2. ПРАКТИЧЕСКАЯ ЧАСТЬ

В данной главе будет разработан модель нейронной сети по выявлению дипфейка, а также написан графический интерфейс для выведения результата.

2.1. Архитектура нейронной сети

Метод основан на подходе “ Self-Supervised MultiModal Versatile Networks” (MMV) Алайрака [1], в котором представления, специфичные для модальности, изучаются с помощью методов контрастивного обучения.

Имея набор немаркированных видеороликов с несколькими модальностями, MMV стремится изучитьмодель, которая может использовать любую модальность, на которой она была обучена, и вывести представление, которое можно сравнить с другими

Данная система (Рисунок 2) предназначена для обнаружения дипфейков на основе анализа видео и аудиоданных. В этой работе представлено описание этой архитектуры, которая включает в себя три нейронные сети для обработки различных типов входных данных и последующего объединения их результатов для классификации видео на настоящие и фальшивые.

Система состоит из трех основных компонентов:

- обработка текстовой информации с использованием Word2Vec;
- обработка видеоданных с использованием TSM-50;
- обработка аудиоданных с использованием ResNet50 для анализа спектограмм.

Для обработки текстовой информации, содержащейся в названии видеофайла, используется модель Word2Vec. Это позволяет преобразовать текстовую информацию в векторное представление, учитывающее семантическую близость между словами. Для анализа видеоданных применяется модель TSM-50 (Temporal Shift Module), которая обладает способностью анализировать временные изменения в видеопотоке. Это позволяет выделять ключевые моменты в видео для последующего анализа. С помощью ResNet50 производится аудиомодальность.

Перед валидацией каждый слой с помощью Shared Space Projection конвертируется либо в VAT, либо VA пространства (где V – видео , A – аудио ,T - текст).

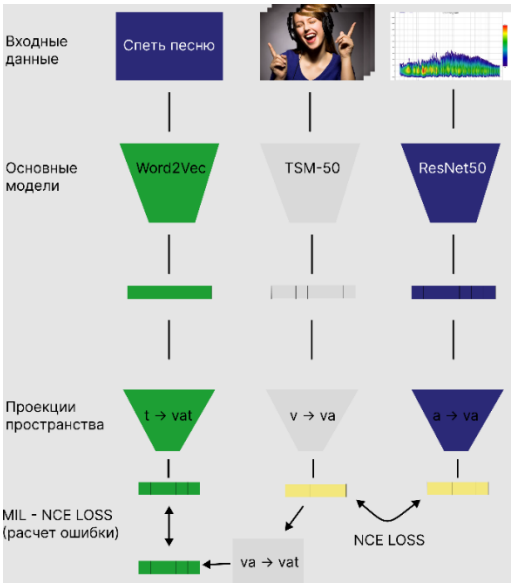

Рисунок 2.1. Архитектура обучения системы обнаружения DeepFake

В качестве оценки ошибки в пространстве текстовой информации используется метрика MIL-NCE (Multiple Instance Learning with Noise Contrastive Estimation). Для обработки видеоданных и аудиоданных используется метрика NCE (Noise Contrastive Estimation). Эти метрики помогают оценить близость между векторными представлениями и определить, является ли видео дипфейком.

