

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	2
1 Анализ требований	3
2 Разработка первой версии	4
2.1 Выбор архитектуры	4
2.2 Подготовка датасета	6
2.2.1 Выбор источника фотографий	6
2.2.2 Генерация искаженных изображений	8
2.2.3 Разделение датасета	12
2.3 Первая итерация	13
2.4 Вторая итерация	17
3 Разработка второй версии	21
3.1 Доработка архитектуры	21
3.2 Расширение датасета	22
3.2.1 Выбор источника фотографий	22
3.2.2 Генерация искаженных изображений	23
3.2.3 Разделение датасета	27
3.3 Третья итерация	28
ЗАКЛЮЧЕНИЕ	31
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	33
ПРИЛОЖЕНИЕ А ИСХОДНЫЙ ТЕКСТ ПРОГРАММЫ	34

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 Разработка первой версии

2.1 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 отдельные 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.

Архитектура схематично представлена на рисунке 1.

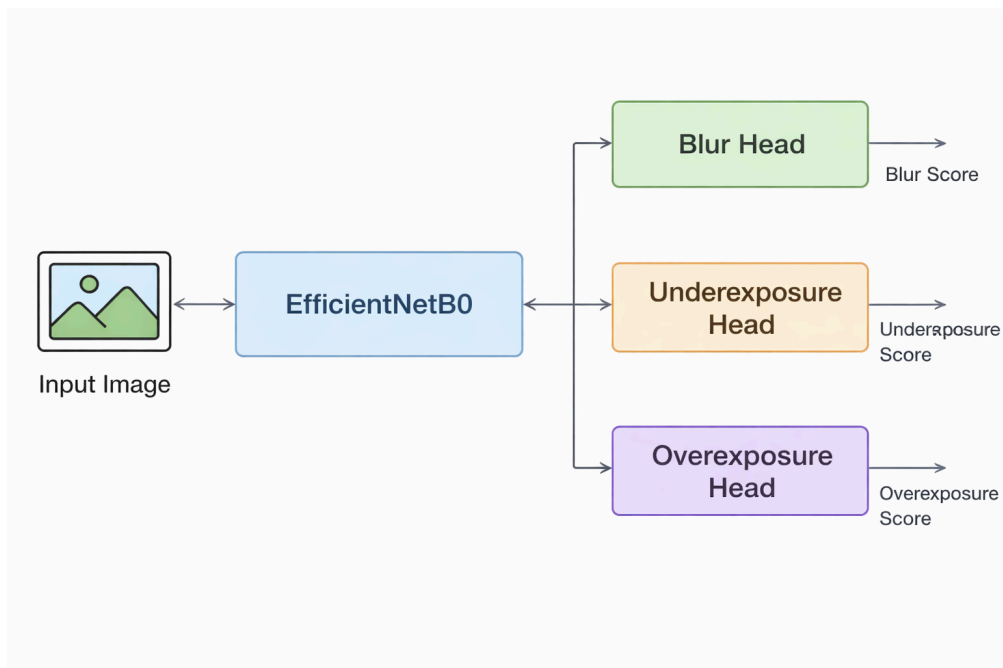


Рисунок 1 — Архитектура нейронной сети

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.

2.2 Подготовка датасета

2.2.1 Выбор источника 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. субъективную) 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).

2.2.2 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);
        str = split(tb{i,1},'.');
        dist_im_name = [str{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

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 формировались по 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
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.

2.2.3 Разделение 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"
TRAIN_OUT = "train.csv"
VAL_OUT = "val.csv"
```

Продолжение листинга 3

```
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.

2.3 Первая итерация

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 $\text{sigmoid}(\text{logit})$.

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,

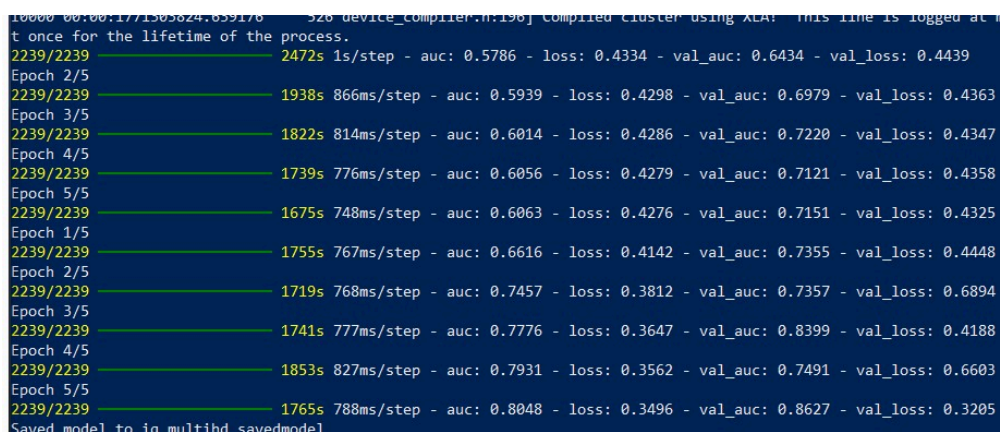
y - истинная метка ($y \in \{0, 1\}$),

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

Первая итерация проводилась 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 представлены на рисунке 2.



