Министерство науки и высшего образования Российской Федерации

НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ (НИ ТГУ)

Институт прикладной информатики и компьютерных наук

ОТЧЕТ

По дисциплине «Нейронные сети»

Задание 1 «Обучение простых полносвязных нейросетевых моделей прямого распространения»

Студент

Группы 932101

Насутион Рафли Аулиа Ризки

Томск – 2024

# **Цель работы**

получить навыки создания полносвязного бинарного и многоклассового классификаторов, а также нейросетевого регрессора с помощью библиотек построения нейронных сетей.

# **Задачи**

Построить три нейросетевые модели:

a) Бинарный Классификатор

b) Многоклассовый классификатор

c) Регрессор

# **1. Описание наборов данных.**

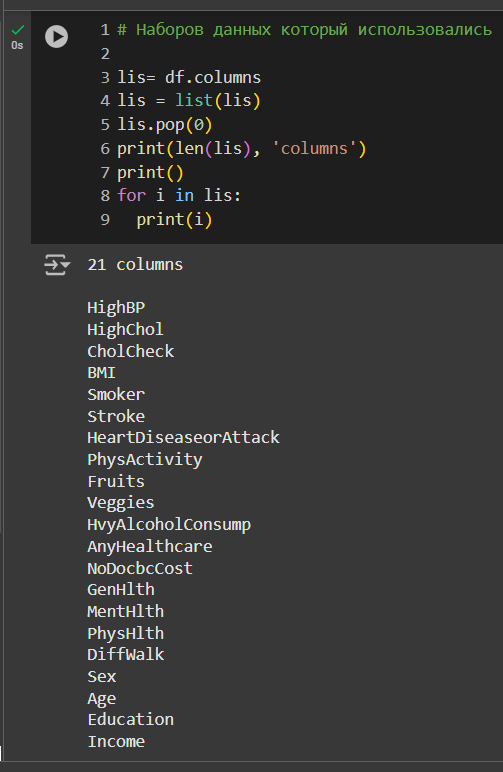
В данной работе использовались следующие наборы данных:

1. Оценка вероятности диагностики диабета у человека (Diabetes Diagnosis Probability Dataset):

* Источник:

<https://www.kaggle.com/datasets/alexteboul/diabetes-health-indicators-dataset> Kaggle (diabetes\_012\_health\_indicators\_BRFSS2015.csv).

* Описание:  
  Набор данных содержит информацию о здоровье и образе жизни людей. Он включает 22 признака, такие как индекс массы тела (BMI), наличие высоких показателей артериального давления и холестерина, уровень физической активности, курение и другие факторы.
* Цель:  
  Предсказать вероятность наличия у человека диабета. Класс 0 соответствует отсутствию диабета, класс 1 — преддиабетическому состоянию, а класс 2 — диабету.

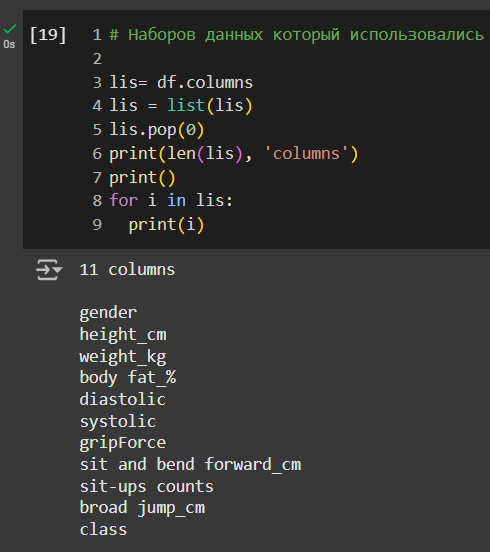


1. Оценка уровня физического развития людей (Physical Performance Dataset):

* Источник:

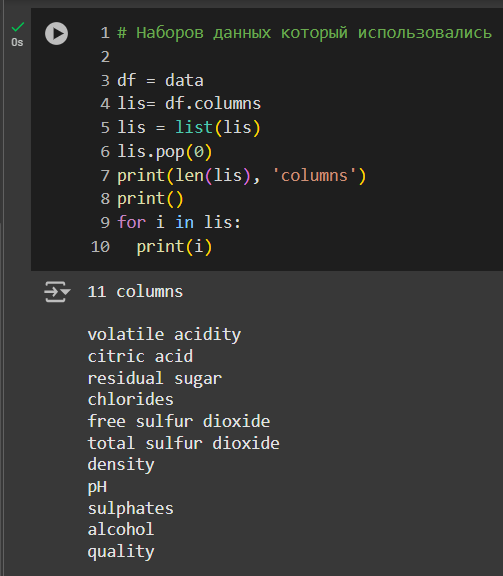
<https://www.kaggle.com/datasets/kukuroo3/body-performance-data> Метка класса: class. Kaggle (body-performance-data).

* Описание:  
  Этот набор данных содержит физические характеристики, такие как возраст, рост, вес, процент жира в теле, сила хвата и другие параметры, которые используются для оценки уровня физической производительности.
* Цель:  
  Классифицировать физическую производительность человека по уровням (например, low, medium, high).



1. Качество вина (Wine Quality Dataset):

* Источник: <https://archive.ics.uci.edu/ml/datasets/Wine+Quality> UCI Machine Learning Repository (winequality-white.csv).
* Описание:  
  Набор данных включает химические свойства вина, такие как кислотность, уровень сахара, хлориды и другие параметры. Используется для прогнозирования качества вина по шкале от 0 до 10.
* Цель:  
  Построить регрессионную модель для предсказания оценки качества вина.



# **2. Признаки, которые были использованы для анализа.**

a. Диабет (Diabetes Diagnosis Probability Dataset):

X = {*x1, x2, x3,…, xn*}

*x1*ϵ(0; 1) *– HighBP*

*x2*ϵ(0; 1) *– HighChol*

*x3*ϵ(0; 1) *– CholCheck*

*x4*ϵ(12; 98) *– BMI*

*x5*ϵ(0; 1) *– Smoker*

*x6*ϵ(0; 1) *– Stroke*

*x6*ϵ(0; 1) *– HeartDiseaseorAttack*

*x7*ϵ(0; 1) *– PhysActivity*

*x8*ϵ(0; 1) *– Fruits*

*x9*ϵ(0; 1) *– Veggies*

*x10*ϵ(0; 1) *– HvyAlcoholConsump*

*x11*ϵ(0; 1) *– AnyHealthcare*

*x12*ϵ(0; 1) *– NoDocbcCost*

*x13*ϵ(0; 5) *– GenHlth*

*x14*ϵ(0; 30) *– MenHlth*

*x15*ϵ(0; 30) *– PhysHlth*

*x16*ϵ(0; 1) *– DiffWalk*

*x17*ϵ(0; 1) *– Sex*

*x18*ϵ(0; 13) *– Age*

*x19*ϵ(0; 6) *– Education*

*x20*ϵ(0; 8) *– Income*

n = 20

* Функции:
  + HighBP (высокое давление)
  + HighChol (высокий уровень холестерина)
  + CholCheck (проверка уровня холестерина)
  + BMI (индекс массы тела)
  + Smoker (курение)
  + Stroke (инсульт)
  + HeartDiseaseorAttack (сердечный приступ)
  + и другие признаки, которые представляют собой факторы здоровья и образа жизни.
* Целевой признак: Diabetes\_012 (класс 0 - нет диабета, класс 1 - преддиабет, класс 2 - диабет).

b. Физическая производительность (Physical Performance Dataset):

X = {*x1, x2, x3,…, xn*}

*x1*ϵ(21; 64) *– age*

*x2*ϵ(M=63%; F=37%) *– gender*

*x3*ϵ(125; 194) *– height\_cm*

*x4*ϵ(26.3; 138) *– weight\_kg*

*x5*ϵ(3; 78.4) *– body\_fat\_%*

*x6*ϵ(0; 156) *– diastolic*

*x7*ϵ(0; 201) *– systolic*

*x8*ϵ(0; 70.5) *– gripForce*

*x9*ϵ(-25; 213) *– sit and bend forward\_cm*

*x10*ϵ(0; 80) *– sit-ups counts*

*x11*ϵ(0; 303) *– broad jump\_cm*

n = 12

* Функции:
  + Age (возраст)
  + Height (рост)
  + Weight (вес)
  + BodyFat (процент жира в теле)
  + GripStrength (сила хвата)
  + и другие физические характеристики.
* Целевой признак: Class (класс физической производительности).

c. Качество вина (Wine Quality Dataset):

X = {*x1, x2, x3,…, xn*}

*x1*ϵ(0.08; 1.1) *– volatile acidity*

*x2*ϵ(0.0; 1.66) *– critic acid*

*x3*ϵ(0.6; 65.8) *– residual sugar*

*x4*ϵ(0.009; 0.346) *– chlorids*

*x5*ϵ(2; 289) *– free sulfur dioxide*

*x6*ϵ(9; 440) *–total sulfur dioxide*

*x7*ϵ(0.98711; 1.03898) *– density*

*x8*ϵ(2.72; 3.82) *– pH*

*x9*ϵ(0.22; 1.08) *– sulphates*

*x10*ϵ(8.0; 14.2) *–alcohol*

*x11*ϵ(3.8; 14.2) *– fixed acidity*

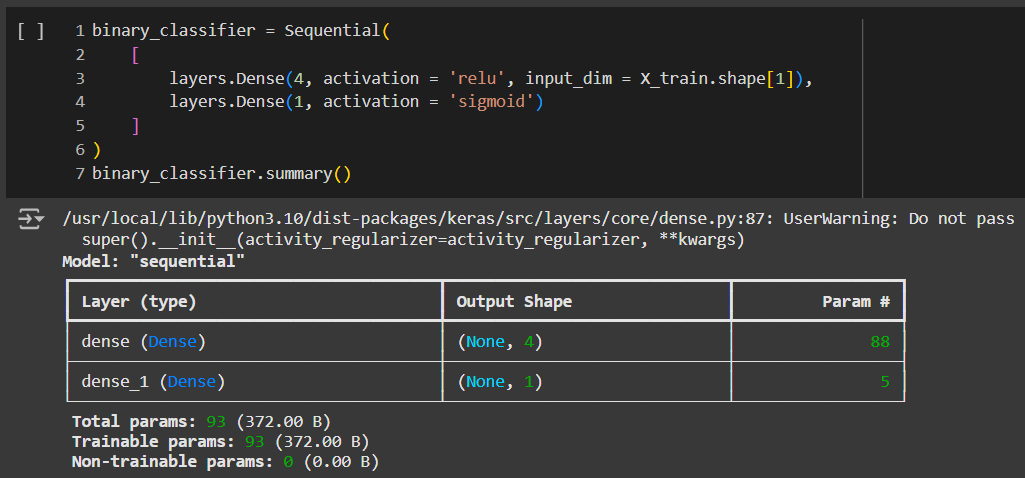
n = 11

* Функции:
  + Fixed Acidity (фиксированная кислотность)
  + Volatile Acidity (летучая кислотность)
  + Citric Acid (лимонная кислота)
  + Residual Sugar (остаточный сахар)
  + Chlorides (хлориды)
  + и другие химические свойства вина.
* Целевой признак: Quality (оценка качества вина).