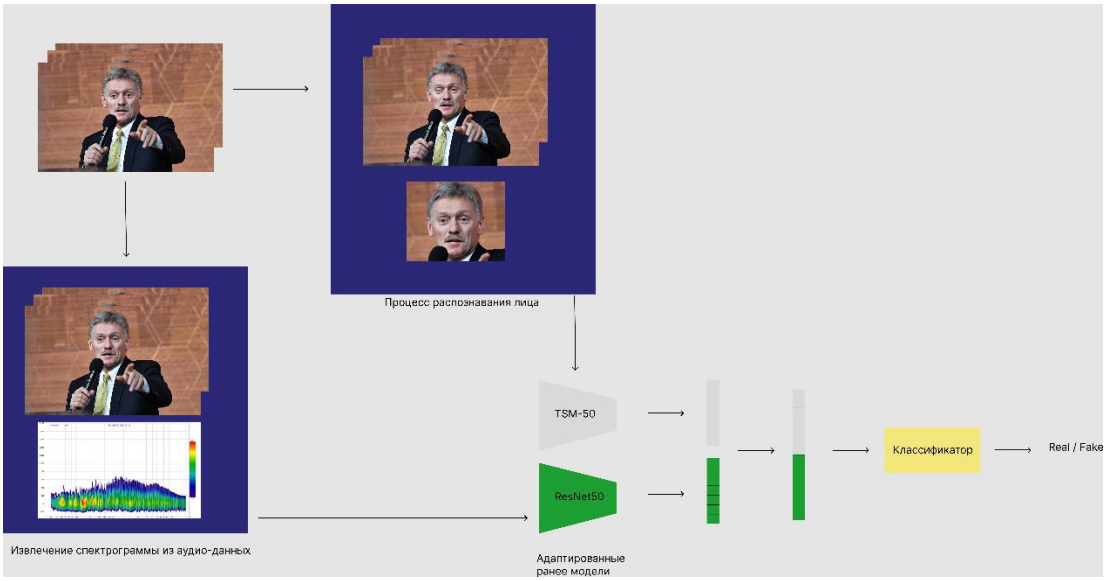

Рисунок 2.2. Архитектура системы обнаружения DeepFake

После получения векторных представлений из текста, видео и аудио они объединяются и проходят через классификатор, который определяет, является ли видео дипфейком или нет. Это достигается путем конкатенации векторов и последующего самообучения классификатора на этом объединенном пространстве.

2.2. Обработка видеоданных

2.2.1. TSM-50

Для распознавания лиц была использована модель на основе CNN, называемую модулем временного сдвига (TSM), которая может изучать пространственно-временные особенности в 3D-данных с вычислительной сложностью 2D CNN (см. приложение Б.1.).

Данный модуль подготавливает данные для их передачи в ансамбль последующих моделей TSM и ResNet.

Для повышения точности распознавания лиц применены техники аугментации данных, чтобы расширить обучающий набор и улучшить обобщающую способность модели.

Нейронная сеть обучается на большом наборе данных, где каждому лицу присваивается уникальный идентификатор. Во время обучения сеть корректирует свои веса таким образом, чтобы минимизировать ошибку классификации лиц в будущем.

Для обработки видео и данных от камер используются специализированные библиотеки, такие как OpenCV, dlib и TensorFlow, которые предоставляют функции для захвата, обработки и анализа видеопотока и изображений.

2.2.2. Shared Space Projection (SSP)

Теперь перейдем к shared space projection (SSP). Это метод, используемый для сопоставления различных модальностей данных, таких как текст и изображения, в общее пространство. Идея состоит в том, чтобы преобразовать

представления из разных пространств в общее пространство, где они могут быть сравнимыми и использоваться в совместных анализах.

Процесс SSP обычно включает следующие этапы:

1. подготовка данных: представления из разных пространств должны быть предварительно обработаны и готовы для сопоставления;
2. проекция в общее пространство: происходит преобразование представлений из исходных пространств в общее пространство с помощью различных методов, таких как конкатенация или другие линейные/нелинейные преобразования;
3. вычисление расстояний или мер сходства: после проекции представлений можно вычислить расстояния или меры сходства между объектами в общем пространстве.

В данном случае необходимо получить аудио-представление, поэтому в дальнейшем происходит работа с пространством VA.

2.2. Обработка аудиоданных

2.2.1. ResNet50

В процессе работы со сверточной моделью ResNet50 (см. приложение Б.2.) каждое видео дополняется соответствующим аудиопотоком, извлеченным из оригинальных видеороликов. В случае подделанных видеороликов определены методы замены лиц, чтобы дополнить видео подходящим аудиопотоком, определяемым техникой манипулирования, которая использовалась для его создания. Каждое обработанное видео является результатом исходной и целевой последовательностей; при использовании техники замены лиц исходная последовательность вставляется в целевое видео, при этом делается попытка сохранить движение губ человека.

Для реализации сопоставления аудио и мимики на видео используется ансамбль предыдущих моделей (см. приложение Б.3.). Метод основан на статье "TSM: Temporal Shift Module for Efficient Video Understanding" [2].

Основные моменты работы подсистемы заключаются в:

1. TSM модуль с ResNet архитектурой: в модуле TSMResNet представлена реализация блока ResNet с добавлением Temporal Channel Shifting, что позволяет учитывать временные изменения при анализе видео-данных. Temporal Channel Shifting является ключевым компонентом TSM модуля, который обрабатывает временные изменения в каналах видео-данных, учитывая эффекты движения и динамики;

2. использование Haiku и JAX: для реализации нейронной сети используются библиотеки Haiku и JAX, обеспечивающие возможность задания модулей и эффективное вычисление на платформе JAX. Haiku позволяет создавать модули с автоматически управляемыми параметрами, облегчая процесс оптимизации моделей;

3. участие в обучении: модель, основанная на TSM с ResNet архитектурой, способна обучаться на размеченных данных, чтобы научиться распознавать особенности DeepFake технологий и делать классификацию видео.

