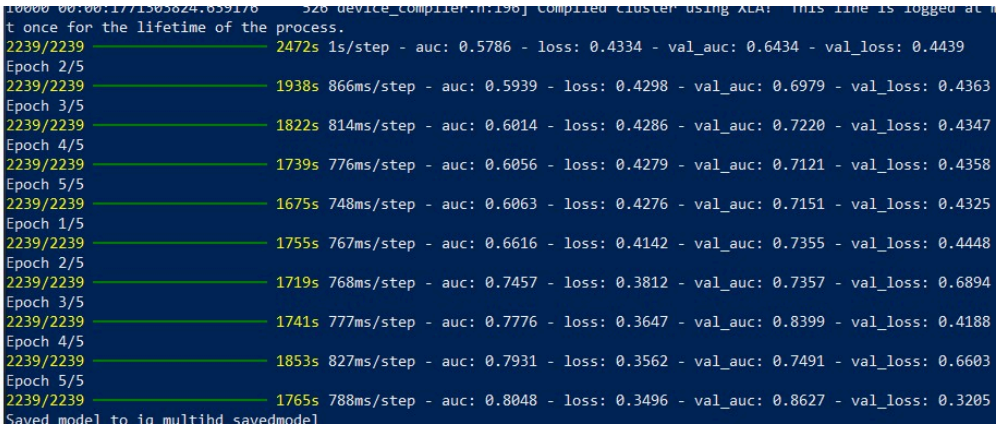

Рисунок 2 — Значения loss и AUC первого поколения

На графиках обучения заметна характерная проблема: проверочная функция потерь val_loss ведёт себя нестабильно и даёт резкие всплески на последних 5 эпохах, несмотря на общий рост AUC. Это поведение связано с тем, что при полном размораживании основы начинают активно обновляться слои пакетной нормализации BatchNorm. BatchNorm (пакетная нормализация) использует статистики текущего пакета и поддерживает скользящие оценки среднего и дисперсии, которые затем влияют на поведение модели на проверке и при реальном использовании. При небольшом размере пакета (в данном случае 16) оценки статистик получаются шумными, а их обновление может приводить к тому, что модель подстраивается под особенности обучающих пакетов и временно ухудшается на проверке. Визуально это проявляется скачками val_loss и колебаниями val_auc. В дальнейшем эта проблема была решена заморозкой BatchNorm на этапе дообучения основы. Графики обучения представлены на рисунке 3.

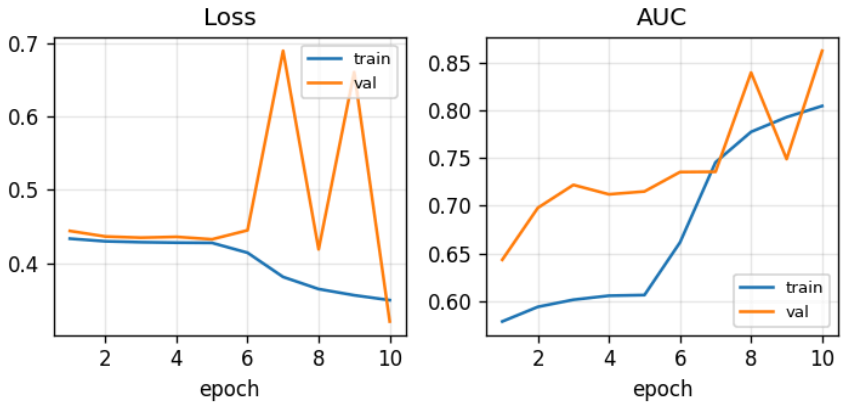

Рисунок 3 — Графики ключевых метрик первого поколения

Дополнительно качество было оценено отдельным скриптом на проверочной выборке в терминах точности (т.е. в более понятных человеку процентах угадывания) при фиксированном пороге решения. Получены значения: общая точность по всем меткам составила 0.8333, отдельно по дефектам - 0.7492 для blur, 0.8725 для under и 0.8784 для over. Результаты проверки точности представлены на рисунке 4.

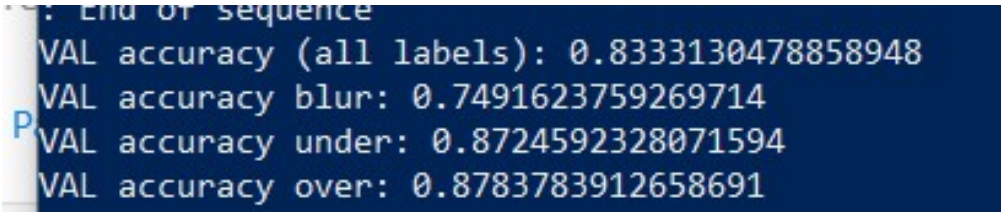

Рисунок 4 — Точность правильных предсказаний первого поколения

В целом первое поколение подтвердило работоспособность выбранного направления: трёхголовая архитектура на EfficientNetB0 обучается на синтетических искажениях и демонстрирует высокую способность различать дефекты, а двухэтапная схема обучения существенно улучшает качество по сравнению с обучением только выходных слоёв. Одновременно были выявлены практические ограничения, связанные с нестабильностью проверки при дообучении основы из-за BatchNorm, что стало основанием для последующих улучшений во втором поколении модели. Программный код первого поколения представлен в приложении А.

3.2.2 Второе поколение

Во втором (финальном) поколении нейронной сети необходимо доработать первое поколение так, чтобы убрать нестабильность на проверке, которую давали слои пакетной нормализации; усилить распознавание нерезкости как самой слабой части модели и сделать контроль обучения более прозрачным за счёт расширенного набора метрик и автоматического построения графиков.