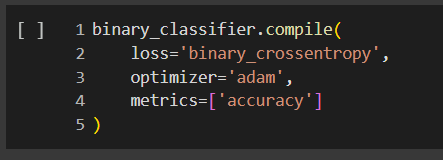
# **3. Параметры архитектур и обучения нейронных сетей, использованные для обучения.**

***a. Бинарный классификатор:***

***Первая модель***

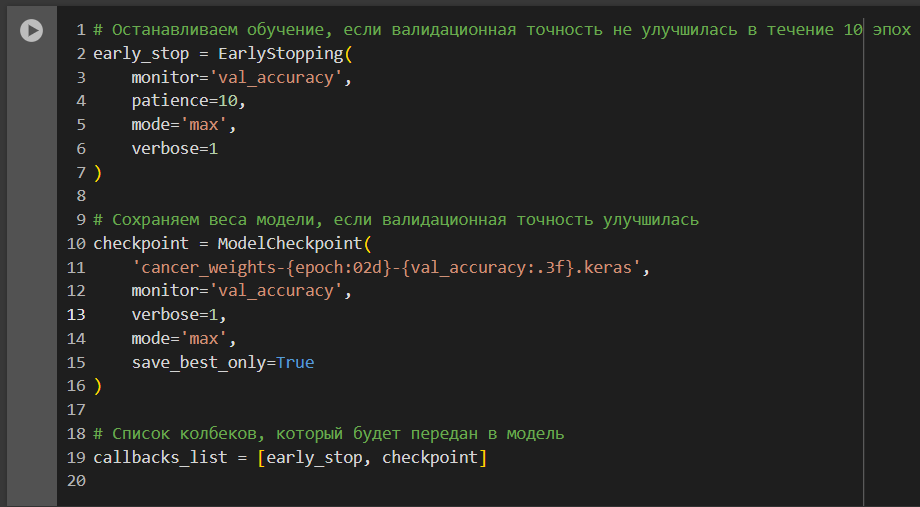
****

1. binary\_classifier = Sequential(...): Эта строка создает модель типа Sequential. Модель Sequential - это линейная стопка слоев в Keras, то есть слои добавляются друг за другом в последовательности. Представьте это как строительные блоки, где каждый блок является слоем нейронной сети. Мы присваиваем эту модель переменной binary\_classifier.
2. [layers.Dense(...), layers.Dense(...)]: Внутри модели Sequential мы определяем слои нашей нейронной сети. Эта модель имеет два слоя:
   * layers.Dense(4, activation='relu', input\_dim=X\_train.shape[1]): Это первый слой, плотно связанный слой, также называемый полносвязным слоем.
     + 4: Указывает, что слой имеет 4 нейрона (или единицы).
     + activation='relu': Устанавливает функцию активации на ReLU (Rectified Linear Unit). Функции активации вводят нелинейность в модель, позволяя ей изучать сложные паттерны. ReLU - это распространенная функция активации, которая выводит вход напрямую, если он положительный, в противном случае выводит 0.
     + input\_dim=X\_train.shape[1]: Указывает форму входных данных слоя. Он установлен на количество признаков (столбцов) в обучающих данных (X\_train). Это важно для первого слоя, чтобы знать, сколько входов ожидать.
   * layers.Dense(1, activation='sigmoid'): Это второй и выходной слой, также плотно связанный слой.
     + 1: Указывает, что слой имеет 1 нейрон, что типично для бинарной классификации.
     + activation='sigmoid': Устанавливает функцию активации на сигмоиду. Сигмоидная функция выдает значение между 0 и 1, которое можно интерпретировать как вероятность. Это подходит для бинарной классификации, где мы хотим предсказать вероятность принадлежности экземпляра к одному из двух классов.
3. binary\_classifier.summary(): Эта строка печатает сводку архитектуры модели, включая количество параметров в каждом слое и общее количество параметров в модели. Это помогает понять структуру и сложность созданной нейронной сети.

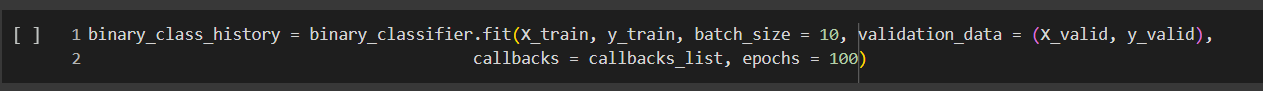


Этот код настраивает модель binary\_classifier для обучения задаче бинарной классификации.

* loss='binary\_crossentropy': Указывает функцию для измерения ошибки прогнозирования, подходящую для бинарной классификации (например, предсказания диабета).
* optimizer='adam': Выбирает алгоритм adam для эффективного обновления весов модели с целью минимизации ошибки.
* metrics=['accuracy']: Использует метрику accuracy для оценки производительности модели путем измерения процента правильных предсказаний.

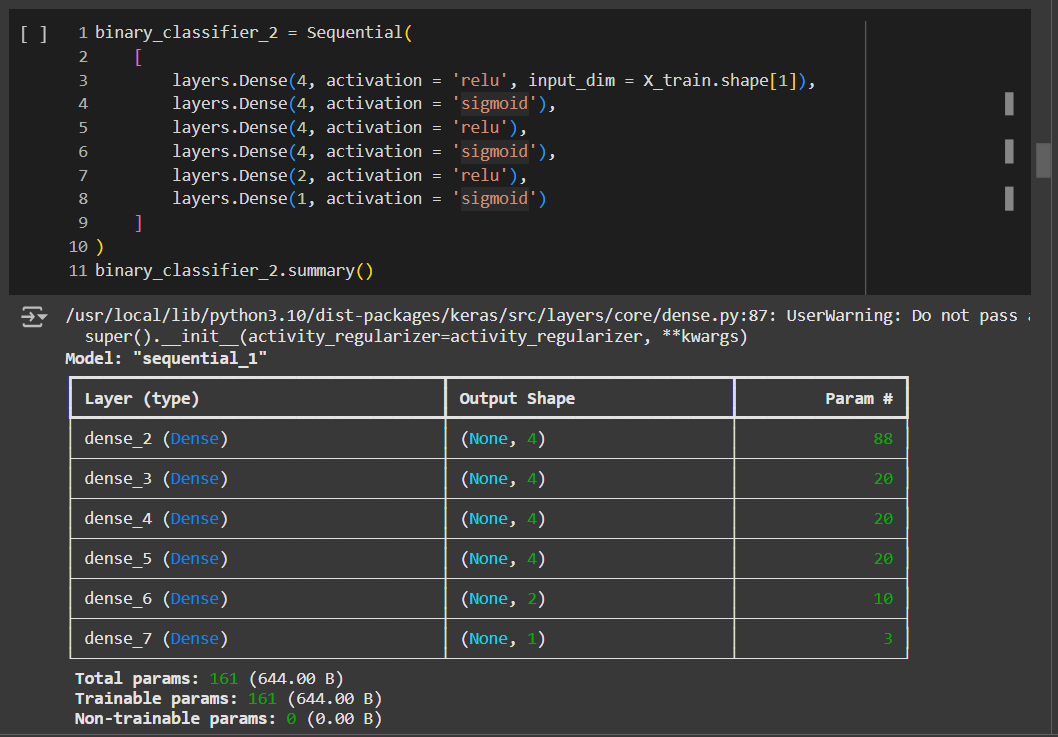


* monitor='val\_accuracy': Эта строка указывает EarlyStopping следить за точностью модели на валидационных данных (val\_accuracy). Валидационные данные - это отдельная часть набора данных, которая не используется для обучения, а скорее для оценки производительности модели во время обучения.
* patience=10: Это устанавливает уровень "терпения" на 10. Это означает, что если val\_accuracy не улучшится в течение 10 последовательных эпох (циклов обучения), процесс обучения будет остановлен.
* mode='max': Поскольку мы хотим максимизировать точность валидации, мы устанавливаем mode на 'max'.
* verbose=1: Этот параметр гарантирует, что сообщения об ранней остановке будут выводиться в консоль.
* 'cancer\_weights-{epoch:02d}-{val\_accuracy:.3f}.keras': Это определяет формат имени файла для сохранения весов модели. Он будет включать номер эпохи (epoch:02d) и точность валидации (val\_accuracy:.3f) в имя файла.
* monitor='val\_accuracy': Аналогично EarlyStopping, это указывает, что модель должна быть сохранена на основе ее производительности на валидационных данных (val\_accuracy).
* verbose=1: Этот параметр включает вывод сообщений в консоль при сохранении модели.
* mode='max': Мы хотим сохранить модель, когда точность валидации максимальна, поэтому режим 'max'.
* save\_best\_only=True: Эта важная опция гарантирует, что сохраняется только модель с лучшей точностью валидации на данный момент, предотвращая ненужное хранение потенциально худших моделей.