2.2.2. Основные метрики

На этапе валидации используется ошибка NCE (Noise Contrastive Estimation) - метод оценки качества видео и аудио материала с целью проверки согласованности между ними и выявления возможных искажений или deepfake (фальшивые) элементов. Метод NCE используется для сравнения реального и фальшивого видео с целью обнаружения и устранения шумов и артефактов, которые могут быть присутствовать в ложных данных.

$$\mathrm{NCE}(x_v, x_a) = -\log\left(\frac{\exp(z_{v,va}^{\top} z_{a,va}/\tau)}{\exp(z_{v,va}^{\top} z_{a,va}/\tau) + \sum_{z' \sim \mathcal{N}(x)} \exp(z'^{\top}_{v,va} z'_{a,va}/\tau)}\right)$$

Рисунок 2.3. Формула NCE ошибки

Где N (x) - набор пар отрицательных модальностей для видео x, z – набор пространственных сеток, зависящий от аудио и/или видео, а τ - температурный параметр.

Пространственное объединение применяется на последнем уровне визуальной основы для получения вектора визуального представления.

Необработанные звуковые сигналы преобразуются в логарифмические спектрограммы с 80 ячейками. Пространственное объединение применяется на последнем уровне визуальной основы.

Векторы визуальной модальности имеют размерность dv =2048 для TSM-50 и dv = 4096 для TSM-50x2. Векторы аудиомодальностей имеют размерность da = 2048. После объединения векторы представления кросс-модальностей размерности 4096 и 6144 создаются для основы нейронной сети с помощью TSM-50 и TSM-50x2. Классификатор представляет собой многослойный персептрон, реализованный с двумя скрытыми слоями.

Adam optimizer используется с графиком косинусоидального затухания с начальным значением 1^{e-3} и $\alpha = 0,95$.

2.3. Обработка текстовых данных

2.3.1. Word2Vec

Word2Vec - это алгоритм, используемый для создания векторных представлений слов на основе их семантического контекста в тексте. Он принимает на вход большой корпус текста и создает векторное представление для каждого слова в этом корпусе. Векторы слов представляют собой числовые вектора фиксированной длины, которые содержат информацию о семантическом значении слова.

Обработка текстовых данных с использованием Word2Vec обычно включает следующие шаги:

1. Предварительная обработка текста: удаление стоп-слов, токенизация, приведение к нижнему регистру и т. д.
2. Обучение модели Word2Vec на предварительно обработанном корпусе текста.
3. Создание векторных представлений слов на основе обученной модели.

После получения векторных представлений слов из модели Word2Vec производится их преобразование в пространство VAT Shared Space Projection.

Другими словами, к векторам токенов необходимо добавить еще 2 пространственных единицы – видео и аудио.

2.4.2. Основные метрики

Наравне с полученной проекцией, имеем такой же формат после конкатенирования TSM и RESNET результатов. Сравнивая полученные данные, используем ошибку MIL-NCE.

MIL-NCE Loss представляет собой отрицательное среднее пересечение по объединению между предсказанными масками и их истинными масками, что позволяет использовать эту ошибку как функцию потерь для обучения моделей сегментации в компьютерном зрении. Кроме того, важно отметить, что MIL-NCE Loss обычно используется в контексте обучения с использованием метода Multiple Instance Learning (MIL), где обучающие примеры представлены в виде суммарных характеристик набора объектов (сегментов). MIL-NCE Loss адаптирована для использования в таких задачах, где необходимо учитывать относительное положение объектов в сцене. Векторное представление MIL-NCE Loss основывается на векторизации IOU значений для каждого пикселя на изображении.

$$L_g = -log\left(\frac{\sum_{z_v^g \in \mathcal{P}} exp(z_a^T z_v^g)}{\sum_{z_v^g \in \mathcal{P}} exp(z_a^T z_v^g) + \sum_{z' \in \mathcal{N}} exp(z_a'^T z_v'^g)}\right)$$

Рис. 2.4. Формула MIL-NCE ошибки

где N - набор негативных аудио- и визуальных пар, P - набор пространственных сеток в z_u^g.

Именно результаты сравнения будут предсказывать классы TRUE/FALSE. Метрикой оценки качества является ROC-AUC кривая(см. рисунок 2.5.), показывающая зависимость между чувствительностью и специфичностью классов.