Epoch	Time	Step	auc	loss	val_auc	val_loss
2/5	2472s	1s/step	0.5786	0.4334	0.6434	0.4439
3/5	1938s	866ms/step	0.5939	0.4298	0.6979	0.4363
4/5	1822s	814ms/step	0.6014	0.4286	0.7220	0.4347
5/5	1739s	776ms/step	0.6056	0.4279	0.7121	0.4358
1/5	1675s	748ms/step	0.6063	0.4276	0.7151	0.4325
2/5	1755s	767ms/step	0.6616	0.4142	0.7355	0.4448
3/5	1719s	768ms/step	0.7457	0.3812	0.7357	0.6894
4/5	1741s	777ms/step	0.7776	0.3647	0.8399	0.4188
5/5	1853s	827ms/step	0.7931	0.3562	0.7491	0.6603
Final	1765s	788ms/step	0.8048	0.3496	0.8627	0.3205

Рисунок 2 — Значения `loss` и AUC первой итерации

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.

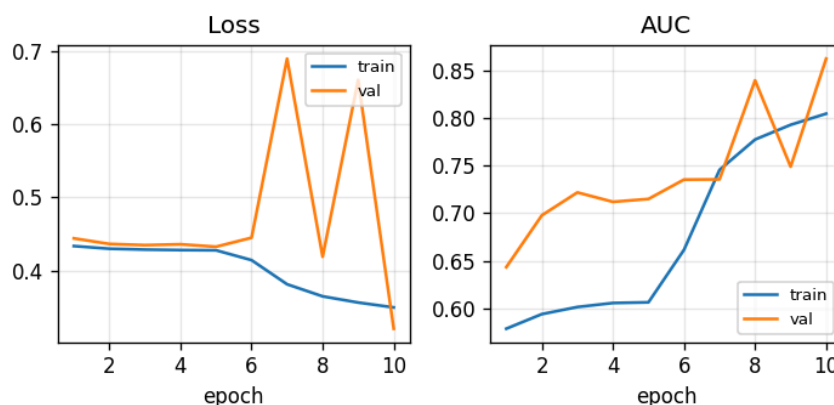


Рисунок 3 — Графики ключевых метрик первой итерации

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.

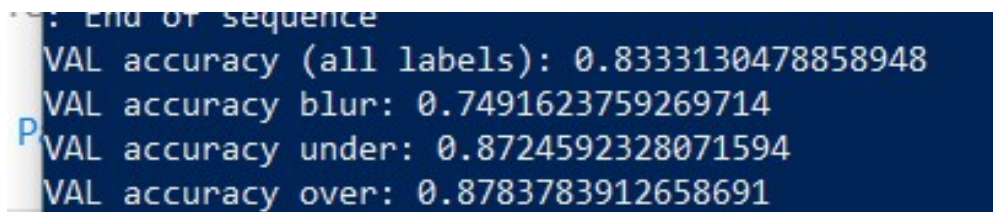


Рисунок 4 — Точность правильных предсказаний первой итерации

В целом первая итерация подтвердила 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 второй итерации модели. Программный код первой итерации представлен в приложении А.

2.4 Вторая итерация

Во второй итерации нейронной сети необходимо доработать первую итерацию 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 первой итерации 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 перекоп 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 первая итерация было слабее.

Вторая важная доработка касается устойчивости дообучения основы. В первой итерации при размораживании EfficientNet график функции потерь заметно «скакал», что связано с BatchNorm. Во второй итерации 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 $5 \cdot 10^{-5}$, 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.

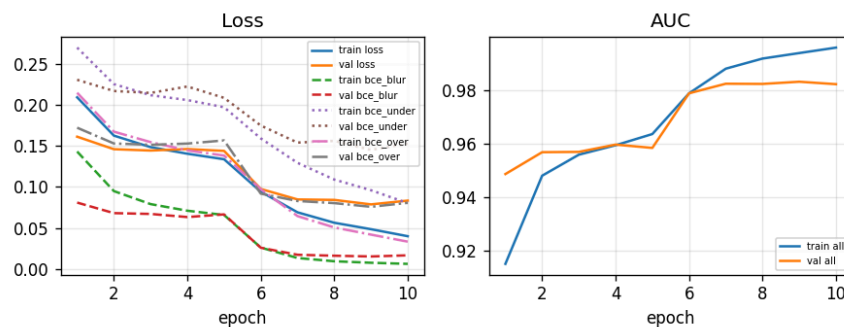


Рисунок 5 — Графики ключевых метрик второй итерации

Третья группа изменений - расширение набора метрик и улучшение журналирования. В первой итерации 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 второй итерации 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.

```

968/2239 970/2239 12:16 580ms/step - auc: 0.9951 - bce_blur: 0.0050 - bce_over: 0.0233 - bce_under:
2239/2239 1639s 732ms/step - auc: 0.9959 - bce_blur: 0.0060 - bce_over: 0.0331 - bce_under: 0.0802 - loss: 0.0398
- val_auc: 0.9822 - val_bce_blur: 0.0165 - val_bce_over: 0.0806 - val_bce_under: 0.1528 - val_loss: 0.0833
Saved model to iq_multihd_savedmodel

```

Рисунок 6 — Значения loss и AUC второй итерации

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.

```

[VAL METRICS]
val samples: 8954
thresholds: blur=0.5 under=0.5 over=0.5

Micro-accuracy (по всем меткам вместе): 0.9713
Exact-match accuracy (все 3 метки сразу): 0.9179

[BLUR]
TP=2216 FP=12 FN=30 TN=6696
pos=2246 neg=6708
precision=0.9946 recall=0.9866 f1=0.9906 accuracy=0.9953

[UNDER]
TP=912 FP=271 FN=230 TN=7541
pos=1142 neg=7812
precision=0.7709 recall=0.7986 f1=0.7845 accuracy=0.9440

[OVER]
TP=937 FP=77 FN=152 TN=7788
pos=1089 neg=7865
precision=0.9241 recall=0.8604 f1=0.8911 accuracy=0.9744

```

Рисунок 7 — Точность предсказаний второй итерации

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. После стабилизации BatchNorm и частичного дообучения основы дальнейшие эпохи с большой вероятностью дадут убывающую отдачу и увеличат риск переобучения под синтетические искажения.