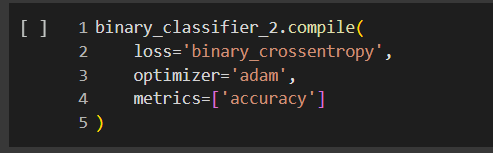


Эта строка кода обучает нейронную сеть (binary\_classifier) на обучающих данных (X\_train, y\_train). Она обрабатывает данные пакетами по 10 образцов, оценивает производительность на валидационном наборе (X\_valid, y\_valid) и использует обратные вызовы (callbacks\_list) для управления обучением (например, раннюю остановку, сохранение лучших весов). Процесс обучения выполняется максимум 100 эпох (итераций по всему набору данных). История обучения сохраняется в binary\_class\_history.

***Вторая модель***

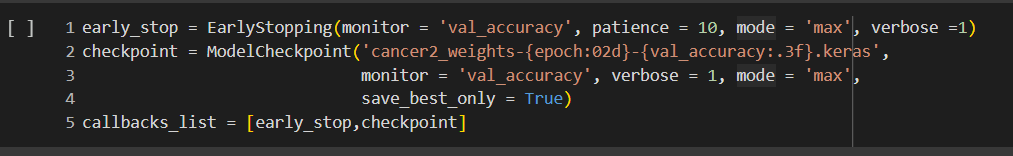


1. binary\_classifier\_2 = Sequential(...): Эта строка инициализирует модель Sequential и присваивает ее переменной binary\_classifier\_2. Это означает, что модель будет обрабатывать информацию поэтапно через каждый слой, определенный в скобках.
2. layers.Dense(...): Это основной тип слоя в нейронных сетях. Каждая строка layers.Dense создает полностью связанный слой, где каждый нейрон в слое связан с каждым нейроном в предыдущем слое.
3. Разбивка слоев: Давайте посмотрим на конкретные слои, созданные внутри модели Sequential:
   * layers.Dense(4, activation='relu', input\_dim=X\_train.shape[1]): Это первый скрытый слой.
     + 4: Он имеет 4 нейрона.
     + activation='relu': Он использует функцию активации ReLU (Rectified Linear Unit), которая вводит нелинейность в модель.
     + input\_dim=X\_train.shape[1]: Это определяет форму входных данных слоя, соответствующую количеству признаков (столбцов) в обучающих данных (X\_train).
   * layers.Dense(4, activation='sigmoid'): Это второй скрытый слой.
     + 4: Он имеет 4 нейрона.
     + activation='sigmoid': Он использует сигмоидную функцию активации, которая выдает значения от 0 до 1, часто используемую для бинарной классификации.
   * layers.Dense(4, activation='relu'): Это третий скрытый слой.
     + 4: Он имеет 4 нейрона.
     + activation='relu': Он использует функцию активации ReLU.
   * layers.Dense(4, activation='sigmoid'): Это четвертый скрытый слой.
     + 4: Он имеет 4 нейрона.
     + activation='sigmoid': Он использует сигмоидную функцию активации.
   * layers.Dense(2, activation='relu'): Это пятый скрытый слой.
     + 2: Он имеет 2 нейрона.
     + activation='relu': Он использует функцию активации ReLU.
   * layers.Dense(1, activation='sigmoid'): Это выходной слой.
     + 1: Он имеет 1 нейрон, потому что это задача бинарной классификации (выдает вероятность от 0 до 1).
     + activation='sigmoid': Он использует сигмоидную функцию активации для вычисления вероятности для бинарной классификации.
4. binary\_classifier\_2.summary(): Эта строка печатает сводку архитектуры модели, показывая слои, их формы и количество параметров в каждом слое. Это помогает понять структуру нейронной сети.



Эта строка кода является частью метода compile для нейронной сети Keras. Метод compile настраивает процесс обучения модели.

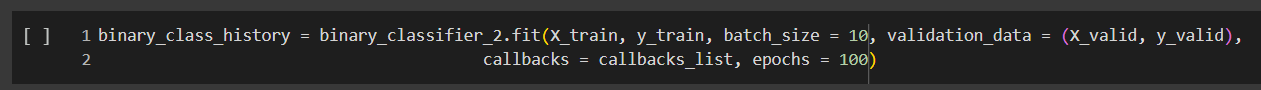
* optimizer: Этот аргумент определяет алгоритм оптимизации, используемый для обновления весов нейронной сети во время обучения. Алгоритмы оптимизации имеют решающее значение для поиска наилучшего набора весов для минимизации ошибок модели.
* 'adam': Это конкретный выбранный алгоритм оптимизации - оптимизатор Adam.



* EarlyStopping: Это обратный вызов Keras, который помогает предотвратить переобучение, явление, когда модель хорошо работает на обучающих данных, но плохо на невидимых данных.
* monitor = 'val\_accuracy': Это указывает, что обратный вызов будет отслеживать точность модели на валидационных данных (val\_accuracy). Валидационные данные - это отдельная часть набора данных, которая не используется для обучения, а скорее для оценки производительности модели во время обучения.
* patience = 10: Это означает, что если отслеживаемая величина (val\_accuracy в этом случае) не улучшается в течение 10 последовательных эпох (циклов обучения), обучение будет остановлено.
* mode = 'max': Это указывает, что мы хотим максимизировать отслеживаемую величину (val\_accuracy).
* verbose = 1: Это устанавливает уровень детализации. Значение 1 означает, что сообщения будут выводиться, когда обратный вызов выполняет действие (например, остановка обучения).
* ModelCheckpoint: Этот обратный вызов сохраняет веса модели в определенные моменты во время обучения.
* 'cancer2\_weights-{epoch:02d}-{val\_accuracy:.3f}.keras': Это определяет шаблон имени файла для сохранения весов модели.
  + cancer2\_weights - это базовое имя файла.
  + {epoch:02d} - это заполняемое место для номера эпохи, отформатированного с двумя цифрами.
  + {val\_accuracy:.3f} - это заполняемое место для точности валидации, отформатированной с тремя десятичными знаками.
  + .keras - это расширение файла.
* monitor = 'val\_accuracy': Аналогично EarlyStopping, это указывает, что обратный вызов будет отслеживать точность валидации.
* verbose = 1: Сообщения будут выводиться при сохранении модели.
* mode = 'max': Мы хотим сохранить модель, когда точность валидации максимальна.
* save\_best\_only = True: Это важный параметр. Он гарантирует, что сохраняется только лучшая модель (на основе отслеживаемой величины). Если новая эпоха не дает лучшей точности валидации, веса модели не перезаписываются.



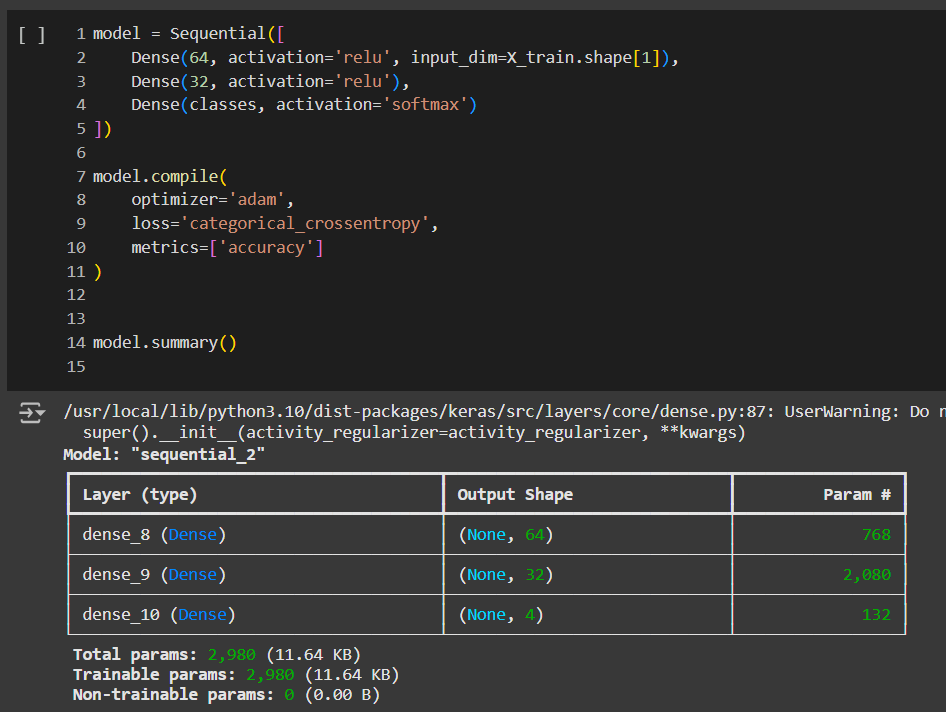
Эта строка просто создает список под названием callbacks\_list, содержащий объекты early\_stop и checkpoint. Этот список будет передан методу fit модели, позволяя использовать оба обратных вызова во время обучения.