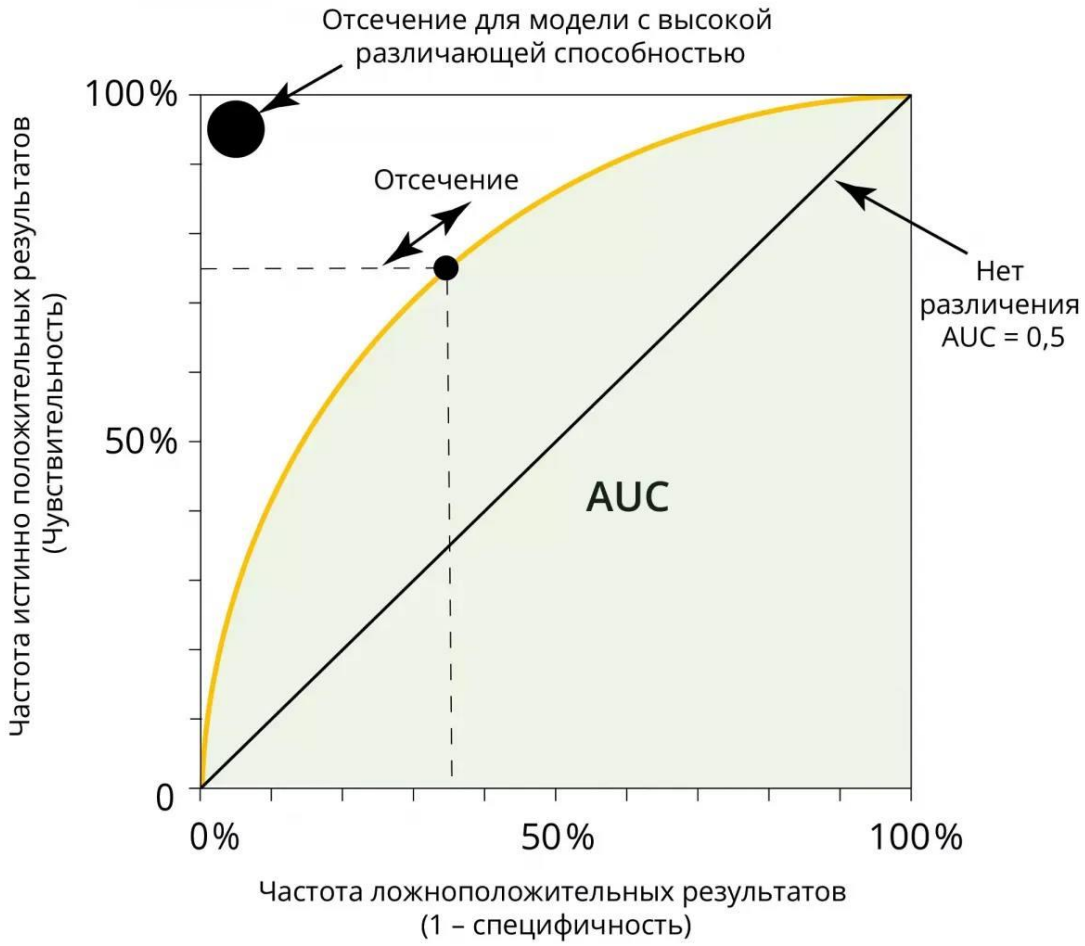

Рисунок 2.5. Метрика оценки производительности системы

2.5. Программа для распознавания DeepFake технологий

2.5.1. Тренировочные и валидационные данные

Основа была предварительно обучена на многомодальных наборов данных, которые являются инвариантными к задаче обнаружения фейков. Для предварительного обучения использовались наборы данных HowTo100M и AudioSets. Время, затраченное на обучение, составило 18,5 часов.

HowTo100M - это крупномасштабный набор данных о видеороликах с комментариями, в котором особое внимание уделяется обучающим видеороликам с явным намерением объяснить визуальный контент на экране.

Этот набор данных содержит 136 миллионов видеоклипов с подписями, взятых из видеороликов YouTube. Набор данных AudioSet представляет собой крупномасштабную коллекцию звуковых событий, аннотированных вручную.

В нем представлено 632 класса звуковых событий и более 2 миллионов 10-секундные клипы с человеческими надписями, взятые с YouTube.

Были проведены эксперименты на нескольких наборах данных, чтобы проверить работоспособность системы в различных сценариях. Для обучения использованы FaceForensics++ (FF++). FF++ содержит 1000 реальных видео и 4000 поддельных видеороликов, сгенерированных с помощью двух алгоритмов изменения лица и двух алгоритмов реконструкции лица. Для изменения лица используются DeepFake и FaceSwap. Нейронные структуры Face2Face используются в качестве алгоритмов реконструкции. FF++ обеспечивает три степени сжатия; было решено использовать высоко-качественную версию, представляющую собой нечто среднее между необработанным и сильно сжатым видео. Видео, которые обычно можно найти в социальных сетях, соответствуют этой степени сжатия.

Для тестирования было решено использовать DeepFake Detection Challenge (DFDC) [3], включающий восемь алгоритмов модификации лица. Тестовый набор DFDC состоит из 5000 видеороликов. DeeperForensics (DFo) - набор данных, созданный с использованием исходных видеороликов FF++ и техника замены лиц. Для оценки этих наборов данных мы используем метод FF++ train-test split. DeepfakeTIMIT - это набор аудиовизуальных данных, в котором поддельные видеоролики создаются с использованием подхода, основанного на GAN, без каких-либо манипуляций со звуком. Набор данных содержит 640 видеороликов, которые были использованы для тестирования в этой работе в их штаб-версии.

Общий вес всех датасетов составил 214.5 ГБ (при этом из некоторых были извлечены только чанки).