Отдельно стоит обсудить метрику under. У метки under много ложноположительных срабатываний (FP=271), особенно в сравнении с другими выходами. Поэтому её точность ниже (precision 0.7709 при accuracy 0.9440). Чаще всего это ночные/темные сцены: кадр намеренно тёмный, но модель воспринимает это как недосвет. Для over эта проблема тоже прослеживается, но не так явно, потому что «естественные» затемнения в реальном мире встречаются намного чаще сильных пересветов. Проблему количества ложноположительных срабатываний головы under не решить увеличением количества параметров или изменением датасета. Решение этой проблемы требует пересмотра архитектуры нейронной сети.

Таким образом, на второй итерации устранена нестабильность проверки, усилена самая слабая голова (blur), добавлен детальный контроль обучения по метрикам, а итоговые числа подсказывают, что выход under требует доработки.

Программный код второй итерации представлен в приложении А.

3 Разработка второй версии

3.1 Доработка архитектуры

После экспериментов с первой версией модели была проведена доработка архитектуры. Базовая схема «один общая часть + несколько выходов» была сохранена. Основное изменение связано с добавлением четвертой головы `nighth`, предназначенной для распознавания ночных сцен. Это связано с тем, что `under` может ошибочно срабатывать на ночных кадрах, где низкая яркость является нормой, а не дефектом. Голова `night` реализована как отдельный выход модели и используется в прикладной логике: при высокой вероятности ночной сцены фильтрацию по `under` можно ослаблять или отключать, чтобы не браковать корректные ночные фотографии - это должно помочь повысить точность головы `under` и снизить ее количество ложноположительных срабатываний непосредственно в прикладном приложении. Примечание - здесь и далее понятие «ночная сцена» не обязательно означает прямом смысле ночную сцену (хотя и в том числе), но так же включает в себя темные и слабоосвещенные фотографии, сделанные, например, в условиях плохого освещения. Итоговая архитектура схематично представлена на рисунке 8.

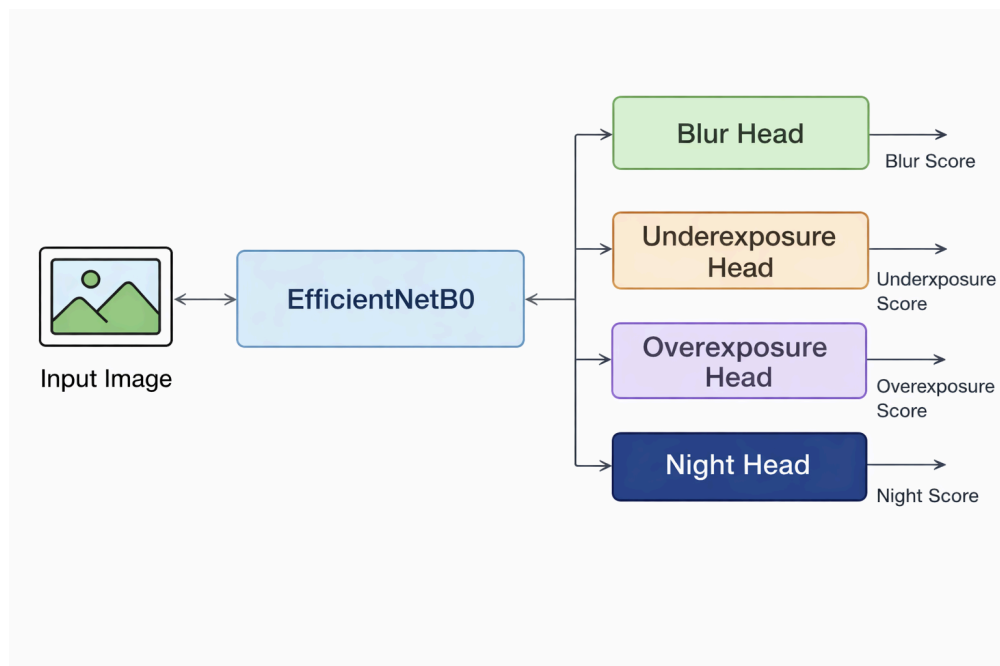


Рисунок 8 — Архитектура нейронной сети

От идеи дообучить under распознавать ночные сцены решено отказаться, потому что в прикладном приложении планируется разбивать фотографию большого разрешения на множество частей размером 512x384 пикселей, каждая из которых будет проходить через нейронную сеть и на основе общей оценки всех частей будет выдаваться вердикт о качестве фотографии.

