**1. Постановка задачи**

Цель: построить модель рекурентной нейронной сети, классифицирующую изображения персонажей как принадлежащих к одной из двух категорий:

* Doom
* Animal Crossing

Пусть имеется обучающая выборка из изображений:

Требуется найти функцию

где θ — параметр нейросети,минимизирующие функцию потерь.

В рамках обучения использовался оптимизатор adam.

Функция потерь, бинарная кросэнтропия:

В качестве функции активации использовалась сигмойда:

**2. Тренировочные и тестовые наборы данных**

|  |  |
| --- | --- |
| **Характеристика** | **Значение** |
| Количество классов | 2 |
| Названия классов | doom, animal\_crossing |
| Общее количество примеров | 1597 изображений |
| Размер обучающих данных | 70% |
| Размер валидационных данных | 20 |
| Размер тестовых данных | 10% |
| Преобразование размеров | 64×64 |

**3. Метрика качества**

Для бинарной классификации используем **(Accuracy)**, а также дополнительно **Precision** и **Recall** для более подробного

где

* (True Positive) — число «doom»-изображений, правильно отнесённых к классу «doom».
* (True Negative) — число «animal\_crossing»-изображений, правильно отнесённых к классу «animal\_crossing».
* (False Positive)— «animal\_crossing», ошибочно помеченные как «doom».
* (False Negative) — «doom», ошибочно помеченные как «animal\_crossing».

**4. Исходный формат хранения данных**

В каталоге данные хранятся в виде:

doom-crossing/

├── animal\_crossing/

│ ├── image1.jpg

│ ├── ...

├── doom/

│ ├── image1.jpg

│ ├── ...

Формат — обычные изображения (.jpg) и (.png) в папках по классам.

**5. Формат подачи данных в сеть**

Входные изображения разбиваются на батчи, затем нормализуются в значения от 0 до 1.

Батчи делятся 70% на обучения 20% на валидацию данных(во время обучения) и 10% на тестирование получившейся модели.

**6. Разработанные программы/скрипты. Листинг кода. Описание разработанных функций.**

Загрузка данных

data = tf.keras.utils.image\_dataset\_from\_directory('/content/doom-crossing', image\_size=(IMG\_WIDTH, IMG\_HEIGHT), batch\_size=BATCH\_SIZE)

# Нормализируем данные

data = data.map(lambda x,y: (x/255, y))

Разбиваем данные на тестовую обучающую и валидационные выборки. Т.к. изображения не разбиваются на батчи идеально, формируется дополнительный неполный батч.

train\_size\_percentage = 0.7

val\_size\_percentage = 0.2

test\_size\_percentage = 0.1

spared\_batches = len(data) % 10

train\_size = int(len(data) \* train\_size\_percentage)

val\_size = int(len(data) \* val\_size\_percentage)

test\_size = int(len(data) \* test\_size\_percentage) + spared\_batches

train = data.take(train\_size)

val = data.skip(train\_size).take(val\_size)

test = data.skip(train\_size + val\_size).take(test\_size)

Задаем архитектуру сети рекурентой LSTM сети

model1 = tf.keras.Sequential([

    tf.keras.layers.Rescaling(1./255, input\_shape=IMG\_SHAPE),

    tf.keras.layers.Reshape((64, 64\*3)),

    tf.keras.layers.LSTM(128),

    tf.keras.layers.Dense(64, activation='relu'),

    tf.keras.layers.Dense(1, activation='sigmoid')

])

Функция обучения и тестирования модели. Обучает модель, а затем выводит получившиеся графики потерь и точности, вычисляет точность на тестовых данных. Строит матрицу ошибок.

def train\_and\_test\_model(model, train, val, initial\_epochs, callbacks):

    hist = model.fit(train,

                     epochs=initial\_epochs,

                     validation\_data=val,

                     callbacks=[tensorboard\_callback])