2.5.2. Запуск программного обеспечения

Для начала работы с программой необходимо подготовить данные для тренировки и валидации. Для этого запускаем файл data_preparation.py.

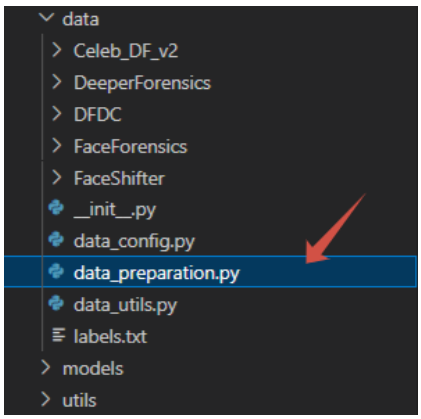

Рисунок 2.6. Запуск data_preparation

После чего запускаем файл main.py, переходим на сервер, загружаем видео и смотрим результаты.

```
C:\Users\пк\Desktop\VideoDeepFakeDetection-main>python main.py
 * Running on http://127.0.0.1:5000
Testing.....
102.82982373737 sec
```

Рисунок 2.7. Результаты работы программы в консоли

Стоит отметить, что процесс обучения происходил на графическом процессоре (GPU или Graphics Processing Unit). Для лучшего воспринимания работы программы использован сервер, который приспособлен под визуализацию и написан на языке HTML. Другими словами, имплементированный веб-сервер хранит только результаты выполненной программы.

Для начала проверим исправность работы модели для реального видео:

Рисунок 2.8. Графический результат работы модели на реальное видео

Полученная система распознает реальные видео с 100% точностью. Перейдем к DeepFake видео, взятым из вышеупомянутых датасетов.

Рисунок 2.9.1. Графический результат работы модели на DeepFake видео

Рисунок 2.9.2. Графический результат работы модели на DeepFake видео

Рисунок 2.9.3. Графический результат работы модели на DeepFake видео

Как можно заменить, программа отлично справляется со своей задачей по детекции неоригинальных видео.

Вывод

В данной работе был проведен анализ эффективности использования ансамбля моделей TSM50 и ResNet50 для выявления поддельных видео, созданных с использованием технологий DeepFake, таких как face-switch и другие методы манипулирования.

Использование ансамбля моделей TSM50 и ResNet50 позволило добиться значительного улучшения в обнаружении поддельных видео по сравнению с предыдущими методами. Эти модели обладают высокой степенью точности и надежности при выявлении характеристик, характерных для фейковых видео. Для улучшения качества работы программы в будущем предусматривается оптимизация параметров и увеличение мета-данных.

ЗАКЛЮЧЕНИЕ

Детекция DeepFake технологий представляет собой важную задачу в современном мире, где производство и распространение поддельных видео становится все более распространенным и затрудняет процесс проверки подлинности контента.

В данной работе ансамбль моделей показал высокую способность распознавания изменений в лицевой мимике, что делает его эффективным инструментом для борьбы с поддельными видеозаписями. Но с развитием технологий появляются новые методы создания реалистичных подделок, которые могут обойти существующие модели детекции.

Таким образом, необходимо постоянное развитие и улучшение методов детекции DeepFake технологий. Это включает в себя разработку новых моделей, улучшение существующих алгоритмов и расширение набора данных для обучения моделей. Кроме того, важно обеспечить доступность и простоту использования этих методов для широкого круга пользователей, чтобы сделать процесс борьбы с поддельными видеозаписями более эффективным и доступным для всех.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Гудфеллоу Я., Бенджио И., Курвилль А. − Глубокое обучение, 2017
2. Адриан Роузброк − Deep Learning for Computer Vision with Python, 2017
3. Lyon B.- Exploring Deepfakes. Deploy powerful AI techniques..., 2023
4. https://arxiv.org/abs/2006.16228 (04.03.2024)
5. https://arxiv.org/abs/1811.08383 (04.03.2024)
6. https://habr.com/ru/companies/first/articles/707246/ (08.05.2024)