Распознавание же ночных сцен невозможно по отдельным частям фотографии, потому что такое распознавание требует контекста всей сцены. Например, на одной фотографии темным может быть звездное небо, а светлым - здание на переднем плане. Если пытаться распознать ночь «по частям», то части со звездным небом будут распознаваться как ночные, а части со зданием - как не ночные. В общем случае сложно сказать, какая часть изображения должна быть «ночной», чтобы считать всю фотографию ночной, в отличие от других меток under и over, по которым в сообществе фотографов существует точное мнение о допустимом объеме недосвета или пересвета.

3.2 Расширение датасета

3.2.1 Выбор источника фотографий

Для обучения дополнительной головы night требовался набор изображений, где можно устойчиво отделять ночные сцены от обычных дневных/светлых. В рамках работы было принято решение собрать отдельный датасет из двух источников, чтобы обеспечить достаточное разнообразие сцен и получить явную разметку на уровне ночь/не ночь без ручной пометки каждого кадра.

В качестве источника ночных сцен использовался датасет ExDark. Его содержимое ориентировано на условия низкой освещённости, поэтому все изображения из этой папки рассматривались как примеры класса night = 1. Для этих файлов метки технических дефектов blur и over фиксировались в ноль (т.к. на этом датасете будет обучаться только голова night, головы blur, over и under обучаться не будут), а метка under назначалась случайно (0 или 1), поскольку цель данного источника- именно обучение распознаванию ночной сцены, а не точная оценка недосвета как дефекта.

В качестве источника «обычных» сцен использовался датасет MIT Indoor Scenes, преимущественно состоящий из светлых фотографий. Все изображения из этого источника считались примерами класса $\text{night} = 0$. Чтобы одновременно иметь в этом источнике как примеры без дефекта, так и примеры с дефектом *under*, внутри каждой категории изображения делились на две равные части: для первой половины задавалась метка $\text{under} = 1$, для второй — $\text{under} = 0$. Метки *blur* и *over* также фиксировались в ноль. Такой подход позволяет получить контролируемую смесь «нормальных» и «искусственно недосвеченных» дневных сцен при сохранении одинакового распределения сюжетов.

3.2.2 Генерация искаженных изображений

Искажения для изображений, помеченных как $\text{under} = 1$ и $\text{night} = 0$, создавались автоматически в MATLAB. Для таких файлов применялся тип искажения 17 (*darken*) с уровнями 4–5, после чего изображение перезаписывалось без изменения имени. Логика генерации затемнения и вызов функции искажения были реализованы на основе ранее использованных MATLAB-скриптов генерации искажений для KADIS-700K: была переиспользована проверенная функция *imdist_generator* и общий принцип «тип искажения + уровень -> получение изменённого изображения». Одновременно в *night_labels.csv* обновлялись поля *dist_type* и *dist_level*, чтобы разметка соответствовала фактически применённому преобразованию. Модифицированный код представлен в листинге 4 и в приложении А.

Листинг 4 — Модифицированный код добавления искажений

```
% apply_under_to_day_images_from_night_labels
% Читает night_labels.csv
(path,blur,under,over,night,dist_type,dist_level,ref)
% Если under==1 и night==0 -> применяет искажение dist_type=17
(darken)
% и ПЕРЕЗАПИСЫВАЕТ изображение ПО ТОМУ ЖЕ ПУТИ (имя не
меняется) .
%
% ВАЖНО:
% 1) Этот скрипт изменяет исходные файлы. Для подстраховки
можно включить BACKUP.
```

Продолжение листинга 4

```
% 2) dist_level выбирается случайно из [4 5] (как у тебя в
KADIS).
% 3) dist_type/dist_level в таблице обновляются и CSV
перезаписывается.

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

INPUT_CSV = 'night_labels.csv';
OUTPUT_CSV = 'night_labels.csv'; % перезаписать этот же файл

% Подстраховка: сделать копию файла перед перезаписью (1 = да,
0 = нет)
BACKUP_CSV = 1;
BACKUP_DIR = 'backup_before_under';

% Для 17 типа (darken) берём уровни как в проекте
allowed_levels_17 = [4 5];

% Качество JPEG при перезаписи (если файл jpg/jpeg)
JPEG_QUALITY = 95;

%% read labels
tb = readtable(INPUT_CSV);

% базовая проверка колонок
needCols =
{'path','blur','under','over','night','dist_type','dist_level','ref'};
for c = 1:numel(needCols)
    if ~ismember(needCols{c}, tb.Properties.VariableNames)
        error('В CSV нет обязательной колонки: %s',
needCols{c});
    end
end

if BACKUP_CSV
    if ~exist(BACKUP_DIR, 'dir'), mkdir(BACKUP_DIR); end
    copyfile(INPUT_CSV, fullfile(BACKUP_DIR,
['night_labels_backup_' datestr(now,'yyyymmdd_HHMMSS')
'.csv']));
end

%% find rows to process: under=1 and night=0
mask = (tb.under == 1) & (tb.night == 0);
```