    # График функции потерь

    fig = plt.figure()

    plt.plot(hist.history['loss'], color=doom\_color, label='loss')

    plt.plot(hist.history['val\_loss'], color=ac\_color, label='val\_loss')

    fig.suptitle('Loss', fontsize=20)

    plt.legend(loc="upper left")

    plt.show()

    # График точности

    fig = plt.figure()

    plt.plot(hist.history['accuracy'], color=doom\_color, label='accuracy')

    plt.plot(hist.history['val\_accuracy'], color=ac\_color, label='val\_accuracy')

    fig.suptitle('Accuracy', fontsize=20)

    plt.legend(loc="upper left")

    plt.show()

    # Расчет точности

    precision = Precision()

    recall = Recall()

    binary\_accuracy = BinaryAccuracy()

    for batch in test.as\_numpy\_iterator():

        X, y = batch

        yhat = simple\_model.predict(X)

        # Приведение вероятностей к меткам классов

        yhat\_labels = np.argmax(yhat, axis=1)

        # Обновляем метрики

        precision.update\_state(y, yhat\_labels)

        recall.update\_state(y, yhat\_labels)

        binary\_accuracy.update\_state(y, yhat\_labels)

    print(f'Precision: {precision.result().numpy():.2f}, Recall: {recall.result().numpy():.2f}, Binary Accuracy: {binary\_accuracy.result().numpy():.2f}')

Построение двухслойной LSTM нейронной сети

model2 = tf.keras.Sequential([

    tf.keras.layers.Rescaling(1./255, input\_shape=IMG\_SHAPE),

    tf.keras.layers.Reshape((64, 64\*3)),

    tf.keras.layers.LSTM(128, return\_sequences=True),

    tf.keras.layers.LSTM(64),

    tf.keras.layers.Dense(64, activation='relu'),

    tf.keras.layers.Dense(1, activation='sigmoid')

])

Построение GRU модели.

model3 = tf.keras.Sequential([

    tf.keras.layers.Rescaling(1./255, input\_shape=IMG\_SHAPE),

    tf.keras.layers.Reshape((64, 64\*3)),

    tf.keras.layers.GRU(128),

    tf.keras.layers.Dense(64, activation='relu'),

    tf.keras.layers.Dense(1, activation='sigmoid')

])

