Задание:

Цель: получить базовые навыки работы с одной из библиотек глубокого обучения (Caffe, Torch, TensorFlow или MXNet на выбор) на примере сверточных нейронных сетей.

Выполнение задания:

1. В ходе данного проекта я освоила базовые навыки работы с библиотекой

глубокого обучения TensorFlow на примере построения и обучения полностью

связанных нейронных сетей для задачи многоклассовой классификации.

Постановка математической задачи может быть описана на примере простой

сверточной сети, с одном сверточным слоем, активацией ReLU.

Пусть входное изображение имеет размерность H×W×C, где:

• H — высота изображения,

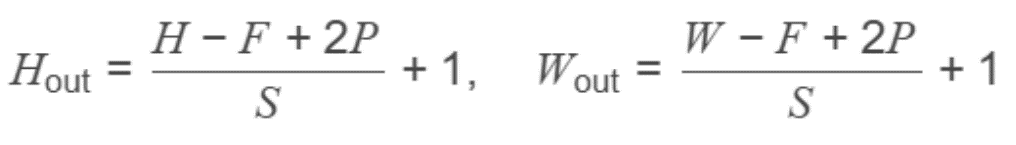
• W — ширина изображения,

• C — количество каналов (например, 3 для RGB).

Обозначим входное изображение как X, где X∈RH×W×C.

Свертка применяется к входу X с фильтром W размером F×F×C. Пусть у нас есть

K фильтров. Тогда выходная размерность карты признаков будет:



Где:

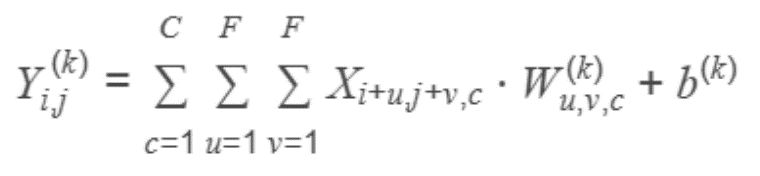
• F — размер фильтра (например, 3×3),

• P — размер паддинга (обычно P=0 или P=1),

• S — шаг фильтра (stride),

• Hout и Wout — высота и ширина выходной карты признаков.

Для каждого фильтра k, выходные данные вычисляются так:



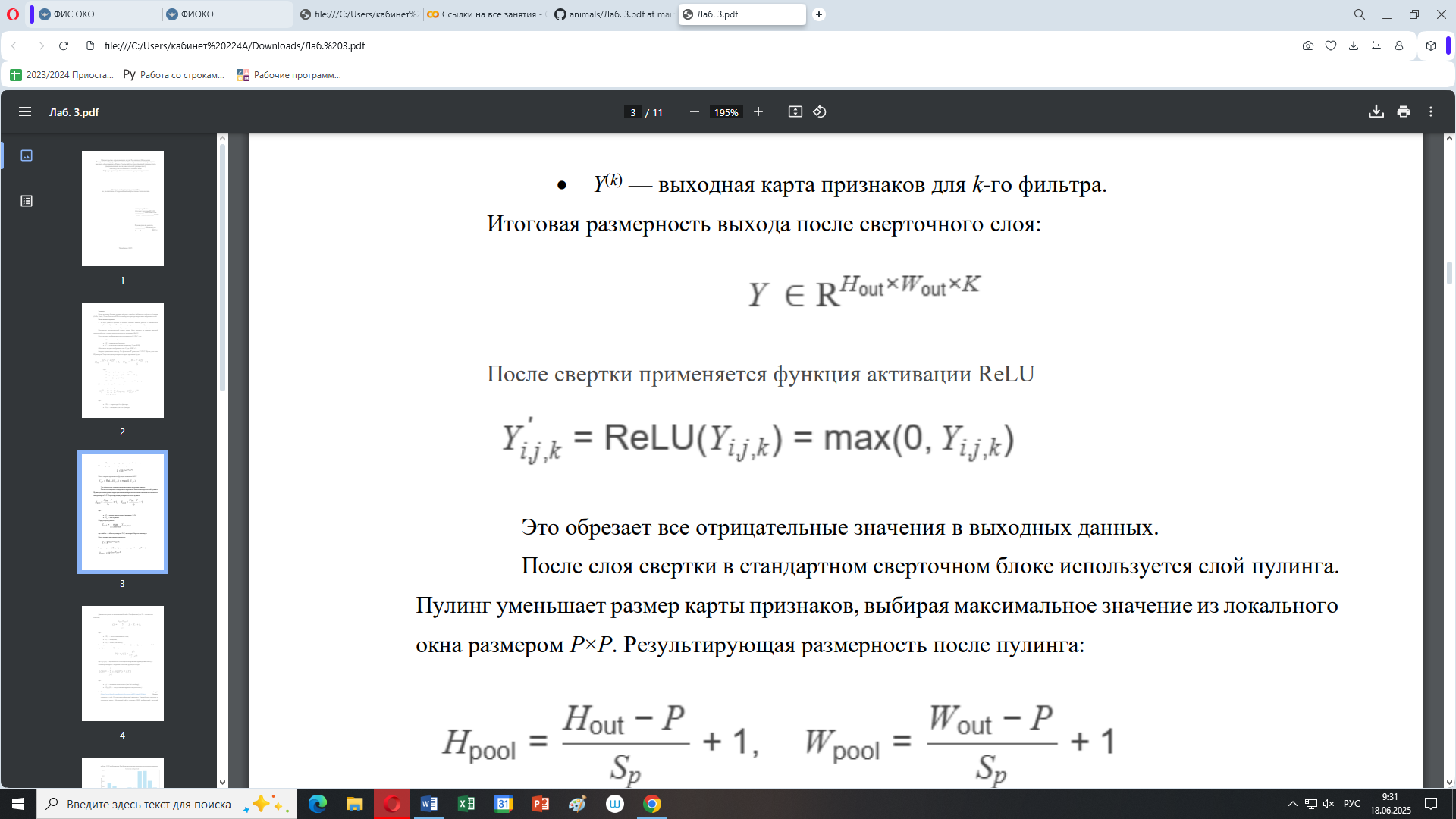
где:

• W(k) — параметры k-го фильтра,

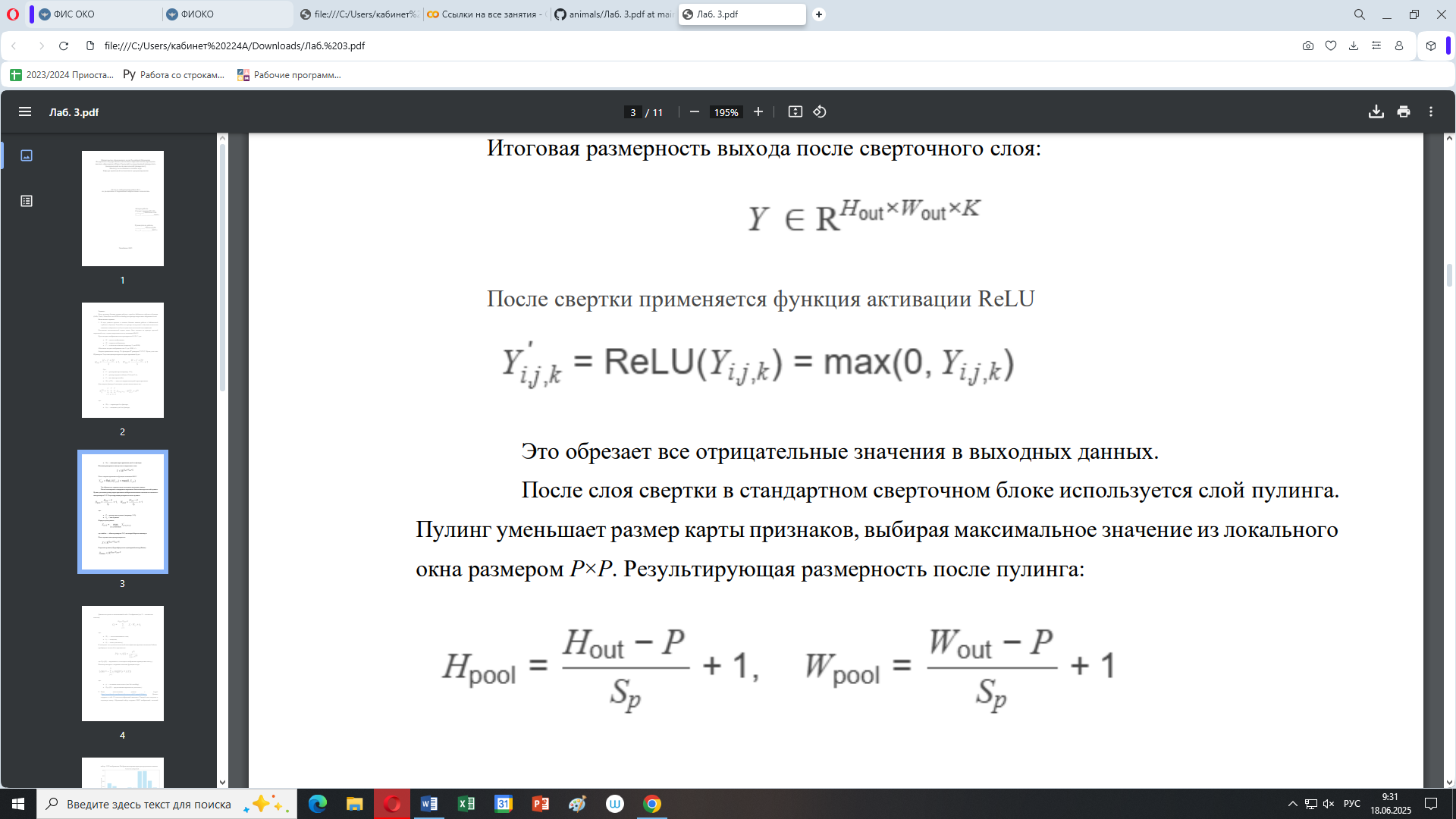
• b (k) — смещение для k-го фильтра,

• Y (k) — выходная карта признаков для k-го фильтра.