Продолжение листинга 4

```
idx = find(mask);
fprintf('[INFO] Найдено строк для обработки (under=1 &
night=0): %d\n', numel(idx));

if isempty(idx)
    fprintf('[INFO] Нечего делать. Выход.\n');
    return;
end

%% process each image
processed = 0;
skipped_missing = 0;
skipped_readerr = 0;

for k = 1:numel(idx)
    i = idx(k);

    img_path = tb.path{i};

    if ~isfile(img_path)
        skipped_missing = skipped_missing + 1;
        fprintf('[WARN] Файл не найден: %s\n', img_path);
        continue;
    end

    try
        im = imread(img_path);
    catch
        skipped_readerr = skipped_readerr + 1;
        fprintf('[WARN] Не удалось прочитать: %s\n',
img_path);
        continue;
    end

    % выбираем уровень затемнения
    dist_type = 17;
    dist_level =
allowed_levels_17(randi(numel(allowed_levels_17))));

    % применяем искажение
    dist_im = imdist_generator(im, dist_type, dist_level);

    % перезаписываем БЕЗ изменения имени
    [~,~,ext] = fileparts(img_path);
    ext = lower(ext);
```

Продолжение листинга 4

```
        try
            if strcmp(ext, '.jpg') || strcmp(ext, '.jpeg')
                imwrite(dist_im, img_path, 'Quality',
JPEG_QUALITY);
            else
                % Для png/bmp/webp и т.п. пишем без параметра
Quality
                imwrite(dist_im, img_path);
            end
        catch
            fprintf('[WARN] Не удалось записать: %s\n', img_path);
            continue;
        end

        % обновляем поля в таблице (логически: это "синтетический
under")
        tb.dist_type(i) = dist_type;
        tb.dist_level(i) = dist_level;

        % blur/over оставляем 0, under уже 1, night уже 0 (как и
было)
        tb.blur(i) = 0;
        tb.over(i) = 0;

        processed = processed + 1;

        if mod(processed, 200) == 0
            fprintf('[INFO] Обработано %d / %d\n', processed,
numel(idx));
        end
    end
end

%% write updated CSV back
writetable(tb, OUTPUT_CSV);
fprintf('[OK] Готово.\n');
fprintf('  Перезаписано изображений: %d\n', processed);
fprintf('  Пропущено (нет файла): %d\n', skipped_missing);
fprintf('  Пропущено (ошибка чтения): %d\n', skipped_readerr);
fprintf('  CSV обновлён: %s\n', OUTPUT_CSV);
```

В результате для хранения разметки был сформирован единый файл `night_labels.csv` с дополнительным полем `nigth`, которого нет в первом датасете.

3.2.3 Разделение датасета

После формирования `night_labels.csv` данные были разделены на обучающую и валидационную выборки. Скрипт, разделяющий датасет, полностью аналогичен скрипту для первой версии, за исключением доработки для работы с полем `night`.

Скрипт, разделяющий датасет на обучающую и валидационную выборки содержится в 5 и в приложении А.

Листинг 5 — Скрипт, разделяющий датасет на обучающую и валидационную выборки

```
import pandas as pd
import numpy as np

LABELS_CSV = "night_labels.csv"
TRAIN_OUT = "night_train.csv"
VAL_OUT = "night_val.csv"

VAL_RATIO = 0.20
SEED = 42

df = pd.read_csv(LABELS_CSV)

# пути под Linux
df["path"] = df["path"].astype(str).str.replace("\\", "/",
regex=False)
df["path"] = df["path"].str.replace("//", "/", regex=False)
df["ref"] = df["ref"].astype(str).str.replace("\\", "/",
regex=False)

rng = np.random.RandomState(SEED)

# stratified split по night на уровне ref
ref_night = df.groupby("ref")["night"].max().reset_index()
refs_1 = ref_night.loc[ref_night["night"] == 1, "ref"].values
refs_0 = ref_night.loc[ref_night["night"] == 0, "ref"].values

rng.shuffle(refs_1)
rng.shuffle(refs_0)

n_val_1 = int(round(len(refs_1) * VAL_RATIO))
n_val_0 = int(round(len(refs_0) * VAL_RATIO))

val_refs = set(refs_1[:n_val_1]) | set(refs_0[:n_val_0])
train_refs = set(refs_1[n_val_1:]) | set(refs_0[n_val_0:])
```

Продолжение листинга 5

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

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

print("Total rows:", len(df))
print("Train rows:", len(train_df), " Val rows:", len(val_df))
print("Night share total:", df["night"].mean())
print("Night share train:", train_df["night"].mean(), " val:",
val_df["night"].mean())
print("Example path (train):", train_df.loc[0, "path"])
```

В результате были получены файлы `night_train.csv` и `night_val.csv`, содержащие как изображения, так и разметку, при этом доля ночных сцен в обеих выборках сохраняется близкой к исходной. Это обеспечивает стабильную проверку качества головы `night`.

3.3 Третья итерация

Процесс обучения был разделён на три этапа. На первом этапе (5 эпох) общая часть была заморожена, и обучались только головы дефектов качества (`blur/under/over`); голова `night` в этот момент не участвовала в оптимизации. На втором этапе (ещё 5 эпох) выполнялась тонкая настройка верхних слоёв бэкбона: слои пакетной нормализации `BatchNorm` фиксировались, нижняя часть `EfficientNet` оставалась замороженной, а обучение шло с уменьшенной скоростью обучения. На третьем этапе (5 эпох) обучалась только голова `night` на отдельном датасете ночных/дневных сцен; при этом вклад остальных голов в функцию потерь отключался весами.

Качество голов `blur/under/over` оценивалось на обычной валидации `val.csv` (8954 изображения) при порогах 0.5. Итоговые значения не отличаются от значений второй итерации - что не удивительно, потому что изменения в обучение трех изначальных голов не вводились.

Качество головы `night` оценивалось отдельно на `night_val.csv` (4283 изображения) при пороге 0.5. Получены значения: TP=1414, FP=28, FN=59,

TN=2782, что соответствует precision 0.9806, recall 0.9599, F1 0.9702 и accuracy 0.9797. Эти результаты показывают, что модель уверенно отделяет ночные сцены от дневных, а значит может использоваться как надёжный «контекстный фильтр» для under.

В процессе проверки оказалось, что среди всех ложных срабатываний under на val.csv в 66.86% случаев (234 из 350) ночная голова выдаёт night_pred = 1. В таком случае потенциально устраняется около 234 ложных срабатываний under, и остаётся порядка 116 FP, что уже сопоставимо с уровнем FP для over (106) и существенно ближе к «стабильным» головам (blur/over/night), FP которых равен около 30. Практически это подтверждает, что идея отдельной головы night является работоспособной именно как механизм снижения ложных браков по under.

Результаты проверки точности представлены на рисунке 9.