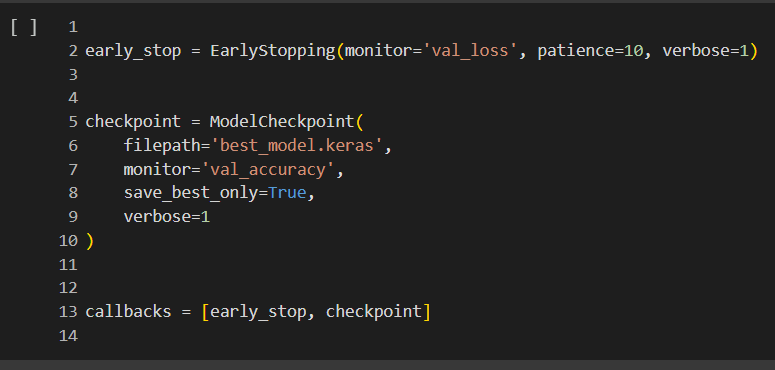
* binary\_class\_history = ...: Эта часть присваивает результаты процесса обучения переменной binary\_class\_history. Эта переменная будет содержать информацию об эффективности модели во время обучения, такую как ее точность и ошибка со временем.
* binary\_classifier\_2.fit(...): Это ядро строки. Функция fit - стандартный метод в библиотеках машинного обучения, используемый для обучения модели. Мы вызываем ее для объекта нашей модели binary\_classifier\_2.
* X\_train, y\_train: Это обучающие данные.
  + X\_train представляет входные признаки (данные, используемые для прогнозирования).
  + y\_train представляет целевую переменную (то, что мы пытаемся предсказать). Модель изучает взаимосвязь между X\_train и y\_train во время обучения.
* batch\_size = 10: Это управляет тем, сколько образцов данных обрабатывается одновременно во время обучения. Размер пакета 10 означает, что модель обновляет свои внутренние параметры после обработки 10 точек данных.
* validation\_data = (X\_valid, y\_valid): Это обеспечивает отдельный набор данных (X\_valid, y\_valid), по которому модель оценивается во время обучения. Это помогает контролировать, насколько хорошо модель обобщает на невидимые данные, и предотвращает переобучение (когда модель хорошо работает на обучающих данных, но плохо на новых данных).
* callbacks = callbacks\_list: Это указывает на список функций (определенных ранее в коде как callbacks\_list), которые выполняются во время обучения. Эти обратные вызовы могут использоваться для таких вещей, как:
  + Ранняя остановка: Остановка процесса обучения на ранней стадии, если эффективность модели на валидационной выборке перестает улучшаться.
  + Сохранение контрольных точек: Сохранение лучших весов модели во время обучения.
* epochs = 100: Это определяет количество раз, которое модель будет проходить по всему набору обучающих данных. В данном случае модель увидит обучающие данные 100 раз.

***b. Многоклассовый классификатор:***

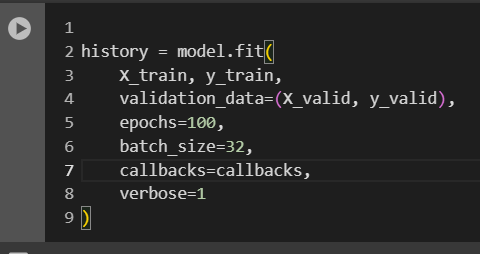
***Первая модель***

****

* model = Sequential(...): Эта строка инициализирует последовательную модель. В Keras последовательная модель представляет собой линейную стопку слоев, где вы добавляете слои один за другим.
* Dense(64, activation='relu', input\_dim=X\_train.shape[1]): Эта строка добавляет первый слой к модели.
* Dense: Это полностью связанный слой, что означает, что каждый нейрон в этом слое связан с каждым нейроном в предыдущем слое.
* 64: Это определяет количество нейронов (единиц) в этом слое.
* activation='relu': Это устанавливает функцию активации для слоя на Rectified Linear Unit (ReLU). ReLU - это распространенная функция активации, которая вводит нелинейность в модель.
* input\_dim=X\_train.shape[1]: Это определяет форму входных данных для слоя, которая представляет собой количество признаков в ваших обучающих данных (X\_train).
  + Dense(32, activation='relu'): Эта строка добавляет второй скрытый слой, аналогичный первому, но с 32 нейронами.
  + Dense(classes, activation='softmax'): Эта строка добавляет выходной слой к модели.
* classes: Это представляет количество классов в вашей классификационной задаче.
* activation='softmax': Это использует функцию активации softmax в выходном слое. Softmax преобразует необработанный выход сети в вероятности для каждого класса, позволяя модели предсказать наиболее вероятный класс для данного входа.



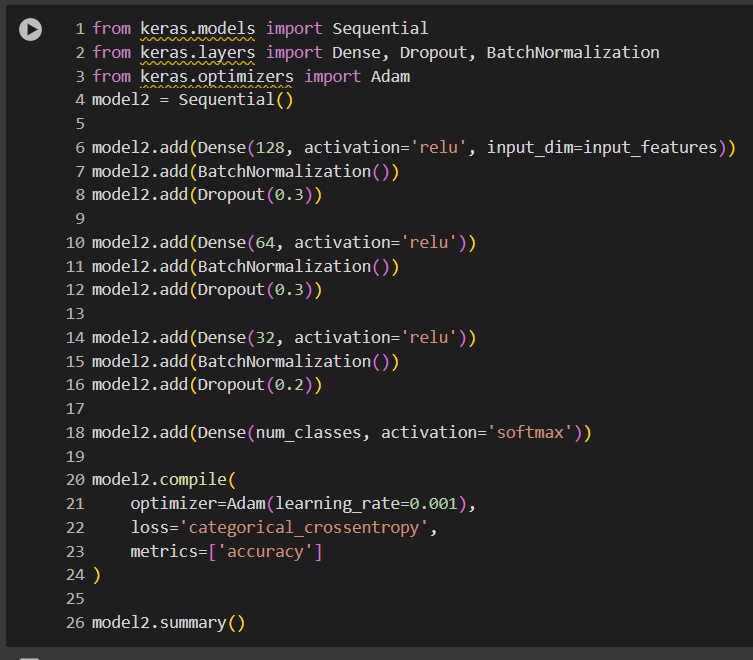
* EarlyStopping: Это функция из модуля keras.callbacks, которая помогает предотвратить переобучение. Переобучение происходит, когда модель слишком хорошо изучает обучающие данные и плохо работает на невидимых данных.
* early\_stop: Это переменная, присвоенная экземпляру EarlyStopping.
* monitor='val\_loss': Это говорит EarlyStopping отслеживать производительность модели на валидационных данных, используя валидационную потерю (val\_loss).
* patience=10': Это устанавливает уровень терпения. Если отслеживаемая метрика (val\_loss в данном случае) не улучшается в течение 10 последовательных эпох (циклов обучения), процесс обучения будет остановлен.
* verbose=1': Этот параметр управляет уровнем информации, отображаемой во время обучения. Значение 1 означает, что будут выводиться сообщения, когда срабатывает ранняя остановка.
* ModelCheckpoint: Эта функция, также из keras.callbacks, сохраняет веса модели во время обучения.
* checkpoint: Это переменная, присвоенная экземпляру ModelCheckpoint.
* filepath='best\_model.keras': Это определяет путь к файлу, где будут сохранены веса модели. В этом случае это 'best\_model.keras'.
* monitor='val\_accuracy': Это говорит ModelCheckpoint использовать точность валидации (val\_accuracy) в качестве метрики для определения, когда сохранять модель.
* save\_best\_only=True': С этим параметром включенным, веса модели сохраняются только тогда, когда отслеживаемая метрика (val\_accuracy) достигает своего лучшего значения на данный момент.
* verbose=1': Аналогично Early Stopping, verbose=1 выводит сообщения, когда сохраняется новая лучшая модель.
* callbacks: Этот список объединяет объекты early\_stop и checkpoint, созданные ранее. Этот список будет передан методу fit модели, чтобы использовать как раннюю остановку, так и контрольные точки модели во время обучения.

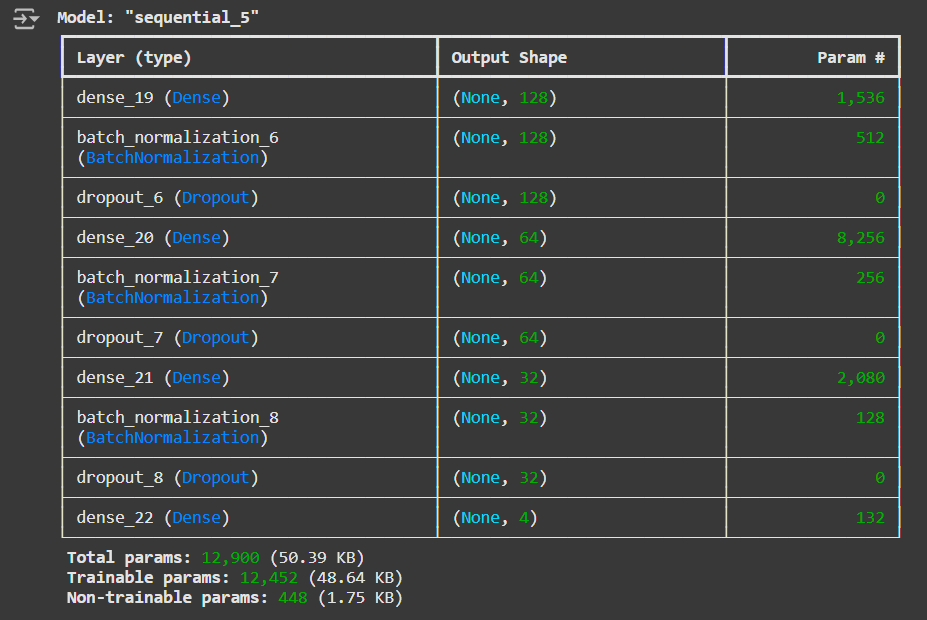


Этот код обучает нейронную сеть (model) на обучающих данных (X\_train, y\_train) и оценивает ее на валидационном наборе (X\_valid, y\_valid).