В первом поколении каждая голова представляла собой 1 нейрон. По результатам первой проверки именно нерезкость распознавалась хуже остальных: точность по blur на проверке была заметно ниже (около 0.75), тогда как under/over были около 0.87–0.88. Это объяснимо: нерезкость проявляется в тонких деталях (высоких частотах), и для её отделения от требуется

более сложная нелинейная комбинация признаков, чем для экспозиционных дефектов, которые сильнее зависят от распределения яркостей.

Поэтому во втором для головы, отвечающей за размытие, добавлена цепочка слоёв 64 -> 32 -> 16 -> 1 с ReLU-активацией и Dropout 0.15. Для under/over усиление более умеренное: 32 -> 16 -> 1 и Dropout 0.1. Такой перекос по мощности сделан осознанно: вычислительная цена растёт незначительно (эти слои стоят после глобального пуллинга и работают с вектором признаков), но модель получает возможность построить более сложную границу решения именно для blur - там, где первое поколение было слабее.

Вторая важная доработка касается устойчивости дообучения основы. В первом поколении при размораживании EfficientNet график функции потерь заметно «скакал», что связано с BatchNorm. Во втором поколении эта проблема решена двумя мерами одновременно. Во-первых, на этапе тонкой настройки BatchNorm-слои принудительно замораживаются (их параметры и статистики перестают обновляться). Во-вторых, размораживается не вся основа, а только её верхняя часть: все слои, кроме последних примерно 40, оставляются замороженными. Тем самым сохраняются универсальные низкоуровневые признаки (контуры, текстуры), а адаптация идёт за счёт верхних слоёв, которые отвечают за более сложные комбинации признаков. Дополнительно уменьшается скорость обучения на втором этапе до 5*10^(-5), чтобы изменения весов были плавными и не приводили к деградации на проверке. В результате проверочные кривые становятся заметно более стабильными: на итоговом графике нет резких провалов val_loss, а к концу обучения значения сходятся к небольшим величинам. Графики обучения представлены на рисунке 5.

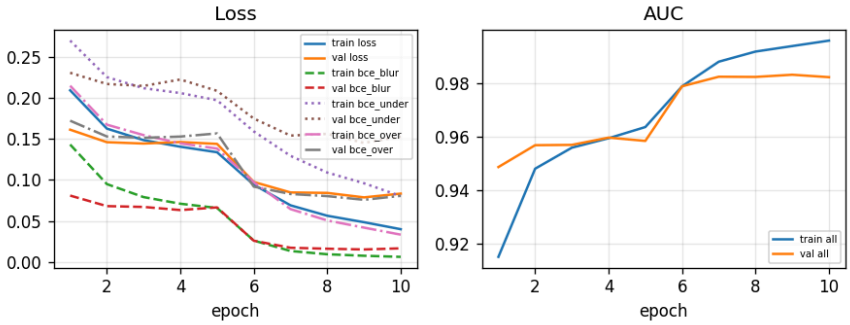

Рисунок 5 — Графики ключевых метрик второго поколения

Третья группа изменений - расширение набора метрик и улучшение журналирования. В первом поколении была только общая AUC, и по ней сложно понять, какая из голов учится хуже и почему. В конце концов скрипт по точности предсказаний дал понять, что самый слабый выход - выход по резкости, но такая проверка на выходе дает слабое представление о том, как каждая из голов обучалась. Во втором поколении добавлены метрики бинарной кросс-энтропии, посчитанной отдельно по каждому выходу. Новые графики также можно наблюдать на рисунке 5.

Финальные числа показывают, что доработки дали существенный прирост и по общим метрикам, и по каждой голове. В конце обучения получены значения: AUC на обучении 0.9959, AUC на проверке 0.9822, при этом проверочные потери составили val_loss = 0.0833. Финальные метрики представлены на рисунке 6.

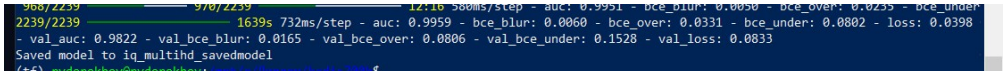

Рисунок 6 — Значения loss и AUC второго поколения

Отдельный скрипт оценки на проверочной выборке показывает прикладные метрики качества при порогах 0.5 для всех голов. Общая точность по всем меткам (micro-accuracy) составила 0.9713, а точность строгого совпадения трёх меток одновременно (exact-match accuracy - доля изображений, где верно угаданы сразу все три метки) составила 0.9179. По дефектам получены следующие показатели. Для blur: precision 0.9946, recall 0.9866, F1 0.9906, accuracy 0.9953 (precision - доля правильных среди всех срабатываний, recall - доля найденных дефектов среди всех реальных дефектов). Для under: precision 0.7709, recall 0.7986, F1 0.7845, accuracy 0.9440 - видно, что именно under даёт основную часть ложных срабатываний (FP=271), поэтому его точность ниже. Для over: precision 0.9241, recall 0.8604, F1 0.8911, accuracy 0.9744, то есть пересвет распознаётся уверенно, но чуть хуже blur. Результаты проверки точности представлены на рисунке 7.