```
=====
[VAL.CSV] blur/under/over
=====
val samples: 8954
thresholds: blur=0.5 under=0.5 over=0.5

Micro-accuracy (3 метки): 0.9689
Exact-match accuracy (3 метки): 0.9119

[BLUR]
TP=2233 FP=31 FN=13 TN=6677
precision=0.9863 recall=0.9942 f1=0.9902 accuracy=0.9951

[UNDER]
TP=949 FP=350 FN=193 TN=7462
precision=0.7306 recall=0.8310 f1=0.7776 accuracy=0.9394

[OVER]
TP=947 FP=106 FN=142 TN=7759
precision=0.8993 recall=0.8696 f1=0.8842 accuracy=0.9723
2026-02-21 14:10:36.956711: W tensorflow/core/lib/png/png_
2026-02-21 14:10:57.686980: W tensorflow/core/lib/png/png_
2026-02-21 14:12:11.224365: W tensorflow/core/lib/png/png_
2026-02-21 14:12:11.224473: W tensorflow/core/lib/png/png_
2026-02-21 14:16:20.546156: I tensorflow/core/framework/lo

=====
[NIGHT_VAL.CSV] ONLY night head
=====
night_val samples: 4283
threshold: night=0.5

[NIGHT]
TP=1414 FP=28 FN=59 TN=2782
precision=0.9806 recall=0.9599 f1=0.9702 accuracy=0.9797
```

Рисунок 9 — Точность предсказаний третьей итерации

Однако график ключевых метрик ведет себя странно после 10-й эпохи. Это объясняется тем, что кривые строятся единым графиком для всех 15 эпох, однако начиная с 11-й эпохи меняется датасет и цель оптимизации: обучение переключается на задачу night, а метки blur/under/over на этом этапе не являются целевыми и фактически отключаются весами. Из-за этого значения метрик по головам качества на третьем этапе теряют смысл (они считаются на данных другого распределения и при иной схеме весов), а общая AUC, построенная по всем меткам, также становится некорректной для сравнения с первыми 10 эпохами. Поэтому корректная интерпретация обучения выполняется не по объединённому графику, а по итоговым метрикам проверки точности на соответствующих валидационных наборах: val.csv для blur/under/over и night_val.csv для night. Графики обучения представлены на рисунке 10.

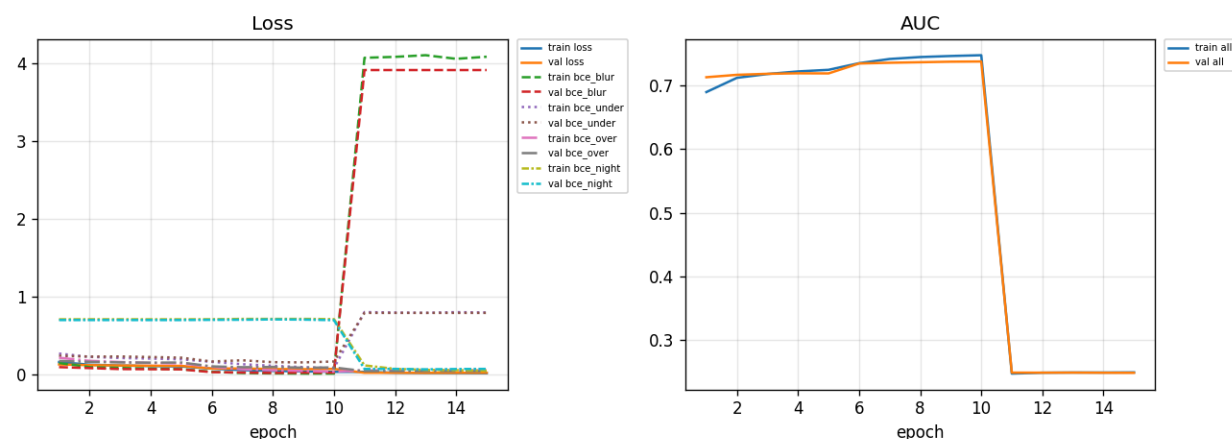


Рисунок 10 — Графики ключевых метрик третьей итерации

Таким образом, полученная версия модели является финальной для текущего этапа работ: она демонстрирует высокое качество по дефектам blur/over и приемлемое качество по under, а добавление головы night даёт практический механизм снижения ложных срабатываний under на ночных сценах в прикладной логике приложения - можно использовать правило: если кадр распознан как ночной, фильтрацию по under не применять. Текущая конфигурация подходит для встраивания в дипломное приложение. Программный код третьей итерации представлен в приложении А.

ЗАКЛЮЧЕНИЕ

В ходе преддипломной практики была разработана нейросетевая подсистема для дипломного приложения-редактора, предназначенная для автоматической оценки технического качества пользовательских фотографий. Базовая постановка задачи включала три технических дефекта: размытие (blur), недосвет (under) и пересвет (over). Для обучения была подготовлена выборка на основе KADIS-700K с отбором релевантных типов искажений, сформирован единый файл разметки labels.csv, выполнено разбиение train/val без утечек по сценам (по полю ref), а также реализован конвейер обучения в TensorFlow через tf.data.

В качестве основы модели использован EfficientNetB0 и архитектура с несколькими выходными головами. Во второй итерации была повышена стабильность обучения за счёт заморозки BatchNorm, увеличения количества нейронов у голов и частичного дообучения верхних слоёв бэкбона, расширен набор метрик для контроля качества по каждой метке. На валидационной выборке по дефектам качества получены значения: micro-accuracy = 0.9689 и exact-match accuracy = 0.9119; по головам blur достигнуты precision = 0.9863 и recall = 0.9942 (F1 = 0.9902), по over — precision = 0.8993 и recall = 0.8696 (F1 = 0.8842). Метка under оставалась наиболее сложной из-за ложных срабатываний (FP = 350 при пороге 0.5).

Для уменьшения ложных срабатываний under в финальной версии добавлена четвёртая голова night, распознающая ночные сцены. Голова night обучалась отдельно на специализированной разметке (night_train.csv/night_val.csv) без влияния на головы качества. По результатам проверки на night_val.csv достигнуты precision = 0.9806, recall = 0.9599, F1 = 0.9702 и accuracy = 0.9797, что подтверждает пригодность night для контекстной фильтрации. Дополнительно показано, что среди ложных срабатываний under на val.csv в 66.86% случаев модель одновременно предсказывает night_pred = 1 (234 из 350), то есть отключение/ослабление фильтрации under при ночной сцене позволяет существенно снизить число ложноположительных срабатываний и приблизить уровень FP по under к значениям других голов.

Таким образом, разработанная четырёхголовая модель обеспечивает высокое качество распознавания размытия и пересвета, а также предоставляет практический механизм уменьшения ошибок по недосвету за счёт учёта контекста сцены. Полученная версия может быть встроена в дипломное приложение для автоматического отбора фотографий с возможностью настройки порогов и правил фильтрации на уровне прикладной логики.

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 22

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
        str = split(tb{i,1},'.');

        dist_im_name = [str{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 generate_kadis700k_night.m