ПРИЛОЖЕНИЕ А

1. Необходимые библиотеки

absl-py==0.12.0

antlr4-python3-runtime==4.8

appdirs==1.4.4

argon2-cffi==21.1.0

astunparse==1.6.3

async-generator==1.10

attrs==21.2.0

audioread==2.1.9

backcall==0.2.0

beautifulsoup4==4.11.1

bitarray==2.4.1

bleach==4.1.0

cached-property==1.5.2

cachetools==4.2.2

click==8.0.4

cmake==3.22.3

colorama==0.4.4

cycler==0.10.0

editdistance==0.6.0

entrypoints==0.3

ffmpeg-bitrate-stats==0.2.2

filelock==3.4.1

flatbuffers==1.12

future==0.18.2

gast==0.3.3

gdown==4.5.1

gitdb==4.0.9

GitPython==3.1.18

google-auth==1.33.0

google-auth-oauthlib==0.4.4

grpcio==1.32.0

h5py==2.10.0

hydra-core==1.0.7

idna==3.2

jax==0.2.17

jaxlib==0.1.68+cuda101

jedi==0.18.0

Jinja2==3.0.2

joblib==1.0.1

jsonschema==3.2.0

kaleido==0.2.1

keras-nightly==2.5.0.dev2021032900

Keras-Preprocessing==1.1.2

kiwisolver==1.3.1

libclang==11.1.0

librosa==0.8.1

llvmlite==0.36.0

Markdown==3.3.4

MarkupSafe==2.0.1

mediapipe==0.8.3

mistune==0.8.4

moviepy==1.0.3

nbclient==0.5.4

nbconvert==6.0.7

nbformat==5.1.3

nest-asyncio==1.5.1

networkx==2.5.1

notebook==6.4.5

numba==0.53.1

numpy==1.19.3

oauthlib==3.1.1

omegaconf==2.0.6

opencv-python==4.5.3.56

opencv-python-headless==4.5.3.56

opt-einsum==3.3.0

optax==0.0.9

packaging==21.0

pandas==1.1.5

pandocfilters==1.5.0

parso==0.8.2

pathtools==0.1.2

pexpect==4.8.0

pickleshare==0.7.5

Pillow==8.3.1

pkg_resources==0.0.0

pooch==1.5.2

protobuf==3.17.3

psutil==5.9.1

ptyprocess==0.7.0

pyasn1==0.4.8

pyasn1-modules==0.2.8

pycparser==2.20

pydub==0.25.1

Pygments==2.10.0

pyparsing==2.4.7

QtPy==1.11.2

qudida==0.0.4

regex==2022.3.15

requests==2.26.0

requests-oauthlib==1.3.0

resampy==0.2.2

rsa==4.7.2

sacrebleu==2.0.0

scikit-image==0.17.2

scikit-learn==0.24.2

scikit-video==1.1.11

scipy==1.4.1

Send2Trash==1.8.0

sentencepiece==0.1.96

sentry-sdk==1.9.2

setproctitle==1.2.3

shortuuid==1.0.9

SimpleITK==2.1.1

six==1.15.0

sklearn==0.0

smmap==5.0.0

SoundFile==0.10.3.post1

soupsieve==2.3.2.post1

tabulate==0.8.9

tb-nightly==2.6.0a20210806

tenacity==8.0.1

tensorboard==2.5.0

tensorboard-data-server==0.6.1

tensorboard-plugin-wit==1.8.0

tensorflow==2.3.0

tensorflow-datasets==4.3.0

tensorflow-estimator==2.3.0

tensorflow-hub==0.12.0

tensorflow-io==0.16.0

tensorflow-metadata==1.1.0

termcolor==1.1.0

terminado==0.12.1

testpath==0.5.0

tf-estimator-nightly==2.7.0.dev2021080901

threadpoolctl==2.2.0

tifffile==2020.9.3

toolz==0.11.1

tornado==6.1

tqdm==4.61.2

traitlets==4.3.3

typing-extensions==3.7.4.3

urllib3==1.26.11

wcwidth==0.2.5

webencodings==0.5.1

Werkzeug==2.0.1

widgetsnbextension==3.5.1

wrapt==1.12.1

zipp==3.5.0

2. Материалы для обучения и проверки

1) HowTo100M - https://huggingface.co/datasets/HuggingFaceM4/howto100m
2) AudioSets - https://research.google.com/audioset/
3) FaceForensics++ - https://github.com/ondyari/FaceForensics/tree/master/dataset.

4) DeepFake Detection Challenge - https://www.kaggle.com/c/deepfake-detection-challenge/data