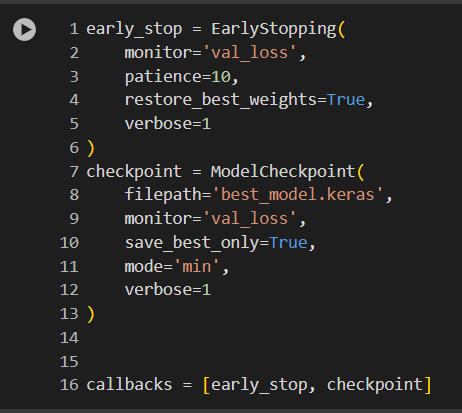
* epochs=100: Модель увидит весь набор обучающих данных 100 раз.
* batch\_size=32: Модель обновляет свои веса после просмотра 32 образцов.
* callbacks=callbacks: Функции для управления обучением, такие как ранняя остановка и сохранение лучшей модели.
* verbose=1: Отображает индикаторы выполнения и метрики обучения.

***Вторая модель***

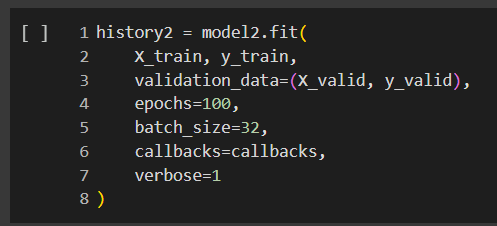
****



* model2.add(Dense(128, activation='relu', input\_dim=input\_features)): Добавляет первый скрытый слой с 128 нейронами, используя функцию активации ReLU. input\_dim определяет количество входных признаков.
* model2.add(BatchNormalization()): Добавляет слой Batch Normalization для нормализации активаций предыдущего слоя.
* model2.add(Dropout(0.3)): Добавляет слой Dropout с коэффициентом отсева 30%, что означает, что 30% нейронов будут случайным образом отброшены во время обучения. Шаблон Dense, BatchNormalization и Dropout повторяется еще для двух скрытых слоев с 64 и 32 нейронами соответственно.
* model2.add(Dense(num\_classes, activation='softmax')): Добавляет выходной слой с num\_classes нейронами (представляющими количество классов в классификационной задаче). Функция активации softmax используется для получения распределения вероятностей по классам.
* optimizer=Adam(learning\_rate=0.001): Указывает оптимизатор Adam с коэффициентом обучения 0,001.
* loss='categorical\_crossentropy': Указывает функцию потерь, используемую во время обучения, которая подходит для многоклассовой классификации.
* metrics=['accuracy']: Указывает метрики, которые будут оцениваться во время обучения и тестирования, здесь это точность.



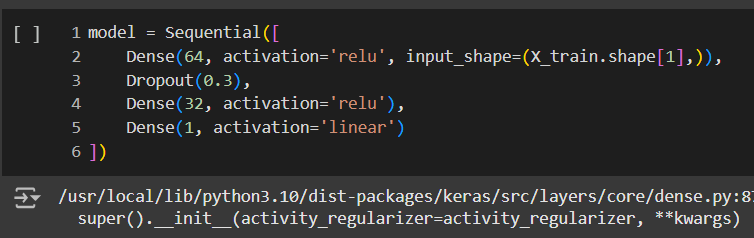
* Ранняя остановка и сохранение контрольных точек используются для улучшения обучения модели машинного обучения и предотвращения переобучения.
* Ранняя остановка (early\_stop) отслеживает ошибку на валидационной выборке (val\_loss) и останавливает обучение, если она не улучшается в течение 10 эпох (patience=10), при этом восстанавливая лучшие найденные веса.
* Сохранение контрольных точек (checkpoint) сохраняет веса модели в файл best\_model.keras, но только тогда, когда ошибка на валидационной выборке улучшается (save\_best\_only=True).
* Обе техники объединяются в список (callbacks) и используются во время обучения (model.fit) для обеспечения того, чтобы модель была хорошо обучена и избегала переобучения.



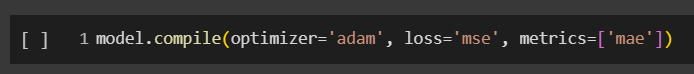
* history2 = model2.fit(...):
  + model2.fit - это функция, которая обучаеет модель с использованием предоставленных данных и параметров.
  + history2 - это переменная, которая хранит историю обучения (например, потерю и точность по эпохам), возвращаемую функцией fit. Эта информация полезна для последующего анализа процесса обучения.
* X\_train, y\_train:
  + X\_train: Эта переменная содержит признаки обучающих данных. Она содержит входные значения, используемые для обучения модели.
  + y\_train: Эта переменная содержит метки или целевые значения обучающих данных. Она содержит ожидаемый выход для соответствующих входных признаков.
* validation\_data=(X\_valid, y\_valid):
  + Этот аргумент предоставляет отдельный набор данных, X\_valid (признаки) и y\_valid (метки), для оценки производительности модели во время обучения.
  + Он помогает контролировать переобучение (когда модель хорошо работает на обучающих данных, но плохо на невидимых данных).
* epochs=100:
  + Это определяет количество раз, которое процесс обучения будет проходить по всему набору обучающих данных.
  + Одна эпоха означает, что модель видела все обучающие данные один раз.
* batch\_size=32:
  + Это определяет количество обучающих образцов, используемых в одной итерации (или шаге) процесса обучения.
  + Модель обновляет свои внутренние параметры после обработки каждого пакета данных.
* callbacks=callbacks:
  + callbacks - это список, содержащий объекты, которые определяют действия, выполняемые во время обучения, такие как:
    - early\_stop: Останавливает обучение на ранней стадии, если эффективность модели на валидационной выборке не улучшается.
    - checkpoint: Сохраняет лучшую модель во время обучения в файл.
* verbose=1:
  + Это контролирует объем информации, отображаемой во время процесса обучения. verbose=1 покажет индикатор выполнения и соответствующие метрики.

***c. Регрессор:***

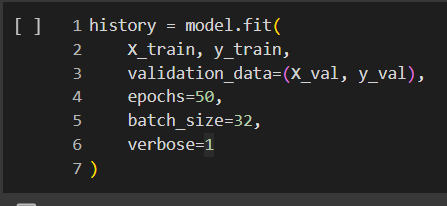
***Первая модель***



1. model = Sequential([ ... ]): Эта строка создает объект последовательной модели и присваивает его переменной model. Этот объект будет содержать все слои нашей нейронной сети.
2. Dense(64, activation='relu', input\_shape=(X\_train.shape[1],)): Эта строка добавляет к модели слой Dense.
   * Dense - это полностью связанные слои, что означает, что каждый нейрон в этом слое связан с каждым нейроном в предыдущем слое.
   * 64 определяет количество нейронов (единиц) в этом слое.
   * activation='relu' устанавливает функцию активации для этого слоя на Rectified Linear Unit (ReLU). Функции активации вводят нелинейность в модель, позволяя ей изучать сложные паттерны.
   * input\_shape=(X\_train.shape[1],) определяет форму входных данных, ожидаемых этим слоем. X\_train.shape[1] представляет количество признаков в ваших обучающих данных. Этот аргумент необходим только для первого слоя в Sequential модели.
3. Dropout(0.3): Эта строка добавляет слой Dropout.
   * Dropout - это техника регуляризации, которая помогает предотвратить переобучение (когда модель хорошо работает на обучающих данных, но плохо на невидимых данных).
   * 0.3 означает, что 30% нейронов в предыдущем слое будут случайным образом "отброшены" (проигнорированы) во время каждой эпохи обучения. Это заставляет сеть изучать более надежные признаки.
4. Dense(32, activation='relu'): Это добавляет еще один слой Dense с 32 нейронами и активацией ReLU. Это часто называют "скрытым слоем", потому что он находится между входным и выходным слоями.
5. Dense(1, activation='linear'): Это выходной слой модели.
   * Он имеет один нейрон (1), потому что эта модель, вероятно, предназначена для задачи регрессии, где цель состоит в том, чтобы предсказать одно непрерывное значение.
   * activation='linear' означает, что выход этого слоя является линейным преобразованием выхода предыдущего слоя. Это характерно для задач регрессии.

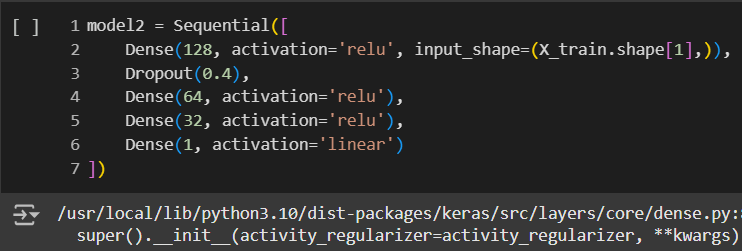


1. model.compile(): Это функция в Keras, которая настраивает модель для обучения. По сути, здесь вы указываете настройки, которые определяют, как модель должна учиться.
2. optimizer='adam':
   * optimizer - это аргумент, который определяет алгоритм оптимизации, используемый во время обучения.
   * 'adam' относится к оптимизатору Adam, популярному алгоритму, известному своей эффективностью и результативностью в различных задачах машинного обучения. Роль оптимизатора заключается в том, чтобы итеративно корректировать внутренние параметры модели (веса и смещения) для минимизации функции потерь.
3. loss='mse':
   * loss - это аргумент, который определяет функцию потерь, которая количественно оценивает разницу между прогнозами модели и фактическими целевыми значениями.
   * 'mse' означает среднеквадратичную ошибку (Mean Squared Error), распространенную функцию потерь для задач регрессии. Она вычисляет среднеквадратичную разницу между прогнозами и истинными значениями. Цель обучения - минимизировать эту ошибку.
4. metrics=['mae']:
   * metrics - это аргумент, который определяет метрики, используемые для оценки производительности модели во время обучения и тестирования.
   * ['mae'] указывает, что средняя абсолютная ошибка (Mean Absolute Error, MAE) будет использоваться в качестве метрики. MAE вычисляет среднюю абсолютную разницу между прогнозами и истинными значениями. Это еще одна распространенная метрика оценки для задач регрессии. Хотя модель в первую очередь пытается минимизировать потерю во время обучения, метрики предоставляют дополнительную информацию о ее производительности.