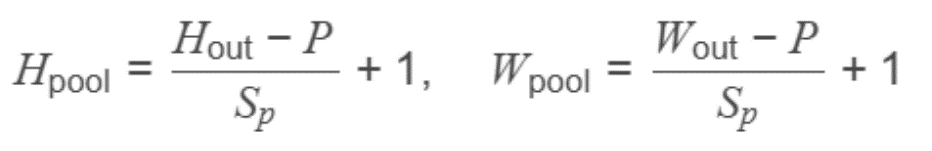
Итоговая размерность выхода после сверточного слоя:



После свертки применяется функция активации ReLU



Это обрезает все отрицательные значения в выходных данных. После слоя свертки в стандартном сверточном блоке используется слой пулинга. Пулинг уменьшает размер карты признаков, выбирая максимальное значение из локального окна размером P×P. Результирующая размерность после пулинга:



где:

• P — размер окна пулинга (например, 2×2),

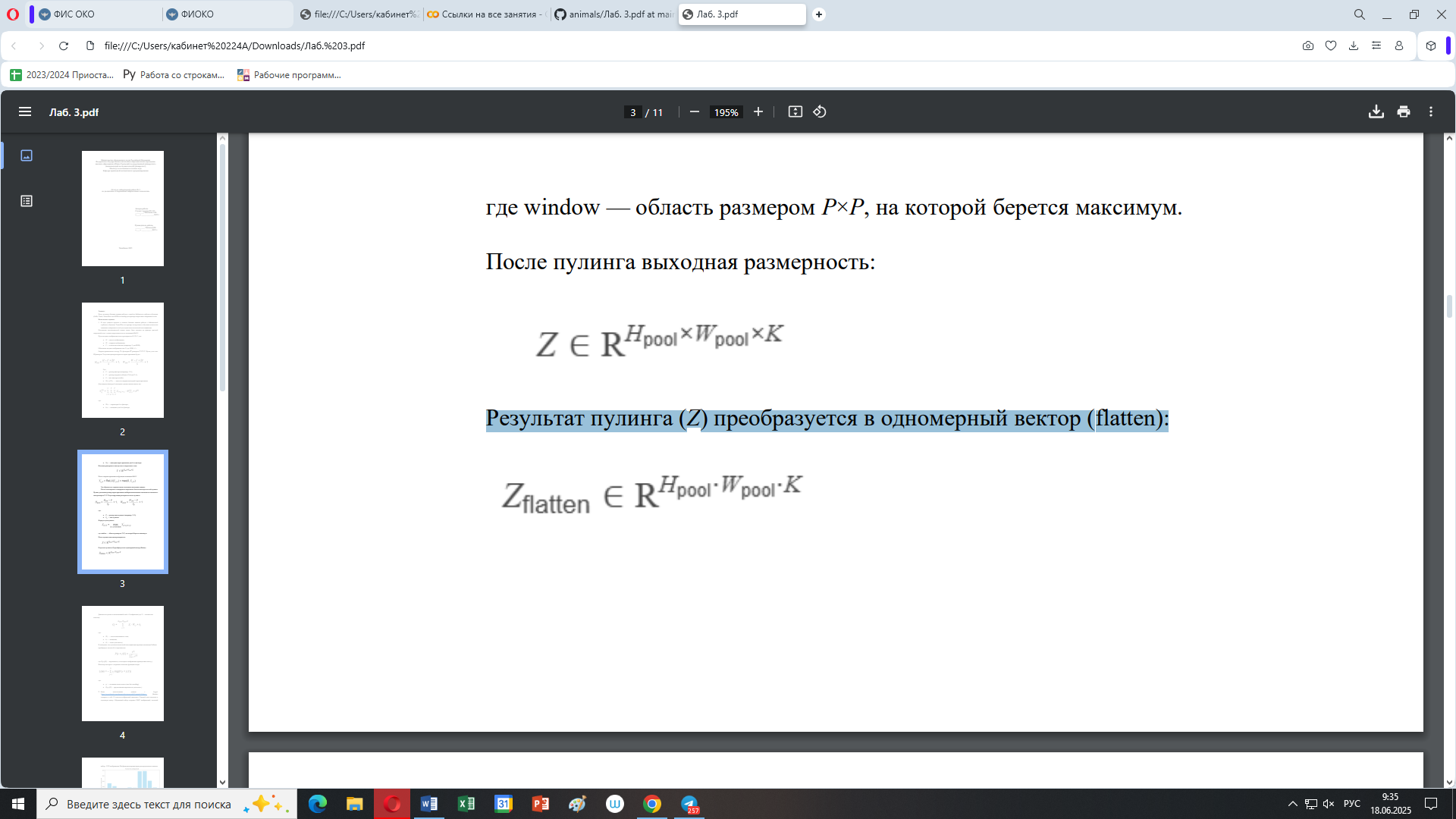
• Sp — шаг пулинга. Формула для пулинга:



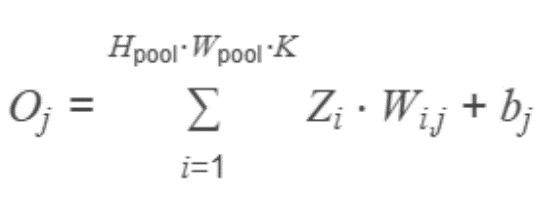
где window — область размером P×P, на которой берется максимум. После пулинга выходная размерность:



Результат пулинга (Z) преобразуется в одномерный вектор (flatten):



Данные поступают в полносвязный слой с N нейронами (где N — количество классов):

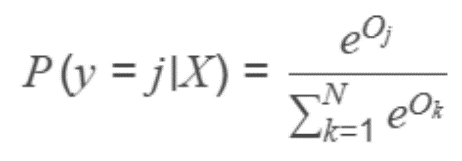


где:

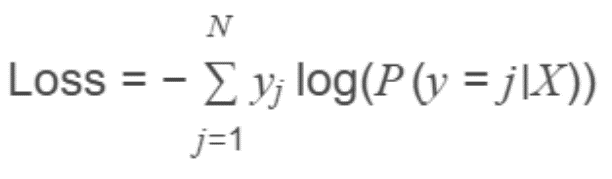
• Wi,j — веса полносвязного слоя,

• bj — смещения,

• Oj — логит для класса j. В выходном слое для многоклассовой классификации функция активации Softmax преобразует логиты Oj в вероятности:



где P(y=j∣X) — вероятность, что входное изображение принадлежит классу j. Используется кросс- энтропия в качестве функции потерь:

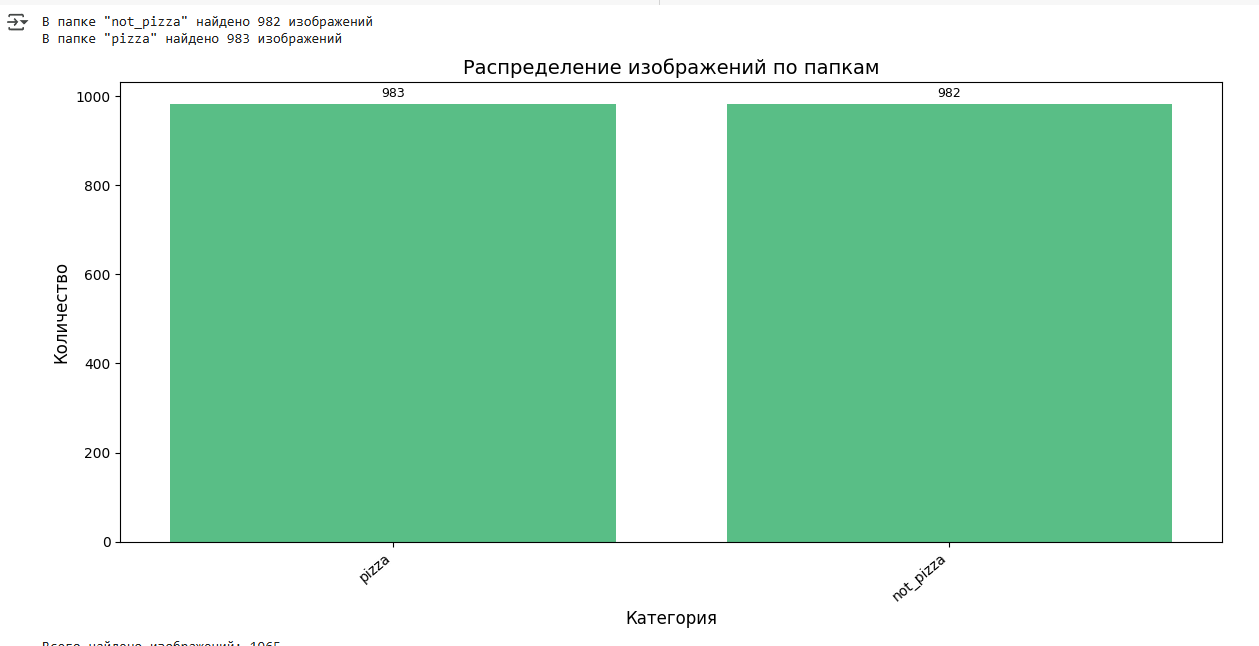


где:

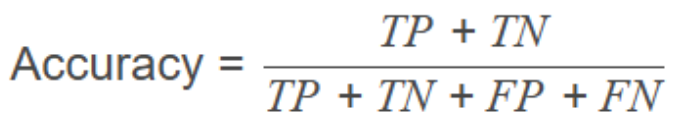
• yj — истинная метка класса (one-hot encoding),

• P(y=j∣X) — предсказанная вероятность для класса j.

Были использованы данные подмножество набора данных Food-101. Боссард, Лукас, Матье Гийомен и Люк Ван Гул. «Food-101 – Mining Discriminative Components with Random Forests». Изображения разделены на две папки: pizza и not\_pizza, имеющее равное количество изображений (по 983 шт.). Все изображения были масштабированы таким образом, чтобы максимальная длина стороны составляла 512 пикселей (однако в коде длина стороны была уменьшена для экономии ОЗУ). Содержание папок соответствует их названию. Данные разделены: Количество обучающих изображений: 1376 Количество валидационных изображений: 294 Количество тестовых изображений: 296.



При тестировании моделей была использована метрика качества Accuracy, которая определяется следующим образом:



где:

- TP (True Positives) — количество истинных положительных предсказаний (правильно классифицированные объекты класса 1).

- TN (True Negatives) — количество истинных отрицательных предсказаний (правильно классифицированные объекты класса 0).

- FP (False Positives) — количество ложных положительных предсказаний (объекты класса 0, ошибочно классифицированные как класс 1).

- FN (False Negatives) — количество ложных отрицательных предсказаний (объекты класса 1, ошибочно классифицированные как класс 0).

Я воспользовалась встроенными функциями ImageDataGenerator и flow\_from\_directory для создания датасета. На вход подавались пути к изображениям и сами изображения, на выходе формировались батчи (пакеты) изображений вместе с соответствующими метками классов (массивами изображений и их меток).

Для обучения нейронной сети я использовала стандартную функцию fit. После завершения обучения визуализировала процесс — строила графики метрик и ошибок для тренировочной и валидационной выборок.

Полносвязная сеть из 4х слоев (3 скрытых), использовалась как базовая, плюс различные функции активации скрытых слоев:

def build\_simple\_model(input\_shape, num\_classes):

    model = Sequential([

        Flatten(input\_shape=input\_shape),

        Dense(100, activation='tanh'),

        Dense(50, activation='relu'),

        Dense(25, activation='relu'),

        Dense(num\_classes, activation='softmax')

    ])

    return model

model\_cpu = build\_simple\_model(input\_shape, num\_classes)

model\_cpu.compile(

    optimizer=Adam(learning\_rate=0.001),

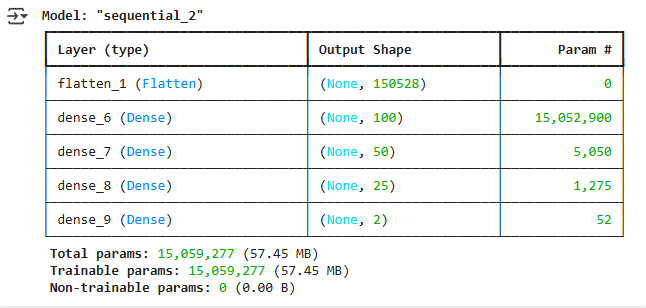
    loss='categorical\_crossentropy',

    metrics=['accuracy']

)

model\_cpu.summary()

Визуальная схема:



Модель не учится — точность и потери стагнируют, метрики не улучшаются с эпохами. Вероятно, сеть упёрлась в базовое значение точности (примерно 66.7%), что может означать следующее: Данные несбалансированны, и сеть просто всегда предсказывает самый частый класс. Слишком простая архитектура, не хватает слоёв/нейронов — модель не способна уловить закономерности. Входные данные некорректно размечены или обработаны. Отсутствие переобучения — метрики на обучении и валидации практически идентичны.