```
%% apply_under_to_day_images_from_night_labels
% Читает night_labels.csv
(path,blur,under,over,night,dist_type,dist_level,ref)
% Если under==1 и night==0 -> применяет искажение dist_type=17
(darken)
% и ПЕРЕЗАПИСЫВАЕТ изображение ПО ТОМУ ЖЕ ПУТИ (имя не
меняется) .
%
% ВАЖНО:
% 1) Этот скрипт изменяет исходные файлы. Для подстраховки
можно включить BACKUP.
% 2) dist_level выбирается случайно из [4 5] (как у тебя в
KADIS) .
% 3) dist_type/dist_level в таблице обновляются и CSV
перезаписывается.

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

INPUT_CSV = 'night_labels.csv';
OUTPUT_CSV = 'night_labels.csv'; % перезаписать этот же файл

% Подстраховка: сделать копию файла перед перезаписью (1 = да,
0 = нет)
BACKUP_CSV = 1;
BACKUP_DIR = 'backup_before_under';

% Для 17 типа (darken) берём уровни как в проекте
allowed_levels_17 = [4 5];

% Качество JPEG при перезаписи (если файл jpg/jpeg)
JPEG_QUALITY = 95;

%% read labels
tb = readtable(INPUT_CSV);

% базовая проверка колонок
needCols =
```

Продолжение листинга А.2

```
{'path','blur','under','over','night','dist_type','dist_level','ref'};
for c = 1:numel(needCols)
    if ~ismember(needCols{c}, tb.Properties.VariableNames)
        error('В CSV нет обязательной колонки: %s',
needCols{c});
    end
end

if BACKUP_CSV
    if ~exist(BACKUP_DIR, 'dir'), mkdir(BACKUP_DIR); end
    copyfile(INPUT_CSV, fullfile(BACKUP_DIR,
['night_labels_backup_' datestr(now,'yyyymmdd_HHMMSS')
'.csv']));
end

%% find rows to process: under=1 and night=0
mask = (tb.under == 1) & (tb.night == 0);

idx = find(mask);
fprintf('[INFO] Найдено строк для обработки (under=1 &
night=0): %d\n', numel(idx));

if isempty(idx)
    fprintf('[INFO] Нечего делать. Выход.\n');
    return;
end

%% process each image
processed = 0;
skipped_missing = 0;
skipped_readerr = 0;

for k = 1:numel(idx)
    i = idx(k);

    img_path = tb.path{i};

    if ~isfile(img_path)
        skipped_missing = skipped_missing + 1;
        fprintf('[WARN] Файл не найден: %s\n', img_path);
        continue;
    end

    try
        im = imread(img_path);
    catch
```

Продолжение листинга А.2

```
        skipped_readerr = skipped_readerr + 1;
        fprintf('[WARN] Не удалось прочитать: %s\n',
img_path);
        continue;
    end