1. history = model.fit(...): Эта строка запускает процесс обучения.
   * model.fit() - это функция, которая обучает модель с использованием предоставленных данных.
   * Результаты процесса обучения (такие как потеря и точность со временем) хранятся в переменной history. Это позволяет нам позже проанализировать эффективность обучения.
2. X\_train, y\_train: Это входные данные для обучения модели.
   * X\_train: Представляет признаки (независимые переменные) обучающего набора данных.
   * y\_train: Представляет целевую переменную (зависимую переменную), которую мы пытаемся предсказать для обучающего набора данных. Модель обучается взаимосвязи между признаками и целевой переменной на основе этих данных.
3. validation\_data=(X\_val, y\_val): Этот аргумент предоставляет данные для оценки эффективности модели во время обучения.
   * X\_val: Признаки валидационного набора данных.
   * y\_val: Целевая переменная для валидационного набора данных.
   * После каждой эпохи (полного прохода по обучающим данным) эффективность модели оценивается на валидационных данных. Это помогает контролировать переобучение (когда модель хорошо работает на обучающих данных, но плохо на невидимых данных).
4. epochs=50: Это определяет количество раз, которое процесс обучения будет проходить по всему набору обучающих данных.
   * Одна эпоха означает, что модель увидела и усвоила все обучающие данные один раз.
   * Большее количество эпох может привести к лучшей обучаемости, но слишком большое количество может привести к переобучению.
5. batch\_size=32: Это определяет количество образцов данных, которые обрабатываются вместе за одну итерацию во время обучения.
   * Вместо того, чтобы обновлять веса модели после просмотра каждой отдельной точки данных, она обновляет их после просмотра пакета из 32 образцов.
   * Размер пакета влияет на скорость и стабильность процесса обучения.
6. verbose=1: Это контролирует объем информации, отображаемой во время обучения.
   * verbose=1 означает, что ход обучения будет выводиться в консоль, показывая потерю и метрики после каждой эпохи.

***Вторая модель***



* model2 = Sequential([...]): Эта строка инициализирует последовательную модель. Последовательная модель представляет собой линейную стопку слоев, где данные последовательно передаются от одного слоя к другому.
* Dense(128, activation='relu', input\_shape=(X\_train.shape[1],)): Это первый слой модели, плотно связанный слой.
  + Dense: Указывает на полностью связанный слой, где каждый нейрон в этом слое связан с каждым нейроном в предыдущем слое.
  + 128: Определяет количество нейронов (единиц) в этом слое.
  + activation='relu': Устанавливает функцию активации для этого слоя на ReLU (Rectified Linear Unit). ReLU - это распространенная функция активации, которая вводит нелинейность в модель.
  + input\_shape=(X\_train.shape[1],): Определяет форму входных данных. X\_train.shape[1] представляет количество признаков в ваших обучающих данных. Это необходимо для первого слоя, чтобы знать ожидаемые входные размеры.
* Dropout(0.4): Это слой Dropout.
  + Dropout: Техника регуляризации, используемая для предотвращения переобучения (когда модель хорошо работает на обучающих данных, но плохо на невидимых данных).
  + 0.4: Это значение представляет коэффициент отсева, что означает, что 40% нейронов в предыдущем слое будут случайным образом деактивированы во время обучения. Это заставляет модель изучать более надежные признаки.
* Dense(64, activation='relu'): Еще один плотно связанный слой с 64 нейронами и активацией ReLU. Этот слой дополнительно обрабатывает информацию из предыдущего слоя.
* Dense(32, activation='relu'): Еще один плотно связанный слой с 32 нейронами и активацией ReLU.
* Dense(1, activation='linear'): Это выходной слой модели.
  + 1: Он имеет один нейрон, что подходит для задачи регрессии, предсказывающей одно непрерывное значение.
  + activation='linear': Использует линейную функцию активации, что означает, что выход прямо пропорционален входу. Это обычно используется для задач регрессии.