Для следующей модели два первых слоя были заменены на свертки и пулинги:

def model\_cnn\_2(input\_shape, num\_classes):

    model = Sequential([

        Input(shape=input\_shape),

        Conv2D(32, (3, 3), activation="relu"),

        MaxPooling2D(pool\_size=(2, 2)),

        Conv2D(64, (3, 3), activation="relu"),

        MaxPooling2D(pool\_size=(2, 2)),

        Flatten(),

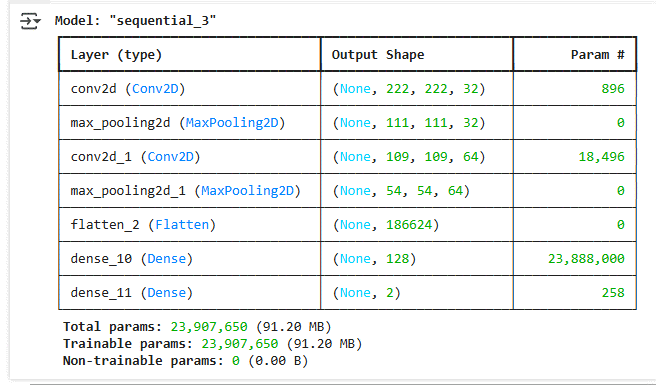
        Dense(128, activation="relu"),

        Dense(num\_classes, activation="softmax")

    ])

    return model

Визуальная схема:

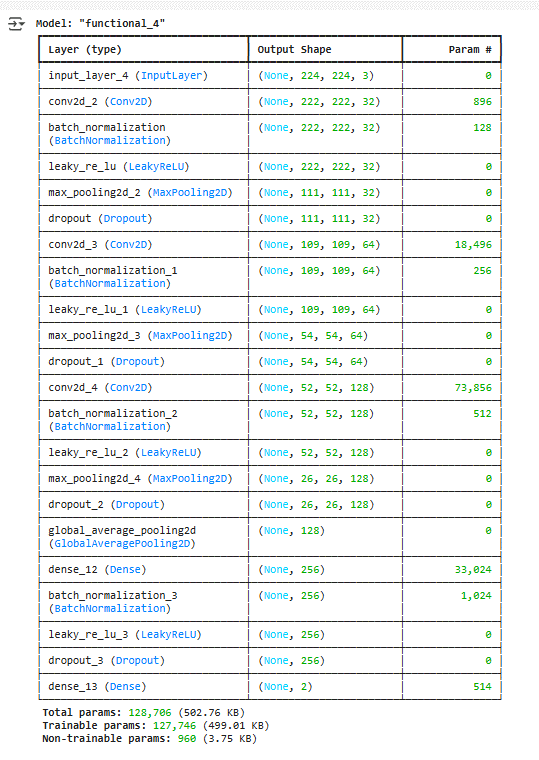


Вывод: Модель не обучается: изменения точности минимальны, значения держатся примерно на уровне случайного угадывания (0.666 ≈ 2/3, возможно задача на 3 класса). Потери стабилизировались слишком рано, графики очень "плоские".

Добавим еще один слой свертки. Для удобства выделим отдельно сверточный блок. Также добавим обнуление нейронов(Dropout) и нормализацию данных.



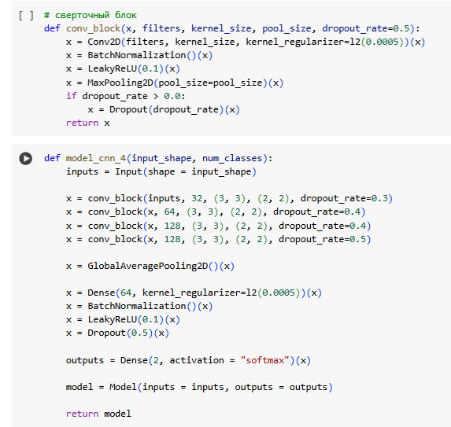
Визуальная схема:



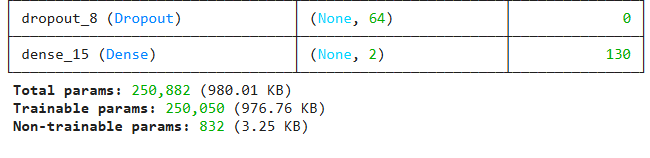
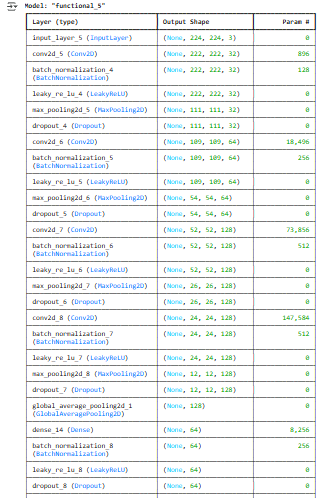
Выводы

* Модель постепенно учится и становится точнее на обучающей выборке.
* Точность на валидации улучшается только к концу процесса обучения, при этом разброс показывает, что результат может страдать из-за случайности или дисбаланса данных.
* Потери валидации ведут себя нестабильно, что обычно указывает на необходимость либо увеличить объём данных, либо поработать с архитектурой/гиперпараметрами.

Добавим l2 регуляризацию в сверточный слой, добавим GlobalAveragePooling2D перед полносвязными слоями, сделаем разным исключение нейроннов на разных слоях (значение dropout).

