**1. Постановка задачи**

Цель: провести начальную настройку весов для дальнейшего использования при обучении полноценной сети. Как и в предидущих работах использовался датасет doom-crossing, в котором есть два класса:

* Doom
* Animal Crossing

Пусть имеется обучающая выборка из изображений:

Требуется найти функцию

где θ — параметр нейросети, минимизирующие функцию потерь.

Автокодировщик состоит из двух частей: энкодера  и декодера . Энкодер переводит входной сигнал в его представление (код): , а декодер восстанавливает сигнал по его коду: . Автокодировщик, изменяя  и , стремится выучить тождественную функцию , минимизируя какой-то функционал ошибки.

В рамках обучения использовался оптимизатор adam.

Функция потерь, бинарная кросэнтропия:

В качестве функции активации использовалась сигмойда:

В качестве метода обучения без учителя для настройки начальных значений весов был выбран автокодировщик. Основная сеть, для которою мы будем использовать это полносвязная сеть, это модель из лабораторной работы по полносвязным сетям.

**2. Тренировочные и тестовые наборы данных**

|  |  |
| --- | --- |
| **Характеристика** | **Значение** |
| Количество классов | 2 |
| Названия классов | doom, animal\_crossing |
| Общее количество примеров | 1597 изображений |
| Размер обучающих данных | 80% |
| Размер валидационных данных | 20% |
| 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 и препроцессинг для автокодировщика

train\_ds\_unsup = keras.utils.image\_dataset\_from\_directory(

    '/content/doom-crossing',

    labels=None,                # без меток

    image\_size=(64, 64),

    batch\_size=32,

    shuffle=True

)

# Нормализуем пиксели в [0,1]

train\_ds\_unsup = train\_ds\_unsup.map(lambda x: x / 255.0)

# Для обучения автоэнкодера подготовим пары (X, X)

train\_ds\_unsup = train\_ds\_unsup.map(lambda x: (x, x))

Задаем архитектуру энкодера и декодера и собираем автоэнкодер

# Энкодер (кодировщик)

encoder = keras.Sequential([

    layers.Input(shape=(64, 64, 3)),

    layers.Conv2D(32, (3, 3), activation='relu', padding='same'),

    layers.MaxPooling2D((2, 2), padding='same'),

    layers.Conv2D(64, (3, 3), activation='relu', padding='same'),

    layers.MaxPooling2D((2, 2), padding='same'),

    layers.Conv2D(128, (3, 3), activation='relu', padding='same'),

    layers.MaxPooling2D((2, 2), padding='same'),

], name='encoder')

# Декодер (дешифратор)

decoder = keras.Sequential([

    layers.Conv2DTranspose(128, (3, 3), strides=2, activation='relu', padding='same'),

    layers.Conv2DTranspose(64, (3, 3), strides=2, activation='relu', padding='same'),

    layers.Conv2DTranspose(32, (3, 3), strides=2, activation='relu', padding='same'),

    layers.Conv2D(3, (3, 3), activation='sigmoid', padding='same')  # выходные пиксели [0,1]

], name='decoder')

# Полный автоэнкодер

input\_img = keras.Input(shape=(64, 64, 3))

encoded = encoder(input\_img)

decoded = decoder(encoded)

autoencoder = keras.Model(input\_img, decoded, name='autoencoder')

autoencoder.compile(optimizer='adam', loss='binary\_crossentropy')

Обучаем наш автоэнкодер

autoencoder.fit(

    train\_ds\_unsup,

    epochs=10,

    shuffle=True

)

# Модель `encoder` теперь содержит обученные веса

encoder\_model = encoder  # кодировщик для дальнейшего использования

Для сверточной сети подготовим датасет.

data = tf.keras.utils.image\_dataset\_from\_directory('/content/doom-crossing', image\_size=(64, 64))

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

Задаем архитектуру модели

# Классификатор использует обученный encoder + наша сетка

classifier = keras.Sequential([

    encoder\_model,

    layers.Flatten(),

    layers.Dense(512, activation='relu'),

    layers.Dense(256, activation='relu'),

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

], name='classifier')

classifier.compile(

      optimizer=tf.keras.optimizers.Adam(learning\_rate=0.0001),

      loss='binary\_crossentropy',

      metrics=[

          tf.keras.metrics.BinaryAccuracy(name='accuracy'),

          tf.keras.metrics.Precision(name='precision'),

          tf.keras.metrics.Recall(name='recall')