ПРИЛОЖЕНИЕ Б

1. Листинг структуры TSM-50(x2)

```
class TSMResNet(hk.Module):

  def __init__(self,
               output_channels: int,
               stride: int,
               use_projection: bool,
               tsm_mode: str,
               normalize_fn: Optional[types.NormalizeFn] = None,
               channel_shift_fraction: float = 0.125,
               num_frames: int = 8,
               name: str = 'TSMResNetBlock'):
    super().__init__(name=name)
    self._output_channels = output_channels
    self._bottleneck_channels = output_channels // 4
    self._stride = stride
    self._use_projection = use_projection
    self._normalize_fn = normalize_fn
    self._tsm_mode = tsm_mode
    self._channel_shift_fraction = channel_shift_fraction
    self._num_frames = num_frames

  def __call__(self,
               inputs: types.TensorLike,
               is_training: bool = True) -> jnp.ndarray:
    preact = inputs
    if self._normalize_fn is not None:
      preact = self._normalize_fn(preact, is_training=is_training)
    preact = jax.nn.relu(preact)

    if self._use_projection:
      shortcut = hk.Conv2D(
          output_channels=self._output_channels,
          kernel_shape=1,
          stride=self._stride,
          with_bias=False,
          padding='SAME',
          name='shortcut_conv')(
              preact)
```

```
else:
  shortcut = inputs

if self._channel_shift_fraction != 0:
  preact = tsmu.apply_temporal_shift(
      preact, tsm_mode=self._tsm_mode, num_frames=self._num_frames,
      channel_shift_fraction=self._channel_shift_fraction)

residual = hk.Conv2D(
    self._bottleneck_channels,
    kernel_shape=1,
    stride=1,
    with_bias=False,
    padding='SAME',
    name='conv_0')(
        preact)

if self._normalize_fn is not None:
  residual = self._normalize_fn(residual, is_training=is_training)
residual = jax.nn.relu(residual)
residual = hk.Conv2D(
    output_channels=self._bottleneck_channels,
    kernel_shape=3,
    stride=self._stride,
    with_bias=False,
    padding='SAME',
    name='conv_1')(
        residual)
if self._normalize_fn is not None:
  residual = self._normalize_fn(residual, is_training=is_training)
residual = jax.nn.relu(residual)
residual = hk.Conv2D(
    output_channels=self._output_channels,
    kernel_shape=1,
    stride=1,
    with_bias=False,
    padding='SAME',
    name='conv_2')(
        residual)

output = shortcut + residual
return output
```

2. Листинг структуры ResNet50

Предварительно были соединены блоки BootleNeck (для оптимизации времени работы системы) и базовый блок, осуществим конкатенацию.

```
class ResNet(hk.Module):
  VALID_ENDPOINTS = (
      'resnet_stem',
      'resnet_unit_0',
      'resnet_unit_1',
      'resnet_unit_2',
      'resnet_unit_3',
      'last_conv',
      'output',
  )

  def __init__(self,
               depth=50,
               num_classes: Optional[int] = 1000,
               width_mult: int = 1,
               normalize_fn: Optional[types.NormalizeFn] = None,
               name: Optional[Text] = None,
               remat: bool = False):
    super(ResNetV2, self).__init__(name=name)
    self._normalize_fn = normalize_fn
    self._num_classes = num_classes
    self._width_mult = width_mult

    self._strides = [1, 2, 2, 2]
    num_blocks = {
        18: [2, 2, 2, 2],
        34: [3, 4, 6, 3],
        50: [3, 4, 6, 3],
        101: [3, 4, 23, 3],
        152: [3, 8, 36, 3],
        200: [3, 24, 36, 3],
    }
    if depth not in num_blocks:
      raise ValueError(
          f'`depth` should be in {list(num_blocks.keys())} ({depth} given).')
    self._num_blocks = num_blocks[depth]
```

```
  if depth >= 50:
    self._block_module = BottleneckBlock
    self._channels = [256, 512, 1024, 2048]
  else:
    self._block_module = BasicBlock
    self._channels = [64, 128, 256, 512]

  self._initial_conv = hk.Conv2D(
      output_channels=64 * self._width_mult,
      kernel_shape=7,
      stride=2,
      with_bias=False,
      padding='SAME',
      name='initial_conv')

  if remat:
    self._initial_conv = hk.remat(self._initial_conv)

  self._block_groups = []
  for i in range(4):
    self._block_groups.append(
        ResNetUnit(
            channels=self._channels[i] * self._width_mult,
            num_blocks=self._num_blocks[i],
            block_module=self._block_module,
            stride=self._strides[i],
            normalize_fn=self._normalize_fn,
            name='block_group_%d' % i,
            remat=remat))

  if num_classes is not None:
    self._logits_layer = hk.Linear(
        output_size=num_classes, w_init=jnp.zeros, name='logits')

def __call__(self, inputs, is_training, final_endpoint='output'):
  self._final_endpoint = final_endpoint
  net = self._initial_conv(inputs)
  net = hk.max_pool(
      net, window_shape=(1, 3, 3, 1),
      strides=(1, 2, 2, 1),
      padding='SAME')
  end_point = 'resnet_stem'
```

```
    if self._final_endpoint == end_point:
      return net

    for i_group, block_group in enumerate(self._block_groups):
      net = block_group(net, is_training=is_training)
      end_point = f'resnet_unit_{i_group}'
      if self._final_endpoint == end_point:
        return net

    end_point = 'last_conv'
    if self._final_endpoint == end_point:
      return net

    if self._normalize_fn is not None:
      net = self._normalize_fn(net, is_training=is_training)
      net = jax.nn.relu(net)
    net = jnp.mean(net, axis=[1, 2])

    assert self._final_endpoint == 'output'
    if self._num_classes is None:
      return net

    return self._logits_layer(net)
```