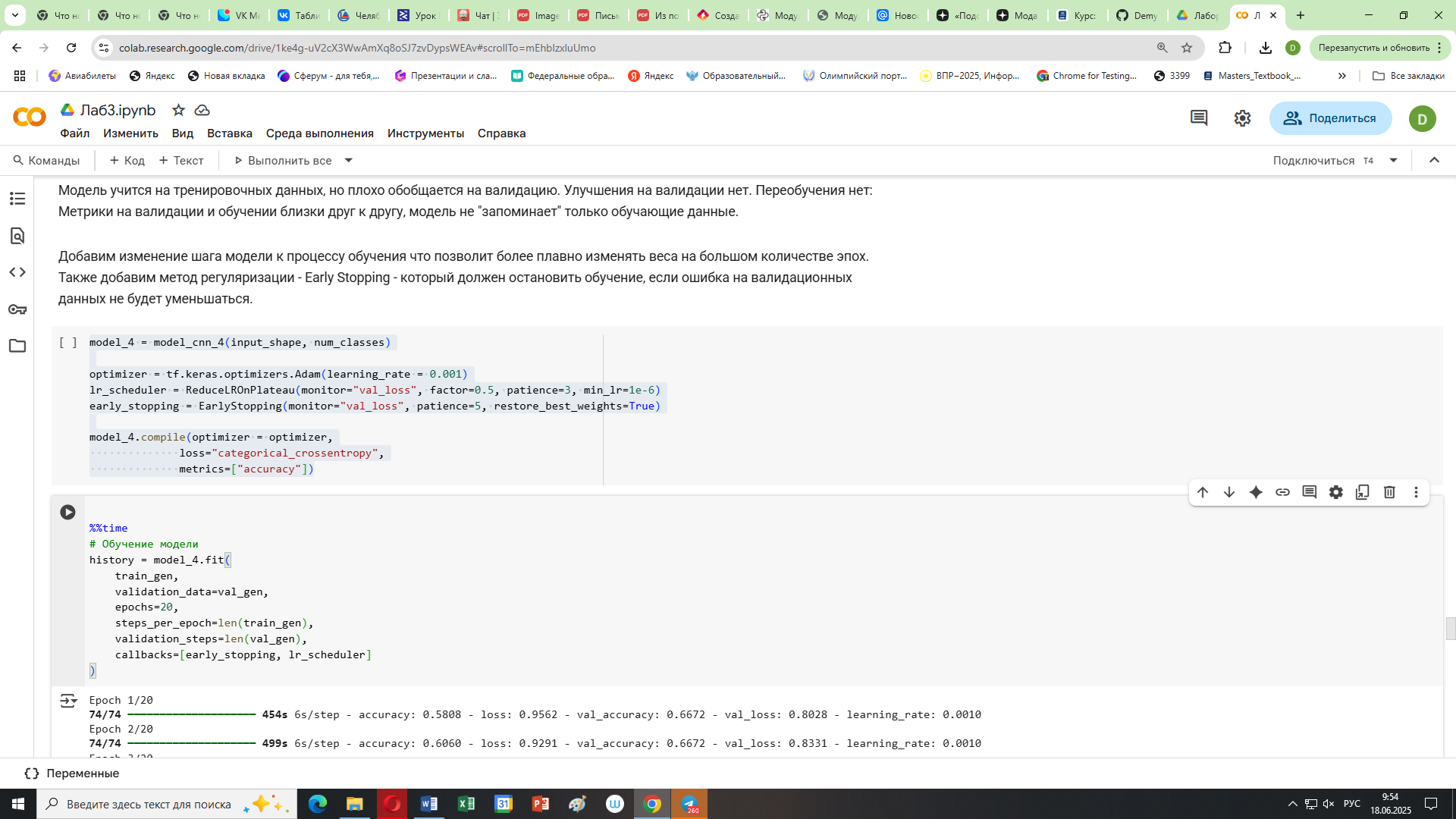


Визуальная схема

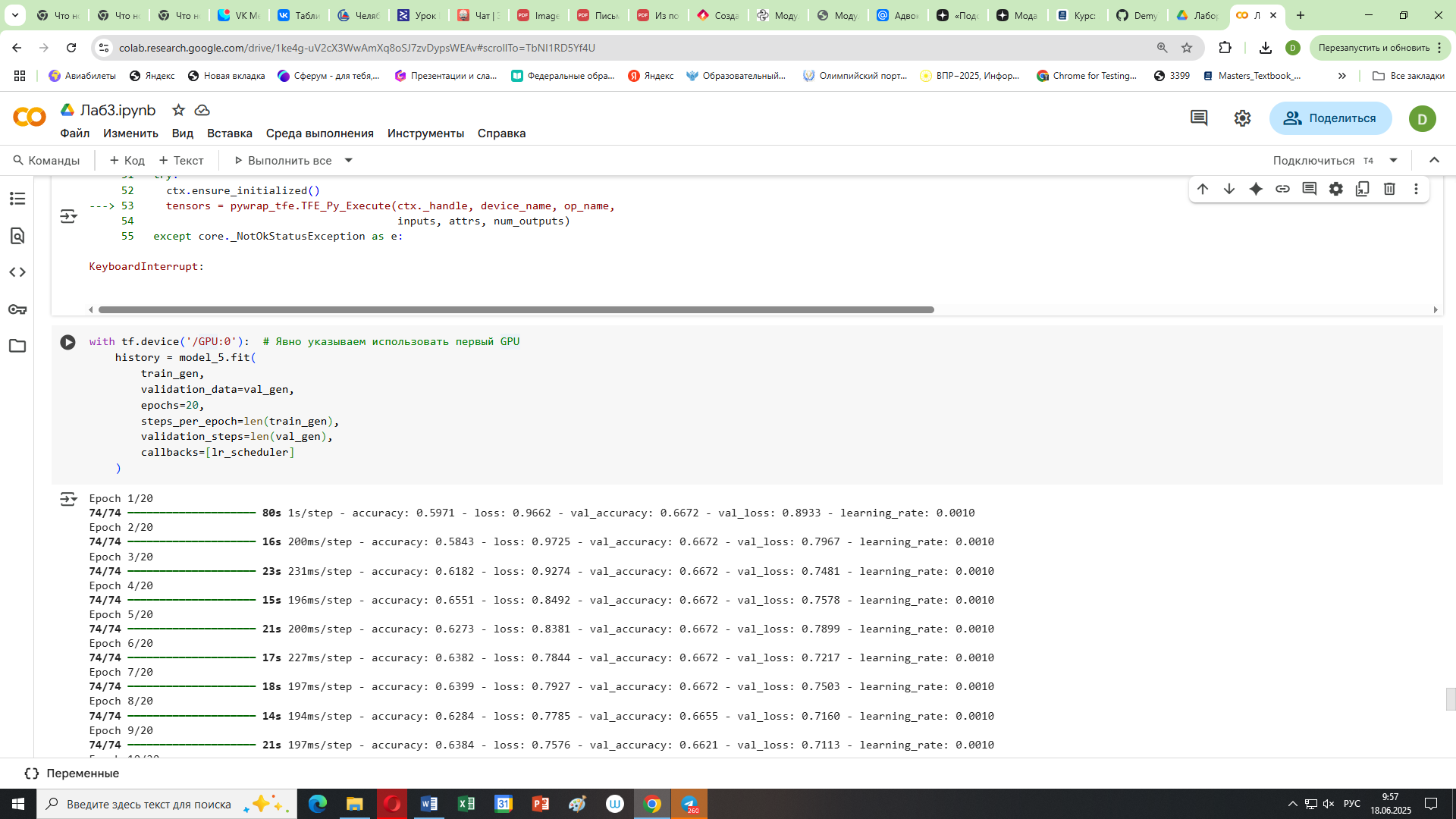


Модель учится на тренировочных данных, но плохо обобщается на валидацию. Улучшения на валидации нет. Переобучения нет: Метрики на валидации и обучении близки друг к другу, модель не "запоминает" только обучающие данные.

Добавим изменение шага модели к процессу обучения что позволит более плавно изменять веса на большом количестве эпох. Также добавим метод регуляризации - Early Stopping - который должен остановить обучение, если ошибка на валидационных данных не будет уменьшаться.



Модель стабильно обучается на тренировочных данных: Видно улучшение точности и снижение потерь на обучении. На валидации метрика "застряла". Это явный сигнал, что модель не учится различать классы на новых данных, а возможно просто всегда выбирает самый частый класс. Переобучения нет: Потери и точность по обучению и валидации близки, кривые идут рядом.



Модель учится на тренировочных данных: точность растет, потери падают. Обобщающая способность низкая: на валидации точность стабильна и, скорее всего, отражает уровень угадывания (например, задача с 3 классами и несбалансированными метками). Переобучения нет: разница между метриками обучения и валидации небольшая, потери и точности к концу обучения почти сравнялись.

Вывод:

Рост точности только на обучении: По графикам видно, что точность на обучающем наборе увеличивалась с каждой эпохой, а вот на валидации оставалась практически неизменной и находилась на уровне случайного угадывания (примерно 0.667).

Потери снижаются у обеих метрик: Потери (loss) на тренировке и валидации плавно уменьшаются, но горизонтальности и "пилы" на графике валидации указывают на то, что модель не извлекает дополнительную полезную информацию при обучении.

Переобучения не наблюдается: Параметры на валидации и обучении отличаются очень не сильно — это значит, что модель не запоминает только тренировочные данные, но и не учится чему-то новому для других примеров.