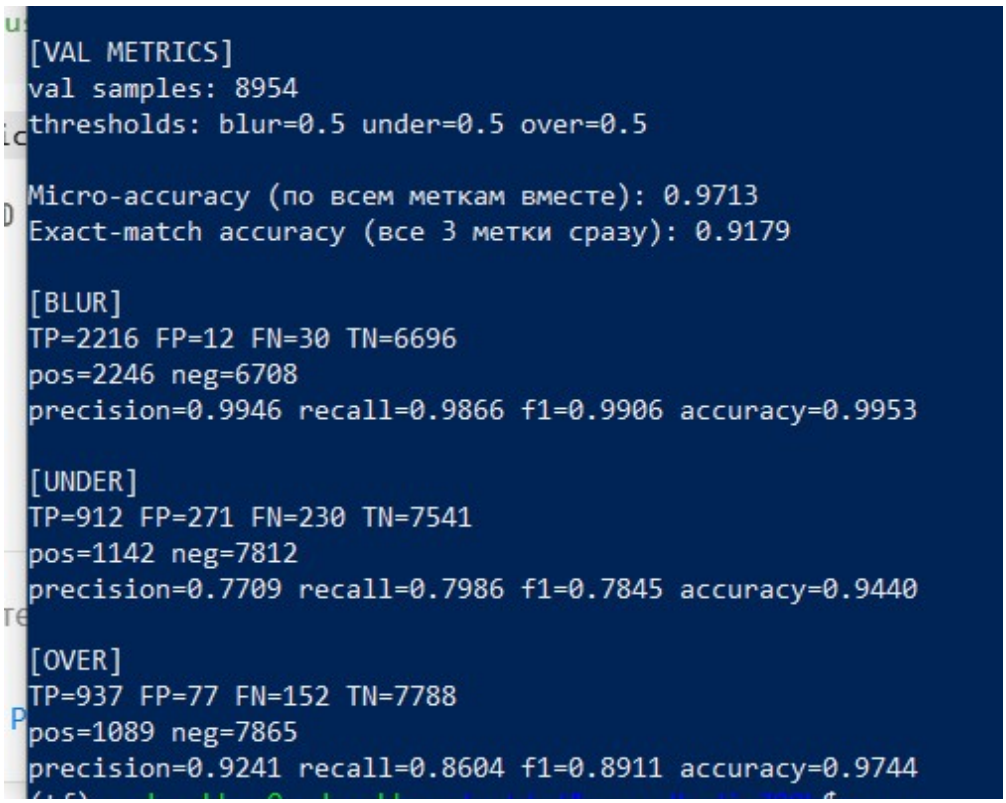

Рисунок 7 — Точность предсказаний второго поколения

С учётом полученных значений дальнейшее обучение в рамках практики не выглядит необходимым. Во-первых, модель уже достигла высокого уровня разделения классов на проверке (val_auc 0.9822) и высокой практической точности по всем меткам (micro-accuracy 0.9713), а графики показывают выход метрик на плато без признаков систематического улучшения к концу 10-й эпохи. Во-вторых, после стабилизации BatchNorm и частичного дообучения основы дальнейшие эпохи с большой вероятностью дадут убывающую отдачу и увеличат риск переобучения под синтетические искажения. В-третьих, наиболее действенный способ повышения качества теперь лежит не в увеличении числа эпох, а в настройке порогов по головам (особенно для under, чтобы снизить ложные срабатывания).

Отдельно стоит обсудить метрику under. У метки under много ложноположительных срабатываний (FP=271), особено в сравнении с другими выходами. Поэтому её точность ниже (precision 0.7709 при accuracy 0.9440). Чаще всего это ночные сцены: кадр намеренно тёмный, но модель, обученная на синтетическом затемнении, воспринимает это как недосвет. Чтобы не браковать ночные фото, в приложении будет предусмотрено отключение

фильтрации по under (или перевод в режим подсказки). Повышать under/over только дальнейшим обучением на текущей синтетике неэффективно из-за доменного разрыва и отсутствия контекстной разметки «ночь — норма». Для over эта проблема тоже прослеживается, но не так явно, потому что «естественные» затемнения в реальном мире встречаются намного чаще сильных пересветов. Есть смысл дорабатывать поведение программно уже в прикладном приложении: прежде всего порогами или пользовательской настройкой.

Таким образом, второе поколение можно считать финальным для этапа преддипломной практики: устранена нестабильность проверки, усилена самая слабая голова (blur), добавлен детальный контроль обучения по метрикам, а итоговые числа подтверждают, что модель уже подходит для встраивания в дипломное приложение.

Программный код второго поколения представлен в приложении А.

ЗАКЛЮЧЕНИЕ

В ходе преддипломной практики была разработана нейросетевая подсистема для дипломного приложения-редактора, которая автоматически оценивает техническое качество фотографий по трём дефектам: размытие (blur), недосвет (under) и пересвет (over). Подготовлен датасет на основе KADIS-700K с нужными типами искажений, сформирована разметка labels.csv, выполнено разбиение train/val без утечек по сценам (по ref) и реализован конвейер обучения в TensorFlow через tf.data.

Построена модель EfficientNetB0 с тремя выходными головами и двухэтапным обучением; во втором поколении устранена нестабильность проверки (заморозка BatchNorm и частичное дообучение основы), усилена голова blur и расширен набор метрик для контроля качества по каждой метке. Итоговые результаты на проверке: val_auc = 0.9822, micro-accuracy = 0.9713, exact-match accuracy = 0.9179; наилучшее качество достигнуто по blur (accuracy 0.9953, F1 0.9906) и over (accuracy 0.9744, F1 0.8911). Для under выявлено много ложных срабатываний на ночных сценах (FP=271). В целом полученного качества достаточно для интеграции модели в дипломное приложение.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Мартынюк П.А. Лекции по Искусственному Интеллекту.