**7.** **Тестовые конфигурации сетей.**

**LSTM сеть.**

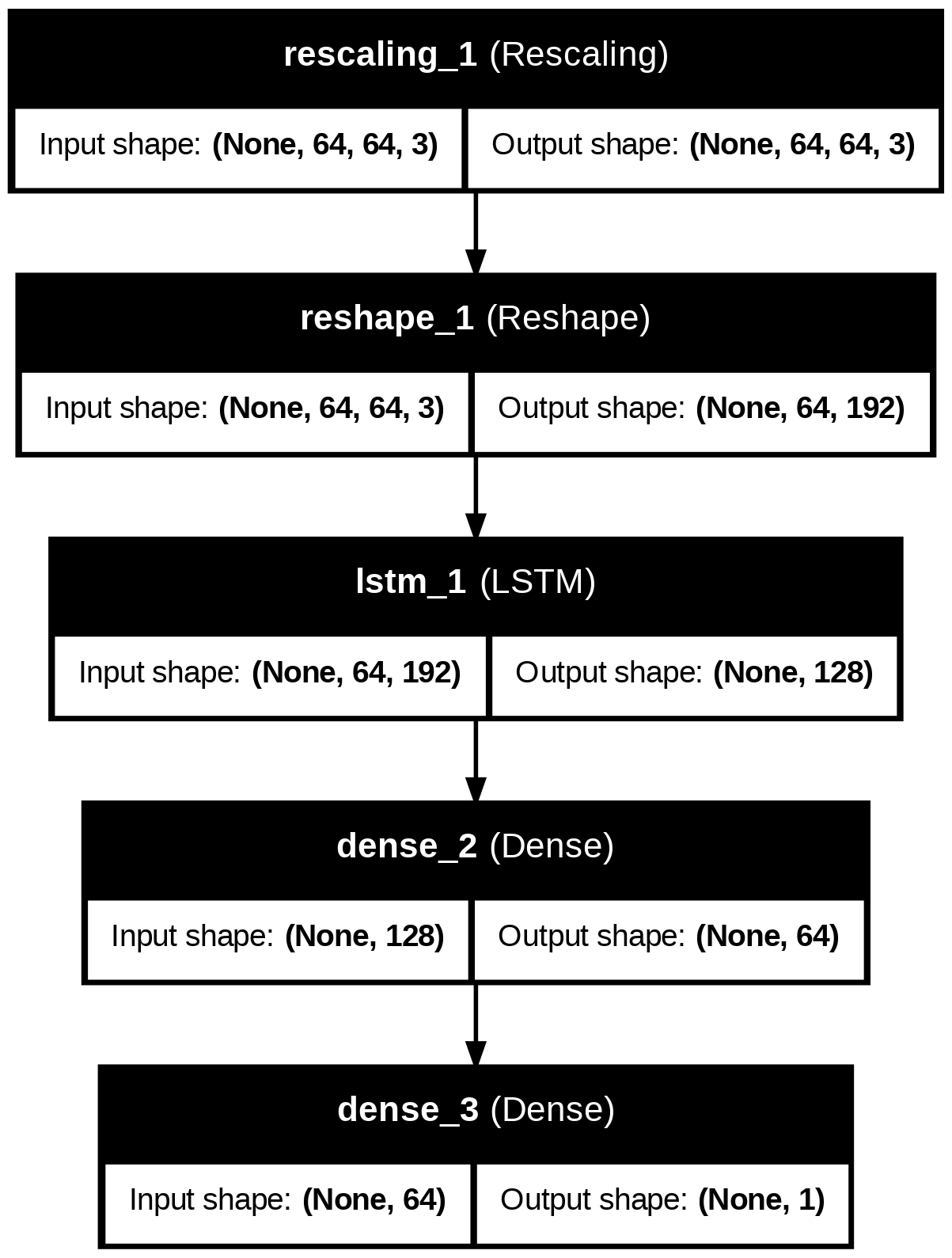


Рисунок 1 – Архитектура LSTM сети

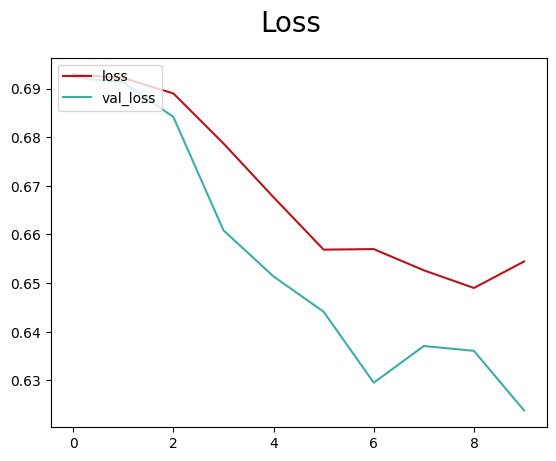


Рисунок – График потерь для простой LSTM сети

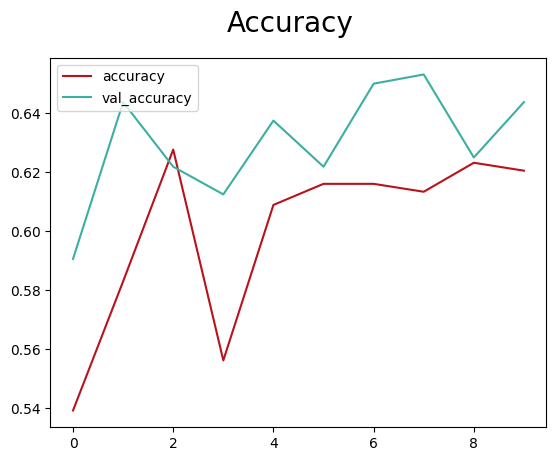


Рисунок – График точности для простой LSTM сети

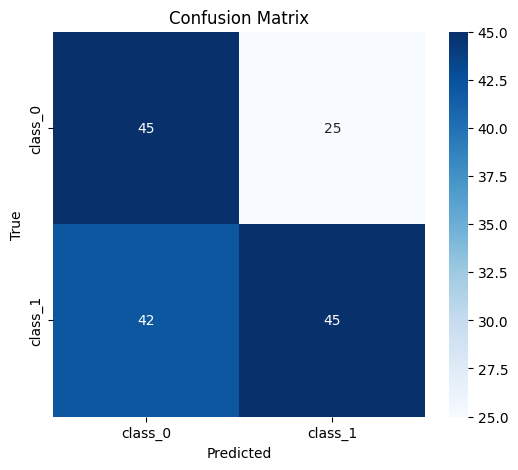


Рисунок – Матрица ошибок для простой LSTM сети