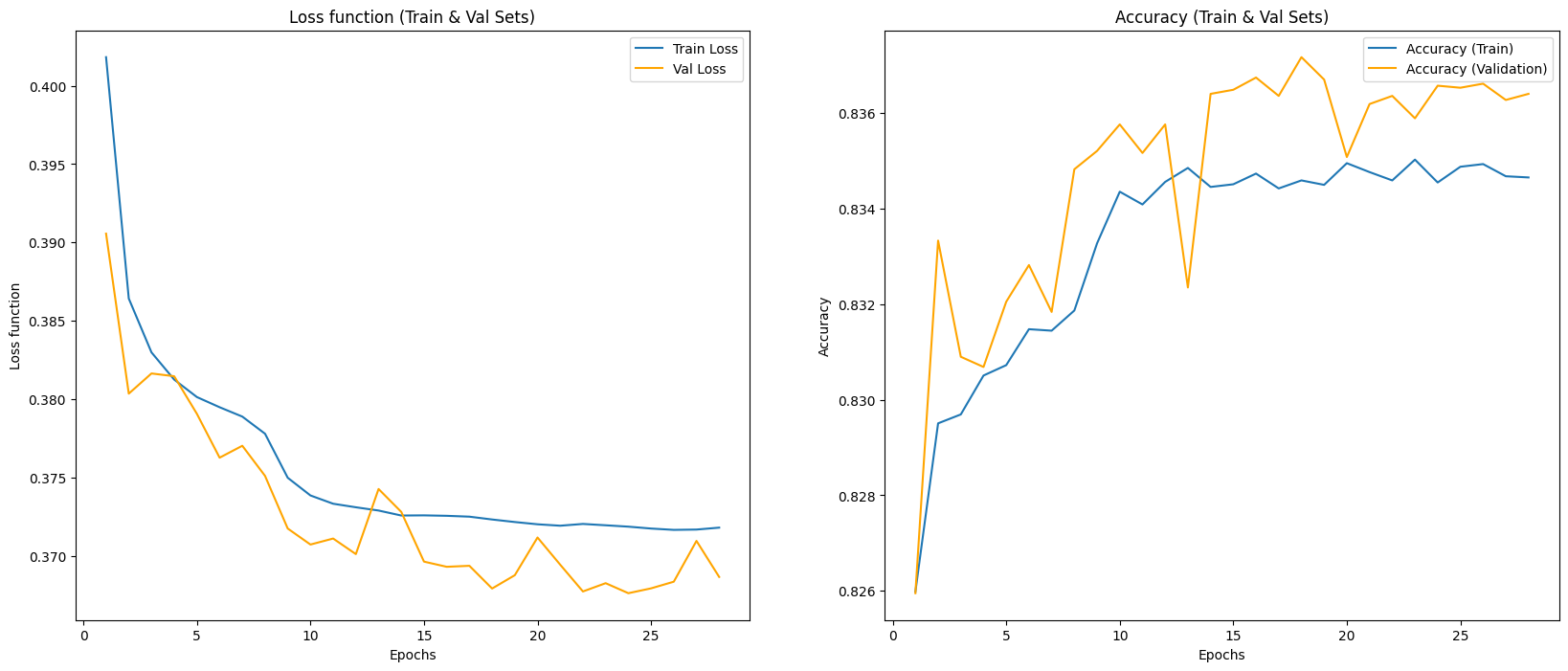


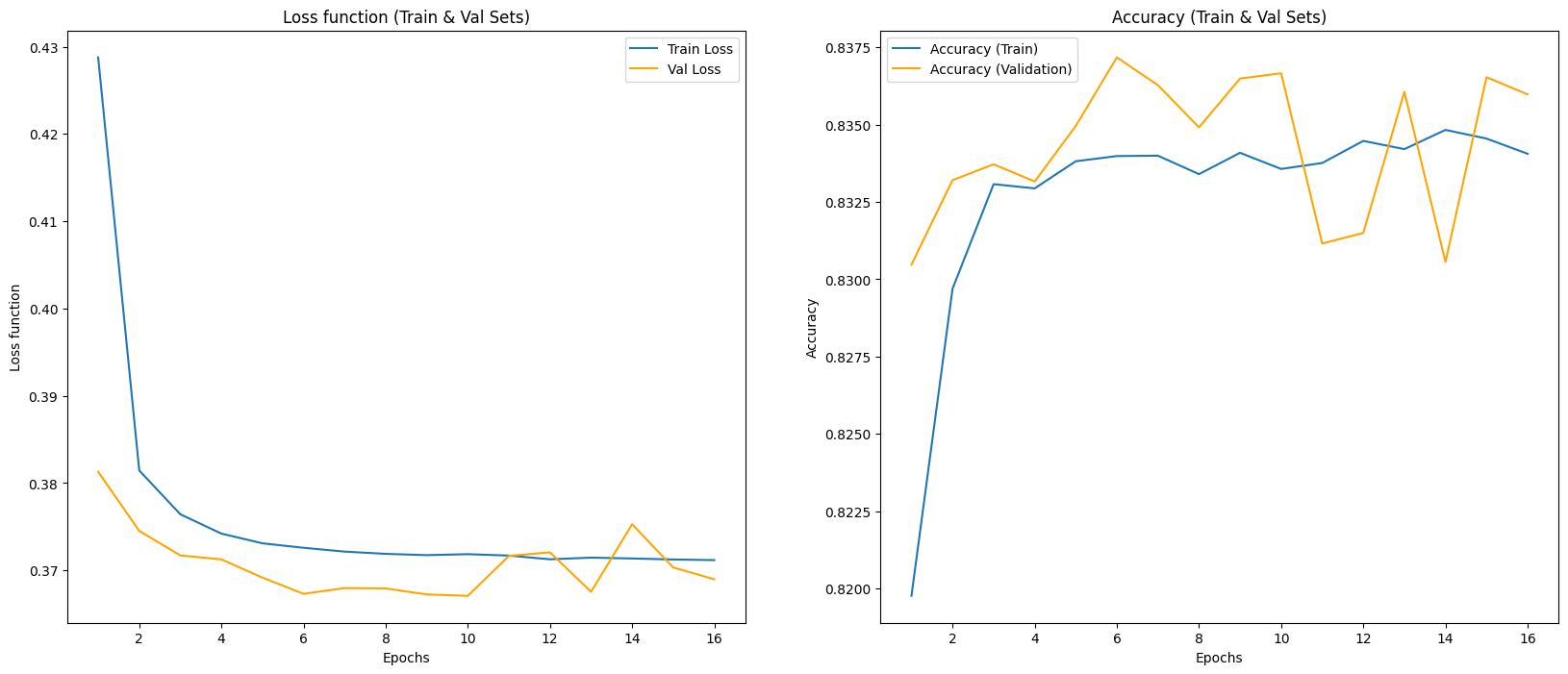
* model2.compile – Эта строка подготавливает модель к обучению.
* optimizer='adam' – Это устанавливает алгоритм оптимизации на "Adam", который является популярным методом обновления весов модели во время обучения для минимизации функции потерь.
* loss='mse' – Это определяет функцию потерь как среднеквадратичную ошибку (MSE), распространенную метрику для задач регрессии, которая измеряет среднюю квадратичную разницу между прогнозируемыми и фактическими значениями.
* metrics=['mae'] – Это указывает на то, что средняя абсолютная ошибка (MAE) должна использоваться в качестве дополнительной оценочной метрики во время обучения. MAE рассчитывает среднюю абсолютную разницу между прогнозами и фактическими значениями.
* EarlyStopping – Эта техника помогает предотвратить переобучение, которое возникает, когда модель хорошо работает на обучающих данных, но плохо на невидимых данных.
* monitor='val\_loss' – Она отслеживает ошибку на валидационной выборке (ошибку на отдельном валидационном наборе данных) во время обучения.
* patience=10 – Обучение остановится, если ошибка на валидационной выборке не улучшится в течение 10 последовательных эпох (итераций по обучающим данным).
* restore\_best\_weights=True – После остановки обучения веса модели будут сброшены до значений, которые достигли наилучшей ошибки на валидационной выборке.
* verbose=1 – Будет печатать сообщения во время обучения, включая случаи срабатывания ранней остановки.
* ModelCheckpoint – Эта функция сохраняет веса модели во время обучения.
* filepath='best\_model.keras' – Указывает путь к файлу, куда будут сохраняться веса модели.
* monitor='val\_loss' – Модель будет сохранена на основе улучшения ошибки на валидационной выборке.
* save\_best\_only=True – Будет сохранена только модель с лучшим на данный момент значением ошибки на валидационной выборке, что позволит избежать ненобходимого хранения менее оптимальных моделей.
* verbose=1 – Печатает сообщения, когда сохраняется новая лучшая модель.
* model2.fit – Запускает процесс обучения для model2.
* X\_train, y\_train – Обучающие данные и соответствующие целевые значения.
* validation\_data=(X\_val, y\_val) – Отдельный набор данных для валидации, используемый для оценки эффективности модели во время обучения.
* epochs=100 – Максимальное количество эпох обучения.
* batch\_size=32 – Количество образцов данных, обрабатываемых перед обновлением весов модели.
* callbacks=[early\_stopping, model\_checkpoint] – Применяет методы ранней остановки и сохранения модели, определенные ранее.
* verbose=1 – Печатает информацию о ходе обучения.