2. KADID-10k IQA Database [Электронный ресурс]. URL: https://database.mmsp-kn.de/kadid-10k-database.html (дата обращения: 20.09.2025).

3. Документация по MATLAB [Электронный ресурс]. URL: https://www.mathworks.com/help/matlab/index.html (дата обращения: 26.09.2025).

4. Использование моделей EfficientNet для классификации изображений [Электронный ресурс]. URL: https://habr.com/ru/companies/sberbank/articles/828842/ (дата обращения: 14.10.2025).

5. Документация по TensorFlow [Электронный ресурс]. URL: https://www.tensorflow.org/guide?hl=ru (дата обращения: 16.10.2025).

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ ТЕКСТ ПРОГРАММЫ

Листов 18

Листинг А.1 — Содержимое файла generate_kadis700k.m

```
%% setup
clear; clc;
addpath(genpath('code_imdistort'));

fid = fopen('labels.csv','w');
fprintf(fid,
'path,blur,under,over,dist_type,dist_level,ref\n');

%% read the info of pristine images

tb = readtable('kadis700k_ref_imgs.csv');

% --- ОСТАВЛЯЕМ ТОЛЬКО НУЖНЫЕ ИСКАЖЕНИЯ ---
types = [2 3 16 17];            % gaussian/lens/motion blur +
brighten + darken
tb = tb(ismember(tb{:,2}, types), :);  % предполагаем, что 2-й
столбец = dist_type

tb = table2cell(tb);


%% labels for ref

ref_list = unique(tb(:,1));  % это cell array с именами файлов

for r = 1:numel(ref_list)
    ref_name = ref_list{r};
    ref_path = fullfile('ref_imgs', ref_name);

    % pristine: дефектов нет
    blur = 0; under = 0; over = 0;
    dist_type = 0; dist_level = 0;

    % ref в конце можно тоже писать ref_name (для pristine это
одно и то же)
    fprintf(fid, '%s,%d,%d,%d,%d,%d,%s\n', ref_path, blur,
under, over, dist_type, dist_level, ref_name);
end


%% generate distorted images in dist_imgs folder

for i = 1:size(tb,1)
    ref_im = imread(fullfile('ref_imgs', tb{i,1}));
    dist_type = tb{i,2};
```

Продолжение листинга А.1

```
    if dist_type == 2
        allowed_levels = [2 3 4];
    elseif dist_type == 3
        allowed_levels = [4 5];
    elseif dist_type == 16
        allowed_levels = [4 5];
    elseif dist_type == 17
        allowed_levels = [4 5];
    else
        continue;
    end

  %%  for k = 1:numel(keep_levels)
        dist_level =
allowed_levels(randi(numel(allowed_levels)));

        dist_im = imdist_generator(ref_im, dist_type,
dist_level);

        % имя файла остаётся совместимым: _тип_уровень.bmp
        strs = split(tb{i,1},'.');

        dist_im_name = [strs{1} '_' num2str(dist_type,'%02d')
'_' num2str(dist_level,'%02d') '.jpg'];
        out_path = fullfile('dist_imgs', dist_im_name);

        if exist(out_path, 'file')
            continue;
        end

        imwrite(dist_im, out_path, 'Quality', 95);

        % вычисляем метки по dist_type
        blur = 0; under = 0; over = 0;
        if dist_type == 2 || dist_type == 3
            blur = 1;
        elseif dist_type == 17
            under = 1;
        elseif dist_type == 16
            over = 1;
        end

        fprintf(fid, '%s,%d,%d,%d,%d,%d,%s\n', out_path, blur,
under, over, dist_type, dist_level, tb{i,1});

 %%   end
```

Продолжение листинга А.1

```
end

fclose(fid);
```

Листинг А.2 — Содержимое файла dataset.py

```
import os
import pandas as pd
from sklearn.model_selection import train_test_split

LABELS_PATH = "labels.csv"
TRAIN_OUT = "train.csv"
VAL_OUT = "val.csv"
VAL_RATIO = 0.2
SEED = 42

df = pd.read_csv(LABELS_PATH)

df["path"] = df["path"].astype(str).str.replace("\\",
os.sep).str.replace("/", os.sep)

# Сплит по ref (чтобы не было утечки)
refs = df["ref"].astype(str).unique()
train_refs, val_refs = train_test_split(refs,
test_size=VAL_RATIO, random_state=SEED, shuffle=True)

train_df =
df[df["ref"].astype(str).isin(train_refs)].reset_index(drop=True)
val_df   =
df[df["ref"].astype(str).isin(val_refs)].reset_index(drop=True)

train_df.to_csv(TRAIN_OUT, index=False)
val_df.to_csv(VAL_OUT, index=False)

print("Saved:", TRAIN_OUT, len(train_df))
print("Saved:", VAL_OUT, len(val_df))
print("Unique refs train/val:", len(train_refs),
len(val_refs))
```

Листинг А.3 — Содержимое файла ML_first_generation.py

```
import tensorflow as tf
import pandas as pd
import os
import matplotlib.pyplot as plt
```