**Двухслойная LSTM сеть.**

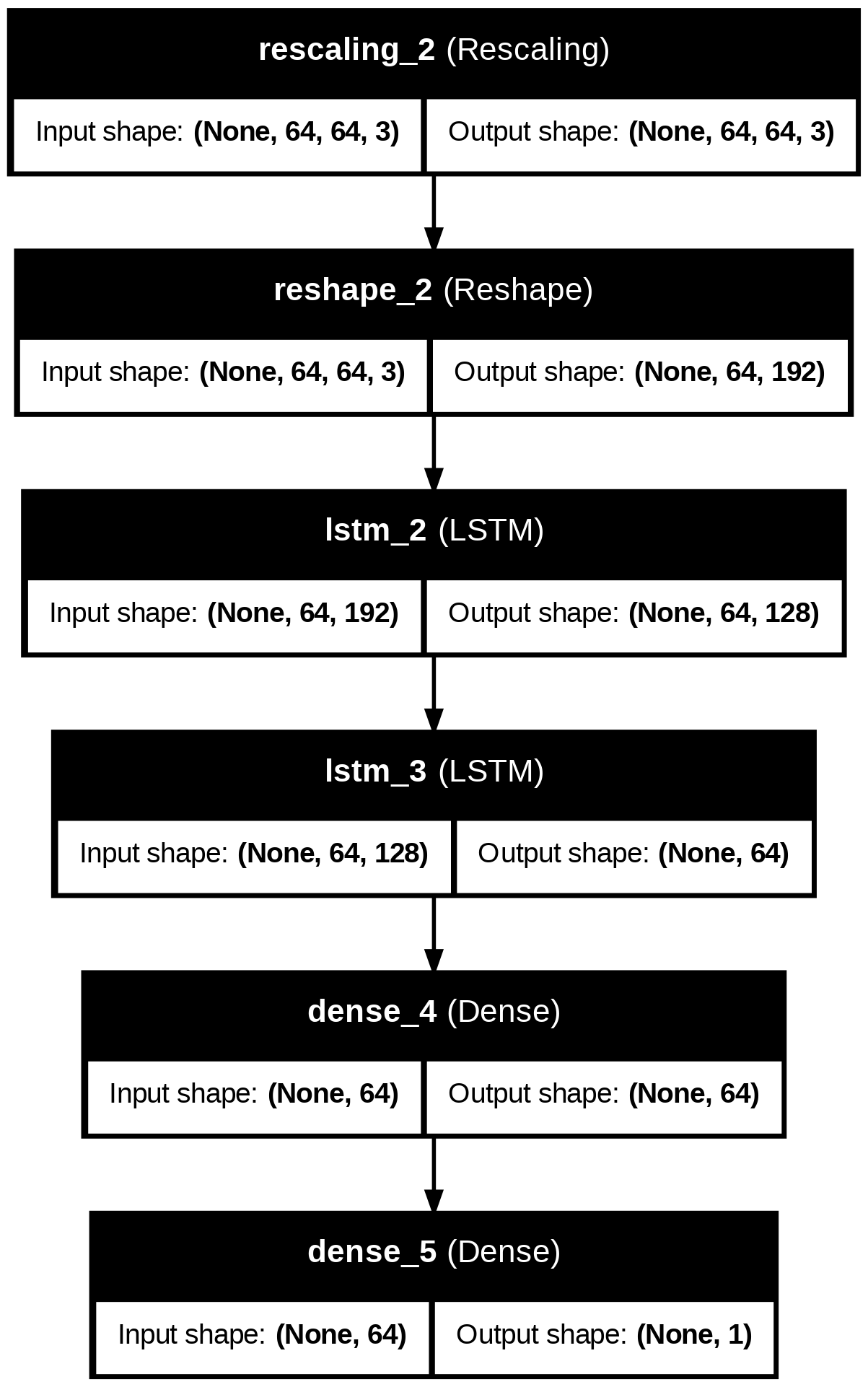


Рисунок – Архитктура двухслойной LSTM сети

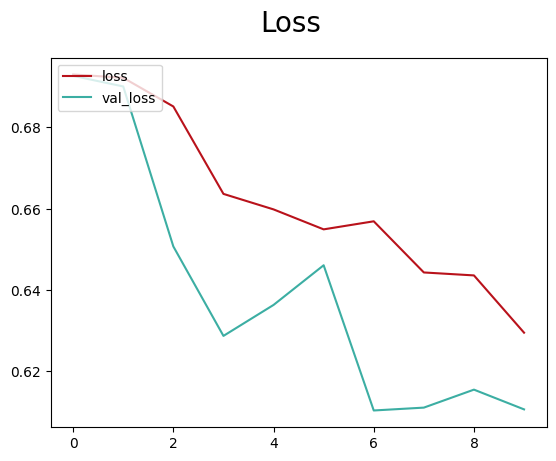


Рисунок – График потерь для двухслойной LSTM сети

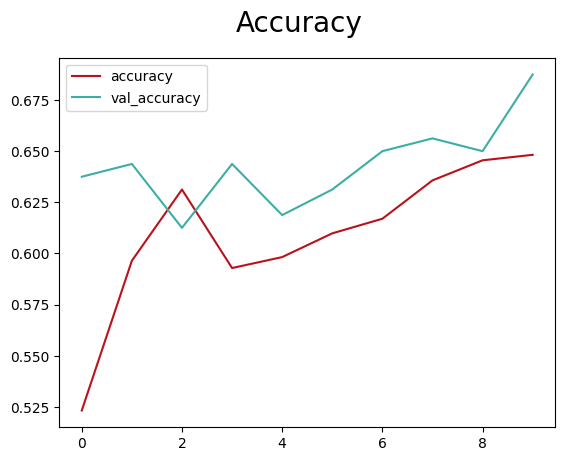


Рисунок – График точности для двухслойной LSTM сети

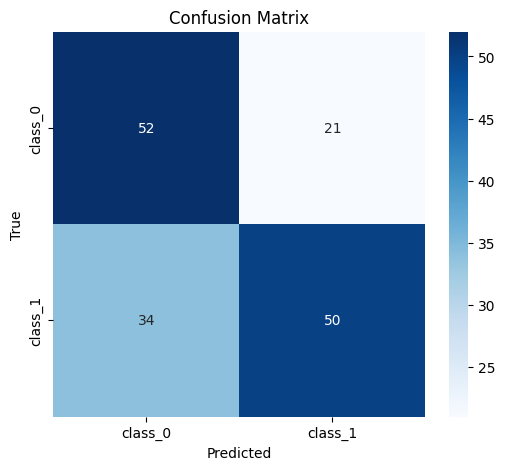


Рисунок 8 – Матрица ошибок LSTM сети

**GRU сеть.**

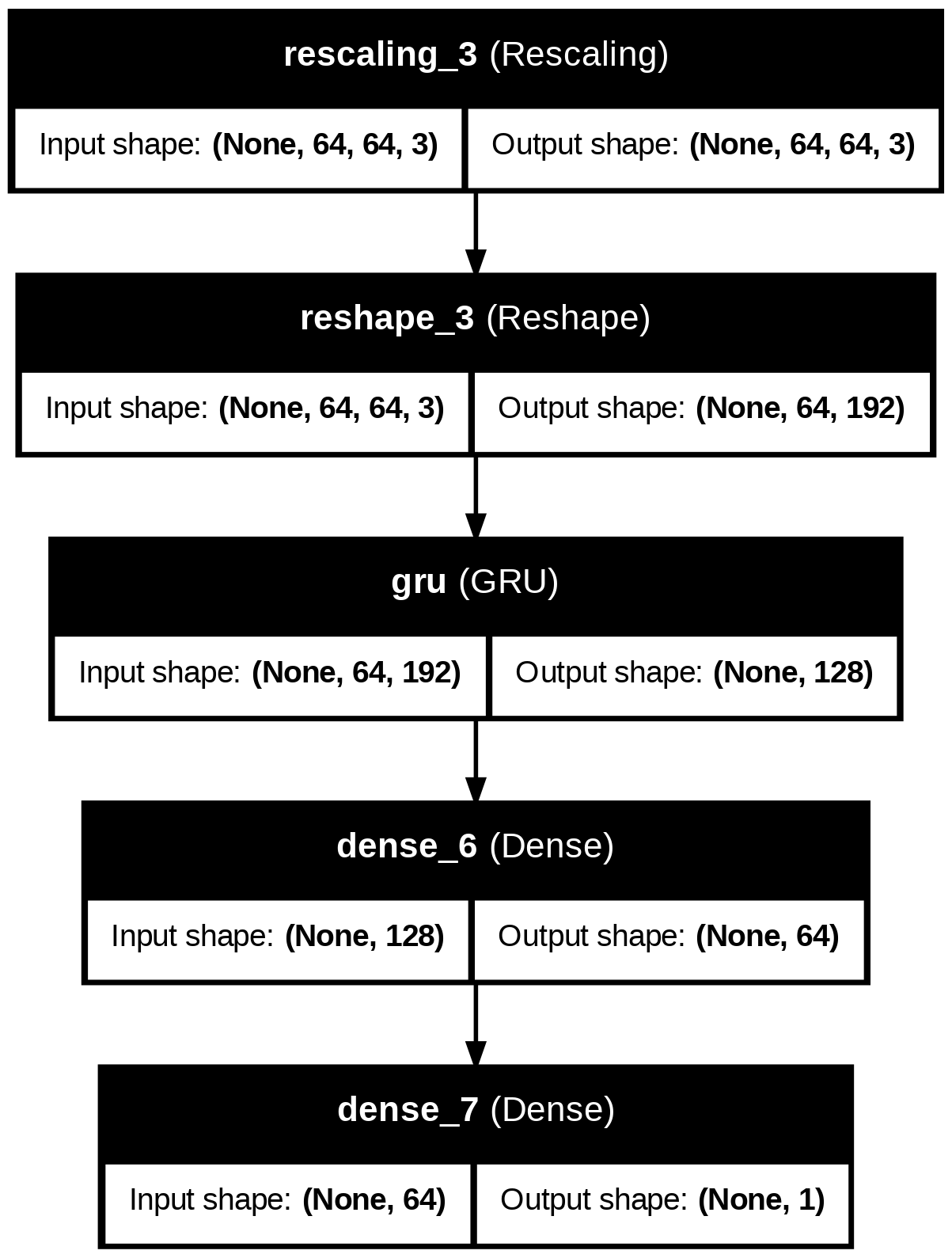