3. Листинг структуры ансамбля ResNet50 и TSM-50(x2)

```
class TSMResNetUnit(hk.Module):
  def __init__(self,
               output_channels: int,
               num_blocks: int,
               stride: int,
               tsm_mode: str,
               num_frames: int,
               normalize_fn: Optional[types.NormalizeFn] = None,
               channel_shift_fraction: float = 0.125,
               name: str = 'tsm_resnet_unit'):
    super().__init__(name=name)
    self._output_channels = output_channels
    self._num_blocks = num_blocks
    self._normalize_fn = normalize_fn
    self._stride = stride
    self._tsm_mode = tsm_mode
    self._channel_shift_fraction = channel_shift_fraction
```

```
    self._num_frames = num_frames

  def __call__(self,
               inputs: types.TensorLike,
               is_training: bool) -> jnp.ndarray:
    net = inputs
    for idx_block in range(self._num_blocks):
      net = TSMResNetBlock(
          self._output_channels,
          stride=self._stride if idx_block == 0 else 1,
          use_projection=idx_block == 0,
          normalize_fn=self._normalize_fn,
          tsm_mode=self._tsm_mode,
          channel_shift_fraction=self._channel_shift_fraction,
          num_frames=self._num_frames,
          name=f'block_{idx_block}')(
              net, is_training=is_training)
    return net

class TSMResNetV2(hk.Module):
  """TSM on ResNet V2 as in https://arxiv.org/abs/1603.05027."""
  VALID_ENDPOINTS = (
      'tsm_resnet_stem',
      'tsm_resnet_unit_0',
      'tsm_resnet_unit_1',
      'tsm_resnet_unit_2',
      'tsm_resnet_unit_3',
      'last_conv',
      'Embeddings',
  )

  def __init__(self,
               normalize_fn: Optional[types.NormalizeFn] = None,
               depth: int = 50,
               num_frames: int = 16,
               channel_shift_fraction: float = 0.125,
               width_mult: int = 1,
               name: str = 'TSMResNetV2'):

    super().__init__(name=name)

    if not 0. <= channel_shift_fraction <= 1.0:
```

```
    raise ValueError(
        f'channel_shift_fraction ({channel_shift_fraction})'
        ' has to be in [0, 1].')

  self._num_frames = num_frames

  self._channels = (256, 512, 1024, 2048)
  self._strides = (1, 2, 2, 2)

  num_blocks = {
      50: (3, 4, 6, 3),
      101: (3, 4, 23, 3),
      152: (3, 8, 36, 3),
      200: (3, 24, 36, 3),
  }
  if depth not in num_blocks:
    raise ValueError(
        f'`depth` should be in {list(num_blocks.keys())} ({depth} given).')
  self._num_blocks = num_blocks[depth]

  self._width_mult = width_mult
  self._channel_shift_fraction = channel_shift_fraction
  self._normalize_fn = normalize_fn

def __call__(
    self,
    inputs: types.TensorLike,
    is_training: bool = True,
    final_endpoint: str = 'Embeddings') -> jnp.ndarray:

  inputs, tsm_mode, num_frames = tsmu.prepare_inputs(inputs)
  num_frames = num_frames or self._num_frames

  self._final_endpoint = final_endpoint
  if self._final_endpoint not in self.VALID_ENDPOINTS:
    raise ValueError(f'Unknown final endpoint {self._final_endpoint}')

  # Stem convolution.
  end_point = 'tsm_resnet_stem'
  net = hk.Conv2D(
      output_channels=64 * self._width_mult,
      kernel_shape=7,
```

```
    stride=2,
    with_bias=False,
    name=end_point,
    padding='SAME')(
        inputs)
net = hk.MaxPool(
    window_shape=(1, 3, 3, 1),
    strides=(1, 2, 2, 1),
    padding='SAME')(
        net)
if self._final_endpoint == end_point:
  return net

for unit_id, (channels, num_blocks, stride) in enumerate(
    zip(self._channels, self._num_blocks, self._strides)):
  end_point = f'tsm_resnet_unit_{unit_id}'
  net = TSMResNetUnit(
      output_channels=channels * self._width_mult,
      num_blocks=num_blocks,
      stride=stride,
      normalize_fn=self._normalize_fn,
      channel_shift_fraction=self._channel_shift_fraction,
      num_frames=num_frames,
      tsm_mode=tsm_mode,
      name=end_point)(
          net, is_training=is_training)
  if self._final_endpoint == end_point:
    return net

if self._normalize_fn is not None:
  net = self._normalize_fn(net, is_training=is_training)
net = jax.nn.relu(net)

end_point = 'last_conv'
if self._final_endpoint == end_point:
  return net
net = jnp.mean(net, axis=(1, 2))
net = tsmu.prepare_outputs(net, tsm_mode, num_frames)
assert self._final_endpoint == 'Embeddings'
return net
```