Продолжение листинга А.3

```
IMG_H, IMG_W = 384, 512  # tf: (H,W)
BATCH = 16
EPOCHS_HEADS = 5
EPOCHS_FT = 5

TRAIN_CSV = "train.csv"
VAL_CSV = "val.csv"

AUTOTUNE = tf.data.AUTOTUNE


class PlotCurvesCallback(tf.keras.callbacks.Callback):
    """
    Рисует:
      - общий loss (train/val)
      - loss по головам (blur/under/over, train/val)  <--
ДОБАВИЛИ
      - AUC общий + по головам (train/val)

    Память почти не расходует: храним только списки длиной =
числу эпох.
    """
    def __init__(self, out_path="training_curves.png"):
        super().__init__()
        self.out_path = out_path

        # Ось X: глобальные эпохи (не сбрасываются между двумя
fit())
        self.global_epoch = 0
        self.epochs = []

        # ---- Общий loss ----
        self.loss = []
        self.val_loss = []

        # ---- Loss по головам (BCE как метрика) ----
        self.bce_blur = []
        self.val_bce_blur = []
        self.bce_under = []
        self.val_bce_under = []
        self.bce_over = []
        self.val_bce_over = []

        # ---- AUC общий ----
        self.auc = []
```

Продолжение листинга А.3

```
        self.val_auc = []

    def on_epoch_end(self, epoch, logs=None):
        logs = logs or {}

        # Глобальная эпоха: 1..N по всем fit()
        self.global_epoch += 1
        self.epochs.append(self.global_epoch)

        # ---- Общий loss ----
        self.loss.append(logs.get("loss"))
        self.val_loss.append(logs.get("val_loss"))

        # ---- Loss по головам ----
        self.bce_blur.append(logs.get("bce_blur"))
        self.val_bce_blur.append(logs.get("val_bce_blur"))

        self.bce_under.append(logs.get("bce_under"))
        self.val_bce_under.append(logs.get("val_bce_under"))

        self.bce_over.append(logs.get("bce_over"))
        self.val_bce_over.append(logs.get("val_bce_over"))

        # ---- AUC общий ----
        self.auc.append(logs.get("auc"))
        self.val_auc.append(logs.get("val_auc"))

        # ---- Рисуем и перезаписываем файл ----
        fig = plt.figure(figsize=(7.5, 3), dpi=120)

        # ===== Loss (общий + по головам) =====
        ax1 = fig.add_subplot(1, 2, 1)

        # общий loss
        ax1.plot(self.epochs, self.loss, label="train loss")
        ax1.plot(self.epochs, self.val_loss, label="val loss")

        # head losses (если они есть в logs; иначе будут None
-> matplotlib может ругаться)
        # поэтому рисуем только если хотя бы одно значение не
None
        if any(v is not None for v in self.bce_blur):
            ax1.plot(self.epochs, self.bce_blur, label="train
bce_blur", linestyle="--")
            ax1.plot(self.epochs, self.val_bce_blur,
label="val bce_blur", linestyle="--")
```

Продолжение листинга А.3

```
        if any(v is not None for v in self.bce_under):
            ax1.plot(self.epochs, self.bce_under, label="train
bce_under", linestyle=":")
            ax1.plot(self.epochs, self.val_bce_under,
label="val bce_under", linestyle=":")

        if any(v is not None for v in self.bce_over):
            ax1.plot(self.epochs, self.bce_over, label="train
bce_over", linestyle="-.")
            ax1.plot(self.epochs, self.val_bce_over,
label="val bce_over", linestyle="-.")

        ax1.set_title("Loss")
        ax1.set_xlabel("epoch")
        ax1.grid(True, alpha=0.3)
        ax1.legend(loc="upper right", fontsize=6)

        # ===== AUC (общий + по головам) =====
        ax2 = fig.add_subplot(1, 2, 2)

        ax2.plot(self.epochs, self.auc, label="train all")
        ax2.plot(self.epochs, self.val_auc, label="val all")

        ax2.set_title("AUC")
        ax2.set_xlabel("epoch")
        ax2.grid(True, alpha=0.3)
        ax2.legend(loc="lower right", fontsize=6)

        fig.tight_layout()
        fig.savefig(self.out_path)
        plt.close(fig)  # важно: закрываем фигуру, чтобы не
копить память




def decode_and_resize(path):
    img_bytes = tf.io.read_file(path)
    img = tf.image.decode_image(img_bytes, channels=3,
expand_animations=False)  # jpg/png/bmp ок
    img = tf.cast(img, tf.float32)    # [0..255]
    img = tf.image.resize(img, (IMG_H, IMG_W),
method="bilinear")
    return img
```

Продолжение листинга А.3

```
def make_ds(csv_path, shuffle=False):
    df = pd.read_csv(csv_path)
    paths = df["path"].astype(str).tolist()
    labels = df[["blur", "under",
"over"]].astype("float32").values

    ds = tf.data.Dataset.from_tensor_slices((paths, labels))

    if shuffle:
        ds = ds.shuffle(buffer_size=min(len(paths), 20000),
reshuffle_each_iteration=True)

    def _map_fn(path, y):
        img = decode_and_resize(path)
        return img, y

    ds = ds.map(_map_fn, num_parallel_calls=AUTOTUNE)
    ds = ds.batch(BATCH).prefetch(AUTOTUNE)
    return ds

def build_model():
    inp = tf.keras.Input(shape=(IMG_H, IMG_W, 3))

    backbone = tf.keras.applications.EfficientNetB0(
        include_top=False, weights="imagenet",
input_tensor=inp
    )
    x = backbone.output
    x = tf.keras.layers.GlobalAveragePooling2D()(x)
    x = tf.keras.layers.Dropout(0.2)(x)

    def head(name: str, x):
        # маленький скрытый слой для конкретной головы
        h = tf.keras.layers.Dense(16, activation="relu",
name=f"{name}_fc")(x)
        # финальный нейрон (логит)
        out = tf.keras.layers.Dense(1, name=f"logit_{name}")
(h)
        return out

    # 3 головы -> логиты (без sigmoid)
    logit_blur  = head("blur", x)
    logit_under = head("under", x)
    logit_over  = head("over", x)

    logits = tf.keras.layers.Concatenate(axis=1,
```