Рисунок 9 Архитектура GRU сети

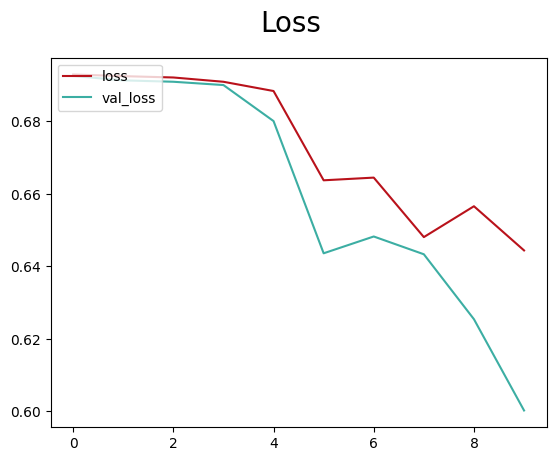


Рисунок – График потерь для GRU сети

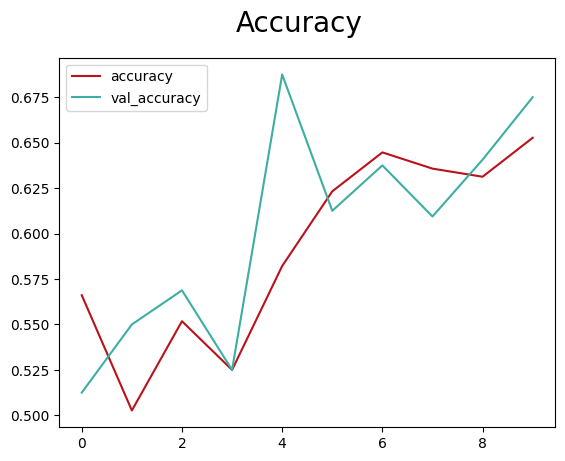


Рисунок – График точности для GRU сети

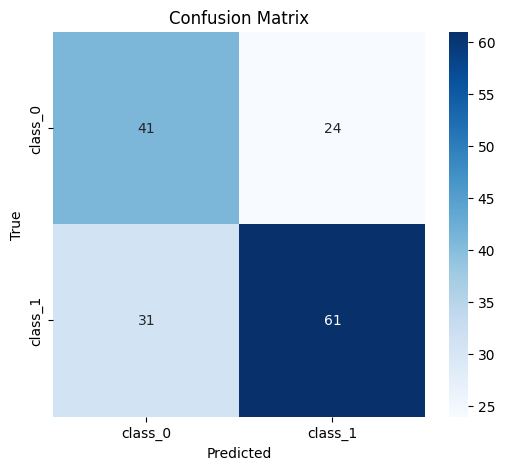


Рисунок – Матрица ошибок GRU сети

**8. Результаты экспериментов**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Модель | Эпох | Точность (accuracy, %) | Precision (%) | Recall (%) | Время (с) |
| Простая рекуррентная LSTM сеть | 10 | 64 | 52 | 57 | 630 |
| Двухслойная LSTM | 10 | 70 | 60 | 65 | 653 |
| GRU сеть | 10 | 72 | 66 | 65 | 607 |

Выводы:

**GRU-сеть показывает наилучшие метрики по всем показателям, включая точность (72%) и precision (66%), при наименьшем времени обучения. Это делает её лучшей среди рекуррентных моделей.**

**Двухслойная LSTM демонстрирует улучшение по сравнению с простой LSTM как в точности, так и в других метриках,**

**Простая LSTM — наименее эффективная: самая низкая точность и recal.**

Вывод: также как и полносвязные сети, как и сверточные сети, рекуррентные модели при увеличении слоев показывает лучшие показатели, при этом GRU сеть среди рекуррентных на нашем наборе данных показывает себя наилучшим образом, при этом в целом показывая хуже результат чем сверточные сети.