      ]

    )

    )

classifier\_rnn = keras.Sequential([

encoder\_model,

layers.Reshape((8, 8 \* 128)), # → (8, 1024)

layers.LSTM(128), # берет последовательность из 8 шагов

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

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

], name='classifier\_rnn')

classifier\_rnn.compile(

optimizer=tf.keras.optimizers.Adam(learning\_rate=0.0001),

loss='binary\_crossentropy',

metrics=[

tf.keras.metrics.BinaryAccuracy(name='accuracy'),

tf.keras.metrics.Precision(name='precision'),

tf.keras.metrics.Recall(name='recall')

]

)

**7. Тестовые конфигурации сетей.**

В рамках лабораторной работы было проведено два эксперимента в моделью из второй лабораторной работы по полносвязным сетям и из четвертой по рекуррентным, воспользуемся обученным автоэнкодером, чтобы настроить начальные веса.

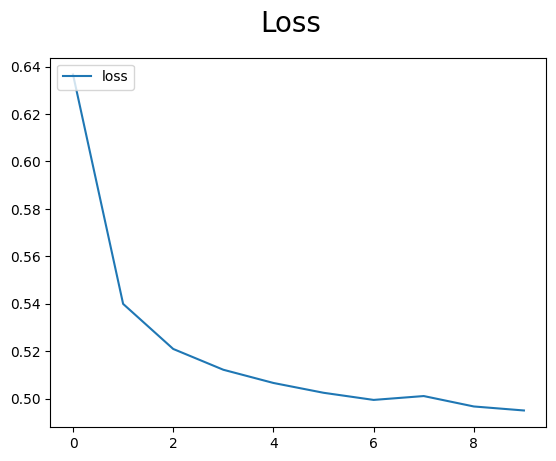
****

Рисунок – Функция потерь автоэнкодера

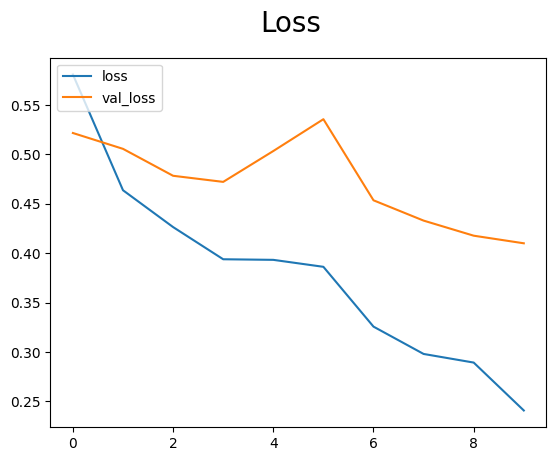


Рисунок – Функция потерь классификатора

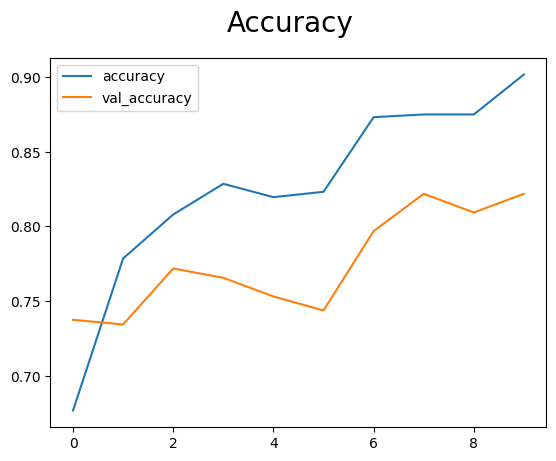