# **4. Графики обучения для архитектур нейронных сетей с лучшими характеристиками эффективности**

***a. Бинарный классификатор:***

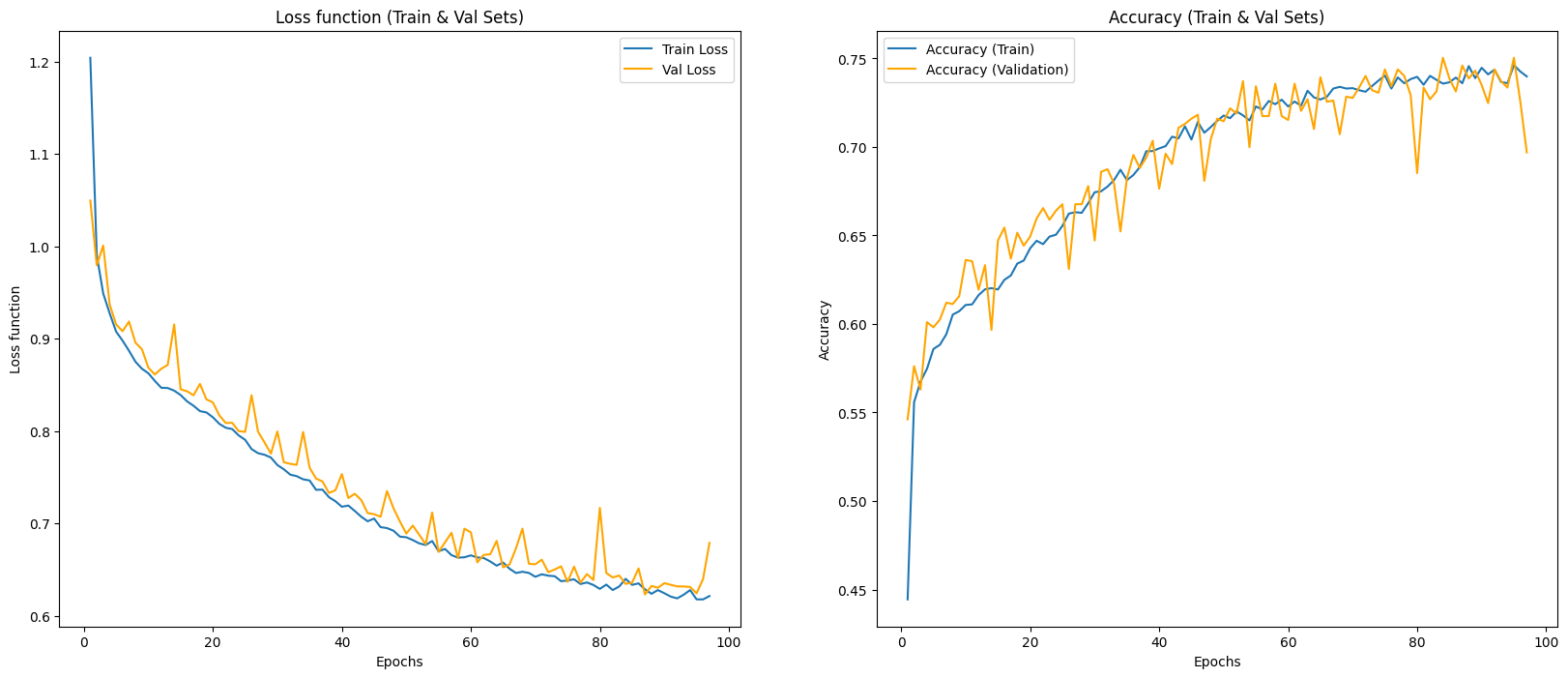
***Первая модель***

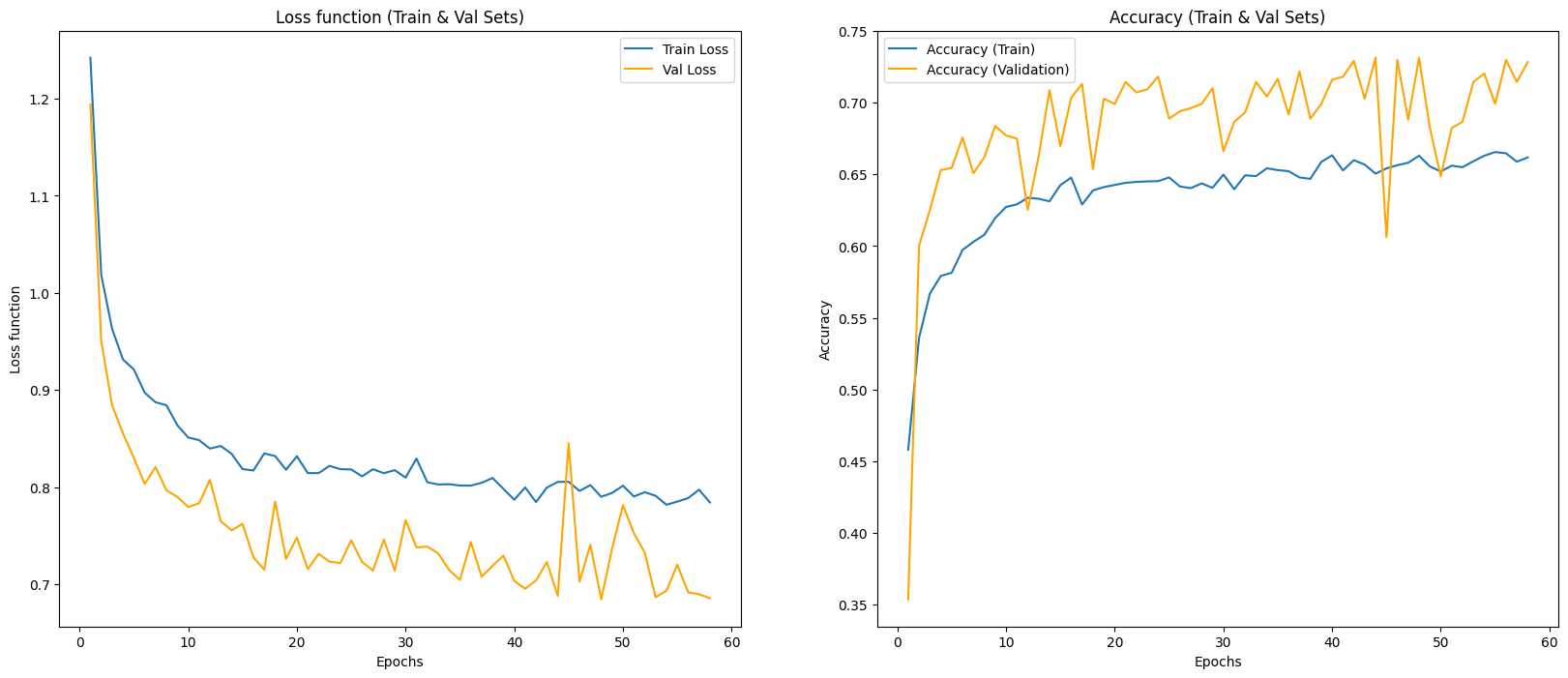


***Вторая модель***

***b. Многоклассовый классификатор:***

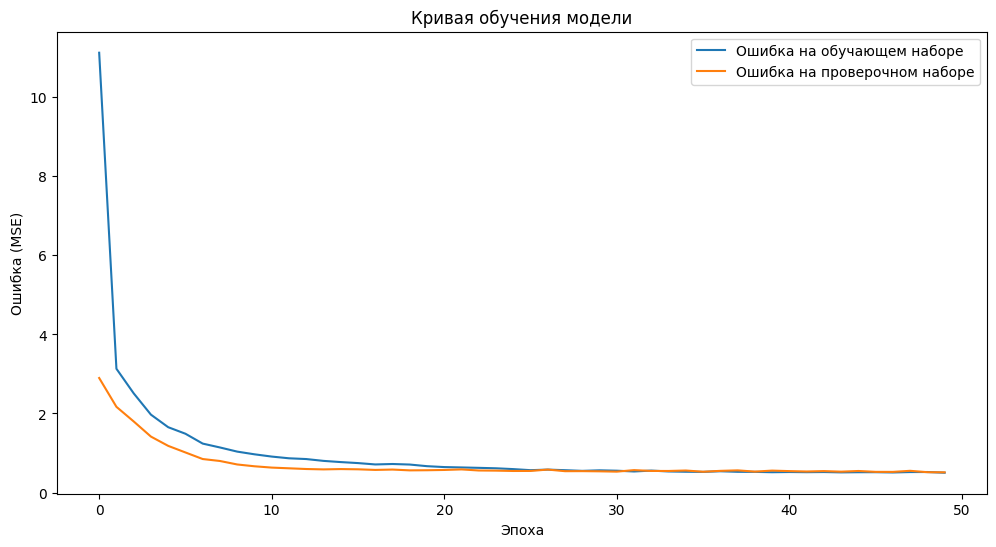
***Первая модель***

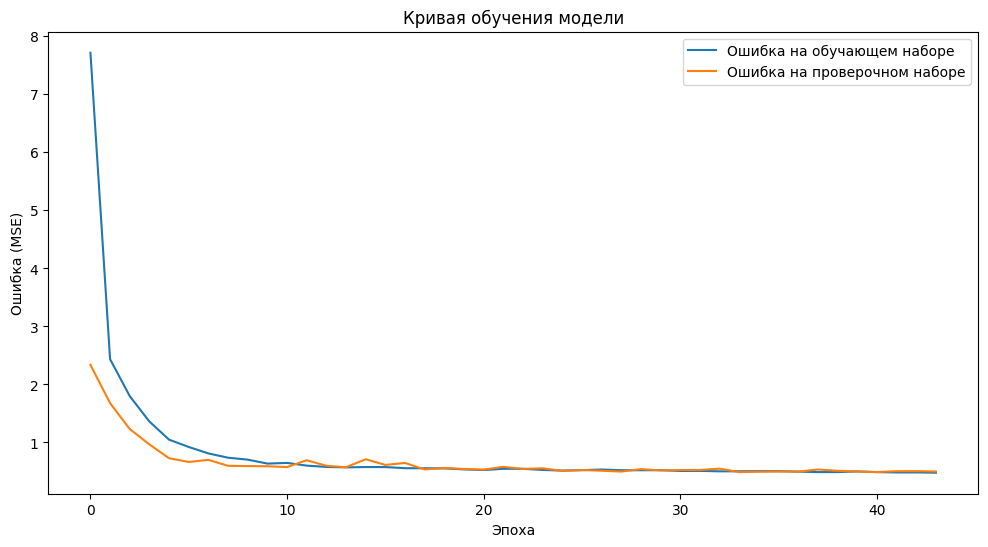


***Вторая модель***

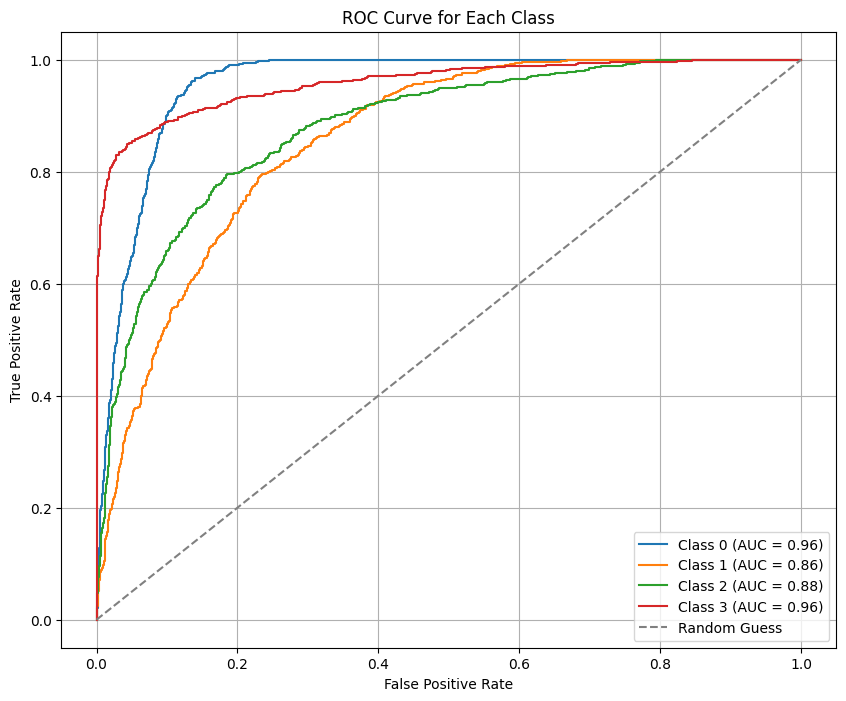
***c. Регрессор:***

***Первая модель***



***Вторая модель***

# **5. ROC-кривые классов для лучших классификаторов.**

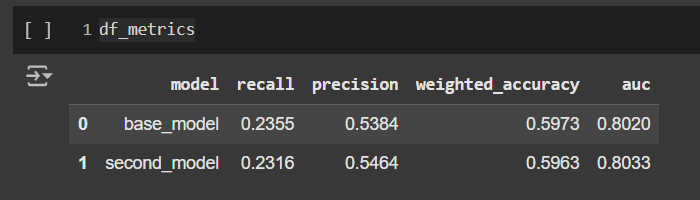


ROC (Receiver Operating Characteristic) кривые часто используются для оценки производительности классификационных моделей, особенно в задачах многоклассовой классификации. На основе предоставленного вами графика:

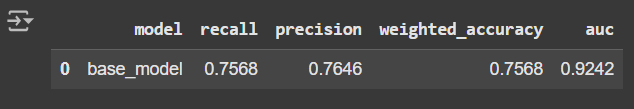
1. Оси:
   * Ось X (False Positive Rate - FPR): Представляет долю отрицательных образцов, ошибочно классифицированных как положительные.
   * Ось Y (True Positive Rate - TPR): Представляет долю положительных образцов, правильно классифицированных.
2. Кривые для каждого класса:
   * График содержит ROC-кривые для четырех классов: Класс 0, Класс 1, Класс 2 и Класс 3.
   * Каждая кривая оценивает, насколько хорошо классификатор отличает данный класс от остальных (стратегия "один против всех").
3. AUC (Площадь под кривой):
   * AUC — это сводный показатель, отражающий качество классификатора:
     + Класс 0: AUC = 0.96 (отличное качество)
     + Класс 1: AUC = 0.86 (хорошее качество)
     + Класс 2: AUC = 0.88 (хорошее качество)
     + Класс 3: AUC = 0.96 (отличное качество)
4. Базовая линия случайной классификации:
   * Пунктирная диагональная линия представляет производительность случайного классификатора (AUC = 0.5). Кривые, расположенные выше этой линии, показывают, что классификатор работает лучше случайного угадывания.
5. Интерпретация:
   * Чем выше и ближе кривая к верхнему левому углу, тем лучше классификатор минимизирует количество ложных срабатываний, одновременно максимизируя количество правильных срабатываний.
   * Классы 0 и 3 имеют наивысшие значения AUC, что указывает на высокую способность классификатора различать эти классы.
   * Классы 1 и 2 имеют немного более низкие значения AUC, но всё равно показывают хорошее качество классификации.

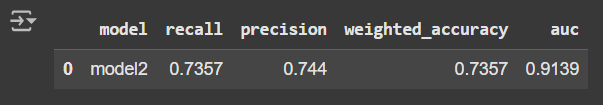
# **6. Оценки моделей на тестовых выборках в виде таблиц/ диаграмм, отображающих метрики качества.**

***Бинарный классификатор***

****

***Многоклассовый классификатор***

****

****

# **7. Программный код.**

<https://colab.research.google.com/drive/1lX7slQEuTssfMIxFyZ2ef-h89kU-FexH?usp=sharing>