Продолжение листинга А.3

```
name="logits")([logit_blur, logit_under, logit_over])

    model = tf.keras.Model(inputs=inp, outputs=logits,
name="iq_multihd")
    return model, backbone

# Метрика: AUC по трём лейблам (работает с вероятностями,
поэтому применим sigmoid внутри)
class MultiLabelAUC(tf.keras.metrics.Metric):
    def __init__(self, name="auc", **kwargs):
        super().__init__(name=name, **kwargs)
        self.auc = tf.keras.metrics.AUC(multi_label=True,
num_labels=3)

    def update_state(self, y_true, y_pred,
sample_weight=None):
        y_prob = tf.sigmoid(y_pred)
        self.auc.update_state(y_true, y_prob,
sample_weight=sample_weight)

    def result(self):
        return self.auc.result()

    def reset_state(self):
        self.auc.reset_state()

class PerLabelBCE(tf.keras.metrics.Metric):
    """BCE (from_logits=True) по одному столбцу: 0=blur,
1=under, 2=over"""
    def __init__(self, label_index: int, name: str, **kwargs):
        super().__init__(name=name, **kwargs)
        self.label_index = label_index
        self.mean = tf.keras.metrics.Mean()

    def update_state(self, y_true, y_pred,
sample_weight=None):
        yt = tf.cast(y_true[:, self.label_index], tf.float32)
        lg = y_pred[:, self.label_index]        # (B,)

        # BCE для логитов: -[ y*log(sigmoid(l)) + (1-y)*log(1-
sigmoid(l)) ]
        loss =
tf.nn.sigmoid_cross_entropy_with_logits(labels=yt, logits=lg)
# (B,)

        self.mean.update_state(loss,
```

Продолжение листинга А.3

```
sample_weight=sample_weight)

    def result(self):
        return self.mean.result()

    def reset_state(self):
        self.mean.reset_state()

def compile_model(model, lr):
    model.compile(
        optimizer=tf.keras.optimizers.Adam(lr),
loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
        metrics=[
            MultiLabelAUC(name="auc"),

            # <<< loss по головам (как метрики)
            PerLabelBCE(0, name="bce_blur"),
            PerLabelBCE(1, name="bce_under"),
            PerLabelBCE(2, name="bce_over"),
        ],
    )


def main():
    physical_devices =
tf.config.experimental.list_physical_devices("GPU")
    print(tf.config.list_physical_devices())
    print("Num GPUs Available: ", len(physical_devices))
tf.config.experimental.set_memory_growth(physical_devices[0],
True)

    train_ds = make_ds(TRAIN_CSV, shuffle=True)
    val_ds = make_ds(VAL_CSV, shuffle=False)

    # 1) формы
    x_batch, y_batch = next(iter(train_ds.take(1)))
    print("\n[DEBUG] Train batch shapes:")
    print("  x:", x_batch.shape, x_batch.dtype)
    print("  y:", y_batch.shape, y_batch.dtype)

    # 2) диапазон значений пикселей (после decode +
convert_image_dtype)
    x_min = tf.reduce_min(x_batch).numpy()
    x_max = tf.reduce_max(x_batch).numpy()
    print("[DEBUG] Pixel range after preprocessing: min =",
```

Продолжение листинга А.3

```
x_min, "max =", x_max)

    model, backbone = build_model()

    print("\n[DEBUG] Model summary:")
    model.summary()

    plot_cb = PlotCurvesCallback("training_curves.png")
    csv_cb = tf.keras.callbacks.CSVLogger("train_log.csv",
append=True)
    callbacks = [plot_cb, csv_cb]

    # Этап 1: учим только головы
    backbone.trainable = False
    compile_model(model, lr=1e-3)
    model.fit(train_ds, validation_data=val_ds,
epochs=EPOCHS_HEADS, callbacks=callbacks)

    # Этап 2: fine-tune (можно разморозить всё или только
верхние слои)
    # Часто стабильнее размораживать не всё сразу, а только
верх:
    backbone.trainable = True

    # заморозить BatchNorm
    for layer in backbone.layers:
        if isinstance(layer,
tf.keras.layers.BatchNormalization):
            layer.trainable = False

    # заморозить нижнюю часть
    for layer in backbone.layers[:-40]:
        layer.trainable = False


    compile_model(model, lr=5e-5)
    model.fit(train_ds, validation_data=val_ds,
epochs=EPOCHS_FT, callbacks=callbacks)

    model.save("iq_multihd_savedmodel.keras")
    print("Saved model to iq_multihd_savedmodel")

if __name__ == "__main__":
    main()
```

Листинг А.4 — Содержимое файла ML_second_generation.py

```
import tensorflow as tf
import pandas as pd
import os
import matplotlib.pyplot as plt

IMG_H, IMG_W = 384, 512  # tf: (H,W)
BATCH = 16
EPOCHS_HEADS = 5
EPOCHS_FT = 5

TRAIN_CSV = "train.csv"
VAL_CSV = "val.csv"

AUTOTUNE = tf.data.AUTOTUNE


class PlotCurvesCallback(tf.keras.callbacks.Callback):
    """
    Рисует:
      - общий loss (train/val)
      - loss по головам (blur/under/over, train/val)  <-- ДОБАВИЛИ
      - AUC общий + по головам (train/val)

    Память почти не расходует: храним только списки длиной = числу эпох.
    """
    def __init__(self, out_path="training_curves.png"):
        super().__init__()
        self.out_path = out_path

        # Ось X: глобальные эпохи (не сбрасываются между двумя fit())
        self.global_epoch = 0
        self.epochs = []

        # ---- Общий loss ----
        self.loss = []
        self.val_loss = []

        # ---- Loss по головам (BCE как метрика) ----
        self.bce_blur = []
        self.val_bce_blur = []
        self.bce_under = []
        self.val_bce_under = []
        self.bce_over = []
```