Рисунок – График точности классификатора

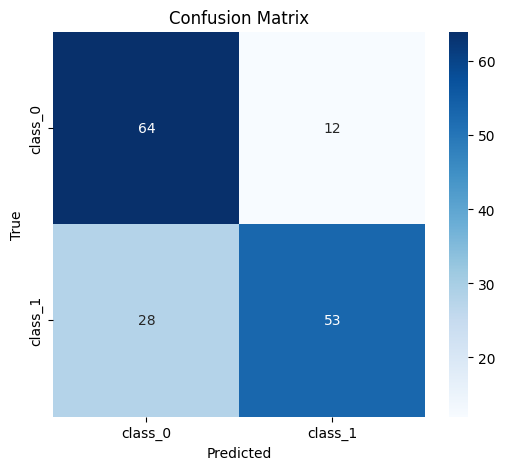


Рисунок – Матрица ошибок классификатора

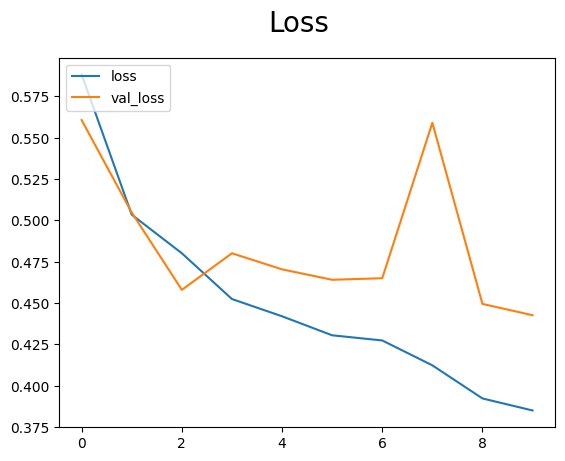


Рисунок – График потерь рекуррентного классификатора

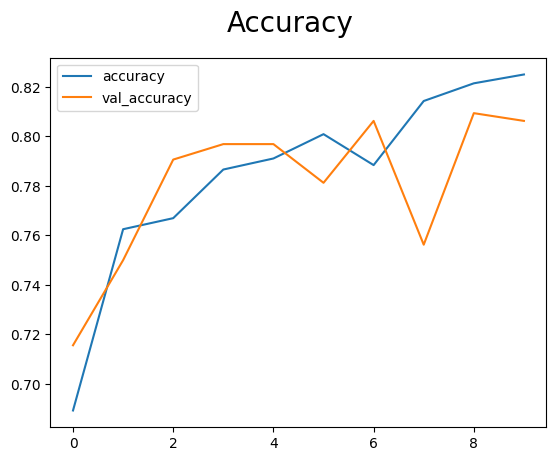


Рисунок – График точности рекуррентного классификатора

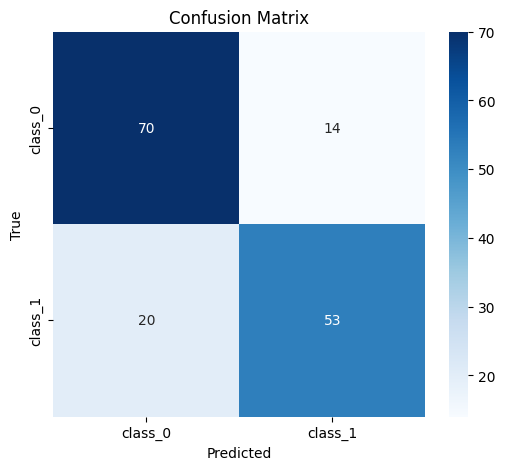


Рисунок – Матрица ошибок рекукрунетого классификатора

**8. Результаты экспериментов**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Модель | Эпох | Точность (accuracy, %) | Precision (%) | Recall (%) | Время (с) |
| Простая полносвязная сеть | 10 | 67 | 77 | 66 | 635 |
| Полносвязная сеть с автоэнкодером | 10 | 82 | 85 | 75 | 197 |
| Рекуррентная сеть | 10 | 64 | 52 | 57 | 630 |
| Рекуррентная сеть с автоэнкодером | 10 | 79 | 73 | 78 | 563 |

Выводы: При использовании метода обучения без учителя при помощи автокодировщика, можно получить модель, которая будет предсказывать с большей точностью, а также сократить время для обучения для итогового классификатора.