    % выбираем уровень затемнения
    dist_type = 17;
    dist_level =
allowed_levels_17(randi(numel(allowed_levels_17))));

    % применяем искажение
    dist_im = imdist_generator(im, dist_type, dist_level);

    % перезаписываем БЕЗ изменения имени
    [~,~,ext] = fileparts(img_path);
    ext = lower(ext);

    try
        if strcmp(ext, '.jpg') || strcmp(ext, '.jpeg')
            imwrite(dist_im, img_path, 'Quality',
JPEG_QUALITY);
        else
            % Для png/bmp/webp и т.п. пишем без параметра
Quality
            imwrite(dist_im, img_path);
        end
    catch
        fprintf('[WARN] Не удалось записать: %s\n', img_path);
        continue;
    end

    % обновляем поля в таблице (логически: это "синтетический
under")
    tb.dist_type(i) = dist_type;
    tb.dist_level(i) = dist_level;

    % blur/over оставляем 0, under уже 1, night уже 0 (как и
было)
    tb.blur(i) = 0;
    tb.over(i) = 0;

    processed = processed + 1;

    if mod(processed, 200) == 0
        fprintf('[INFO] Обработано %d / %d\n', processed,
```

Продолжение листинга А.2

```
numel(idx));
    end
end

%% write updated CSV back
writetable(tb, OUTPUT_CSV);
fprintf('[OK] Готово.\n');
fprintf('  Перезаписано изображений: %d\n', processed);
fprintf('  Пропущено (нет файла): %d\n', skipped_missing);
fprintf('  Пропущено (ошибка чтения): %d\n', skipped_readerr);
fprintf('  CSV обновлён: %s\n', OUTPUT_CSV);
```

Листинг А.3 — 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))
```


Продолжение листинга А.3

```
print("Unique refs train/val:", len(train_refs),  
      len(val_refs))
```

Листинг А.4 — Содержимое файла dataset_linux_night.py

```
import pandas as pd  
import numpy as np  
  
LABELS_CSV = "night_labels.csv"  
TRAIN_OUT = "night_train.csv"  
VAL_OUT = "night_val.csv"  
  
VAL_RATIO = 0.20  
SEED = 42  
  
df = pd.read_csv(LABELS_CSV)  
  
# пути под Linux  
df["path"] = df["path"].astype(str).str.replace("\\\\", "/",  
regex=False)  
df["path"] = df["path"].str.replace("//", "/", regex=False)  
df["ref"] = df["ref"].astype(str).str.replace("\\\\", "/",  
regex=False)  
  
rng = np.random.RandomState(SEED)  
  
# stratified split по night на уровне ref  
ref_night = df.groupby("ref")["night"].max().reset_index()  
refs_1 = ref_night.loc[ref_night["night"] == 1, "ref"].values  
refs_0 = ref_night.loc[ref_night["night"] == 0, "ref"].values  
  
rng.shuffle(refs_1)  
rng.shuffle(refs_0)  
  
n_val_1 = int(round(len(refs_1) * VAL_RATIO))  
n_val_0 = int(round(len(refs_0) * VAL_RATIO))  
  
val_refs = set(refs_1[:n_val_1]) | set(refs_0[:n_val_0])  
train_refs = set(refs_1[n_val_1:]) | set(refs_0[n_val_0:])  
  
train_df =  
df[df["ref"].isin(train_refs)].reset_index(drop=True)  
val_df = df[df["ref"].isin(val_refs)].reset_index(drop=True)  
  
train_df.to_csv(TRAIN_OUT, index=False)
```

Продолжение листинга A.4

```
val_df.to_csv(VAL_OUT, index=False)

print("Total rows:", len(df))
print("Train rows:", len(train_df), " Val rows:", len(val_df))
print("Night share total:", df["night"].mean())
print("Night share train:", train_df["night"].mean(), " val:",
      val_df["night"].mean())
print("Example path (train):", train_df.loc[0, "path"])
```

Листинг A.5 — Содержимое файла ML_first_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
```

Продолжение листинга А.5

```
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 = []
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)
```

Продолжение листинга А.5

```
# ===== 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="--")

    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()
```

Продолжение листинга А.5

```
        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

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)
```

Продолжение листинга А.5

```
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,
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):
```

Продолжение листинга А.5

```
        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:  $-\left[ y \cdot \log(\text{sigmoid}(l)) + (1-y) \cdot \log(1-\text{sigmoid}(l)) \right]$ 
        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"),
        ],
    )

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],
```

Продолжение листинга А.5

```
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
верх:
backbone.trainable = True

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


Продолжение листинга А.5

```
# 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.6 — 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"):
```

Продолжение листинга А.6

```
        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 = []
        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 ----
```

Продолжение листинга А.6

```
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="--")

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")
```

Продолжение листинга А.6

```
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

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(
```

Продолжение листинга А.6

```
        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)
        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
```

Продолжение листинга А.6

```
# 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,
sample_weight=sample_weight)

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

Продолжение листинга А.6

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

    model, backbone = build_model()
```

Продолжение листинга А.6

```
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()
```