Продолжение листинга А.4

```
        self.val_bce_over = []

        # ---- AUC общий ----
        self.auc = []
        self.val_auc = []

    def on_epoch_end(self, epoch, logs=None):
        logs = logs or {}

        # Глобальная эпоха: 1..N по всем fit()
        self.global_epoch += 1
        self.epochs.append(self.global_epoch)

        # ---- Общий loss ----
        self.loss.append(logs.get("loss"))
        self.val_loss.append(logs.get("val_loss"))

        # ---- Loss по головам ----
        self.bce_blur.append(logs.get("bce_blur"))
        self.val_bce_blur.append(logs.get("val_bce_blur"))

        self.bce_under.append(logs.get("bce_under"))
        self.val_bce_under.append(logs.get("val_bce_under"))

        self.bce_over.append(logs.get("bce_over"))
        self.val_bce_over.append(logs.get("val_bce_over"))

        # ---- AUC общий ----
        self.auc.append(logs.get("auc"))
        self.val_auc.append(logs.get("val_auc"))

        # ---- Рисуем и перезаписываем файл ----
        fig = plt.figure(figsize=(7.5, 3), dpi=120)

        # ===== Loss (общий + по головам) =====
        ax1 = fig.add_subplot(1, 2, 1)

        # общий loss
        ax1.plot(self.epochs, self.loss, label="train loss")
        ax1.plot(self.epochs, self.val_loss, label="val loss")

        # head losses (если они есть в logs; иначе будут None
-> matplotlib может ругаться)
        # поэтому рисуем только если хотя бы одно значение не
None
        if any(v is not None for v in self.bce_blur):
```

Продолжение листинга А.4

```
            ax1.plot(self.epochs, self.bce_blur, label="train
bce_blur", linestyle="--")
            ax1.plot(self.epochs, self.val_bce_blur,
label="val bce_blur", linestyle="--")

        if any(v is not None for v in self.bce_under):
            ax1.plot(self.epochs, self.bce_under, label="train
bce_under", linestyle=":")
            ax1.plot(self.epochs, self.val_bce_under,
label="val bce_under", linestyle=":")

        if any(v is not None for v in self.bce_over):
            ax1.plot(self.epochs, self.bce_over, label="train
bce_over", linestyle="-.")
            ax1.plot(self.epochs, self.val_bce_over,
label="val bce_over", linestyle="-.")

        ax1.set_title("Loss")
        ax1.set_xlabel("epoch")
        ax1.grid(True, alpha=0.3)
        ax1.legend(loc="upper right", fontsize=6)

        # ===== AUC (общий + по головам) =====
        ax2 = fig.add_subplot(1, 2, 2)

        ax2.plot(self.epochs, self.auc, label="train all")
        ax2.plot(self.epochs, self.val_auc, label="val all")

        ax2.set_title("AUC")
        ax2.set_xlabel("epoch")
        ax2.grid(True, alpha=0.3)
        ax2.legend(loc="lower right", fontsize=6)

        fig.tight_layout()
        fig.savefig(self.out_path)
        plt.close(fig)  # важно: закрываем фигуру, чтобы не
копить память




def decode_and_resize(path):
    img_bytes = tf.io.read_file(path)
    img = tf.image.decode_image(img_bytes, channels=3,
expand_animations=False)  # jpg/png/bmp ок
    img = tf.cast(img, tf.float32)    # [0..255]
```

Продолжение листинга А.4

```
    img = tf.image.resize(img, (IMG_H, IMG_W),
method="bilinear")
    return img

def make_ds(csv_path, shuffle=False):
    df = pd.read_csv(csv_path)
    paths = df["path"].astype(str).tolist()
    labels = df[["blur", "under",
"over"]].astype("float32").values

    ds = tf.data.Dataset.from_tensor_slices((paths, labels))

    if shuffle:
        ds = ds.shuffle(buffer_size=min(len(paths), 20000),
reshuffle_each_iteration=True)

    def _map_fn(path, y):
        img = decode_and_resize(path)
        return img, y

    ds = ds.map(_map_fn, num_parallel_calls=AUTOTUNE)
    ds = ds.batch(BATCH).prefetch(AUTOTUNE)
    return ds

def build_model():
    inp = tf.keras.Input(shape=(IMG_H, IMG_W, 3))

    backbone = tf.keras.applications.EfficientNetB0(
        include_top=False, weights="imagenet",
input_tensor=inp
    )

    x = backbone.output
    x = tf.keras.layers.GlobalAveragePooling2D()(x)
    x = tf.keras.layers.Dropout(0.2, name="neck_dropout")(x)

    def head_exposure(name: str, x):
        # under/over: 32 -> 16 -> 1
        h = tf.keras.layers.Dense(32, activation="relu",
name=f"{name}_fc1")(x)
        h = tf.keras.layers.Dropout(0.1, name=f"{name}_drop1")
(h)
        h = tf.keras.layers.Dense(16, activation="relu",
name=f"{name}_fc2")(h)
        out = tf.keras.layers.Dense(1, name=f"logit_{name}")
(h)
```

Продолжение листинга А.4

```
        return out

    def head_blur(name: str, x):
        # blur: 64 -> 32 -> 16 -> 1
        h = tf.keras.layers.Dense(64, activation="relu",
name=f"{name}_fc1")(x)
        h = tf.keras.layers.Dropout(0.15, name=f"{name}
_drop1")(h)
        h = tf.keras.layers.Dense(32, activation="relu",
name=f"{name}_fc2")(h)
        h = tf.keras.layers.Dense(16, activation="relu",
name=f"{name}_fc3")(h)
        out = tf.keras.layers.Dense(1, name=f"logit_{name}")
(h)
        return out

    logit_blur  = head_blur("blur", x)
    logit_under = head_exposure("under", x)
    logit_over  = head_exposure("over", x)

    logits = tf.keras.layers.Concatenate(axis=1,
name="logits")(
        [logit_blur, logit_under, logit_over]
    )

    model = tf.keras.Model(inputs=inp, outputs=logits,
name="iq_multihd")
    return model, backbone


# Метрика: AUC по трём лейблам (работает с вероятностями,
поэтому применим sigmoid внутри)
class MultiLabelAUC(tf.keras.metrics.Metric):
    def __init__(self, name="auc", **kwargs):
        super().__init__(name=name, **kwargs)
        self.auc = tf.keras.metrics.AUC(multi_label=True,
num_labels=3)

    def update_state(self, y_true, y_pred,
sample_weight=None):
        y_prob = tf.sigmoid(y_pred)
        self.auc.update_state(y_true, y_prob,
sample_weight=sample_weight)

    def result(self):
        return self.auc.result()
```

Продолжение листинга А.4

```
    def reset_state(self):
        self.auc.reset_state()

class PerLabelBCE(tf.keras.metrics.Metric):
    """BCE (from_logits=True) по одному столбцу: 0=blur,
1=under, 2=over"""
    def __init__(self, label_index: int, name: str, **kwargs):
        super().__init__(name=name, **kwargs)
        self.label_index = label_index
        self.mean = tf.keras.metrics.Mean()

    def update_state(self, y_true, y_pred,
sample_weight=None):
        yt = tf.cast(y_true[:, self.label_index], tf.float32)
        lg = y_pred[:, self.label_index]        # (B,)

        # BCE для логитов: -[ y*log(sigmoid(l)) + (1-y)*log(1-
sigmoid(l)) ]
        loss =
tf.nn.sigmoid_cross_entropy_with_logits(labels=yt, logits=lg)
# (B,)

        self.mean.update_state(loss,
sample_weight=sample_weight)

    def result(self):
        return self.mean.result()

    def reset_state(self):
        self.mean.reset_state()

def compile_model(model, lr):
    model.compile(
        optimizer=tf.keras.optimizers.Adam(lr),
loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
        metrics=[
            MultiLabelAUC(name="auc"),

            # <<< loss по головам (как метрики)
            PerLabelBCE(0, name="bce_blur"),
            PerLabelBCE(1, name="bce_under"),
            PerLabelBCE(2, name="bce_over"),
        ],
    )
```

Продолжение листинга А.4

```
def main():
    physical_devices =
tf.config.experimental.list_physical_devices("GPU")
    print(tf.config.list_physical_devices())
    print("Num GPUs Available: ", len(physical_devices))

tf.config.experimental.set_memory_growth(physical_devices[0],
True)

    train_ds = make_ds(TRAIN_CSV, shuffle=True)
    val_ds = make_ds(VAL_CSV, shuffle=False)

    # 1) формы
    x_batch, y_batch = next(iter(train_ds.take(1)))
    print("\n[DEBUG] Train batch shapes:")
    print("  x:", x_batch.shape, x_batch.dtype)
    print("  y:", y_batch.shape, y_batch.dtype)

    # 2) диапазон значений пикселей (после decode +
convert_image_dtype)
    x_min = tf.reduce_min(x_batch).numpy()
    x_max = tf.reduce_max(x_batch).numpy()
    print("[DEBUG] Pixel range after preprocessing: min =",
x_min, "max =", x_max)

    model, backbone = build_model()

    print("\n[DEBUG] Model summary:")
    model.summary()

    plot_cb = PlotCurvesCallback("training_curves.png")
    csv_cb = tf.keras.callbacks.CSVLogger("train_log.csv",
append=True)
    callbacks = [plot_cb, csv_cb]

    # Этап 1: учим только головы
    backbone.trainable = False
    compile_model(model, lr=1e-3)
    model.fit(train_ds, validation_data=val_ds,
epochs=EPOCHS_HEADS, callbacks=callbacks)

    # Этап 2: fine-tune (можно разморозить всё или только
верхние слои)
    # Часто стабильнее размораживать не всё сразу, а только
```

Продолжение листинга А.4

```
верх:
    backbone.trainable = True

    # заморозить BatchNorm
    for layer in backbone.layers:
        if isinstance(layer,
tf.keras.layers.BatchNormalization):
            layer.trainable = False

    # заморозить нижнюю часть
    for layer in backbone.layers[:-40]:
        layer.trainable = False


    compile_model(model, lr=5e-5)
    model.fit(train_ds, validation_data=val_ds,
epochs=EPOCHS_FT, callbacks=callbacks)

    model.save("iq_multihd_savedmodel.keras")
    print("Saved model to iq_multihd_savedmodel")

if __name__ == "__main__":
    main()
```