**1. Постановка задачи**

Цель: построить модель сверточной 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 простой сверточной сети

model1 = Sequential([

    tf.keras.layers.Conv2D(16, (3,3), 1, activation='relu', input\_shape=IMG\_SHAPE),

    tf.keras.layers.MaxPooling2D(),

    tf.keras.layers.Conv2D(32, (3,3), 1, activation='relu'),

    tf.keras.layers.MaxPooling2D(),

    tf.keras.layers.Conv2D(16, (3,3), 1, activation='relu'),

    tf.keras.layers.MaxPooling2D(),

    tf.keras.layers.Flatten(),

    tf.keras.layers.Dense(256, 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\_model(model, train, val, initial\_epochs, callbacks):

    model.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')

      ]

    )

    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()

def test\_model(model, test):

    y\_true = []

    y\_pred = []

    precision = Precision()

    recall = Recall()

    binary\_accuracy = BinaryAccuracy()

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

        X, y = batch

        yhat = model.predict(X)

        y\_true.extend(y)

        y\_pred.extend((yhat > 0.5).astype(int))

        precision.update\_state(y, yhat)

        recall.update\_state(y, yhat)

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

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

    cm = confusion\_matrix(y\_true, y\_pred)

    labels = test.class\_names if hasattr(test, 'class\_names') else ['class\_0', 'class\_1']

    plt.figure(figsize=(6, 5))

    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=labels, yticklabels=labels)

    plt.xlabel('Predicted')

    plt.ylabel('True')

    plt.title('Confusion Matrix')

    plt.show()

Построение сверточной нейронной сети с применением BatchNorm

model2 = tf.keras.Sequential([

    tf.keras.layers.Conv2D(16, (3,3), 1, activation='relu', input\_shape=IMG\_SHAPE),

    tf.keras.layers.BatchNormalization(),

    tf.keras.layers.MaxPooling2D(),

    tf.keras.layers.Conv2D(32, (3,3), 1, activation='relu'),

    tf.keras.layers.BatchNormalization(),

    tf.keras.layers.MaxPooling2D(),

    tf.keras.layers.Conv2D(64, (3,3), 1, activation='relu'),

    tf.keras.layers.BatchNormalization(),

    tf.keras.layers.MaxPooling2D(),

    tf.keras.layers.Conv2D(128,(3,3), 1, activation='relu'),

    tf.keras.layers.BatchNormalization(),

    tf.keras.layers.MaxPooling2D(),

    tf.keras.layers.Flatten(),

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

    tf.keras.layers.Dropout(0.5),

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

])

Построение другой модели которая вдохновлена сетью alexnet

alexnet = Sequential([

     tf.keras.layers.Conv2D(64, 11, strides=(4,4), activation='relu', input\_shape=IMG\_SHAPE),

     tf.keras.layers.MaxPooling2D(),

     tf.keras.layers.Conv2D(256, 5, padding='same', activation='relu'),

     tf.keras.layers.MaxPooling2D(),

     tf.keras.layers.Conv2D(384, 3, padding='same', activation='relu'),

     tf.keras.layers.Conv2D(384, 3, padding='same', activation='relu'),

     tf.keras.layers.Conv2D(256, 3, padding='same', activation='relu'),

     tf.keras.layers.MaxPooling2D(),

     tf.keras.layers.Flatten(),

     tf.keras.layers.Dense(4096, activation='relu'),

     tf.keras.layers.Dropout(0.5),

     tf.keras.layers.Dense(4096, activation='relu'),

     tf.keras.layers.Dropout(0.5),

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

])

**7. Тестовые конфигурации сетей.**

**Первая модель**

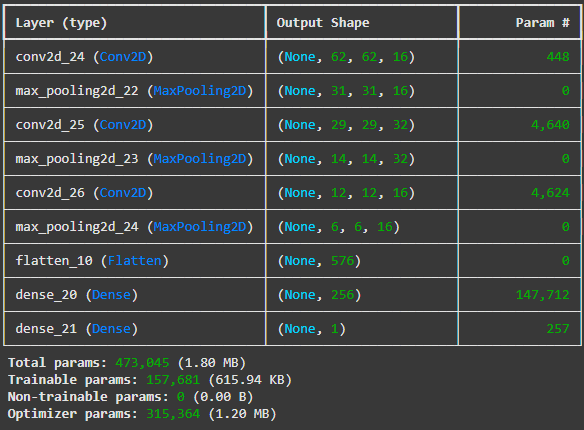


Рисунок – Архитектура простой сверточной сети

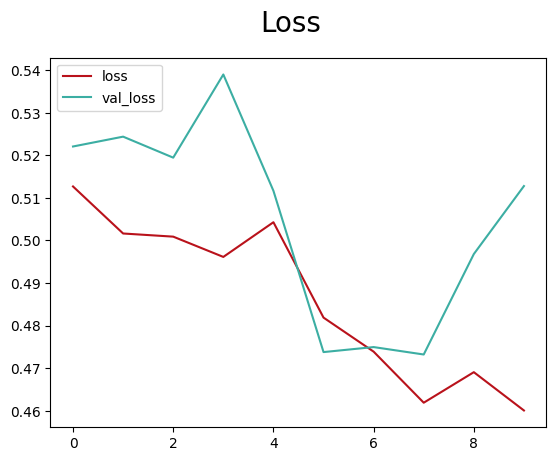


Рисунок 2 – График потерь по эпохам

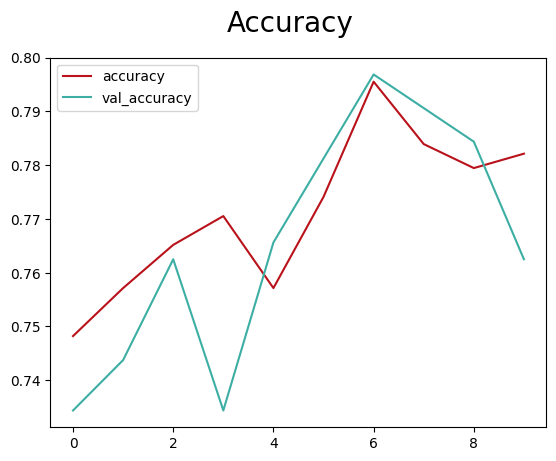


Рисунок 3 – График расчета точности по эпохам при обучении

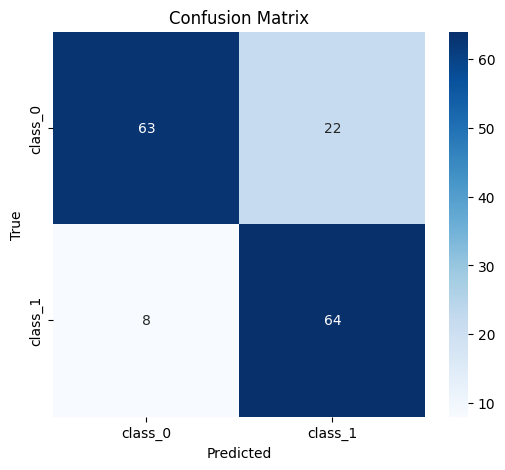


Рисунок 4 – Матрица ошибок

**Вторая модель**

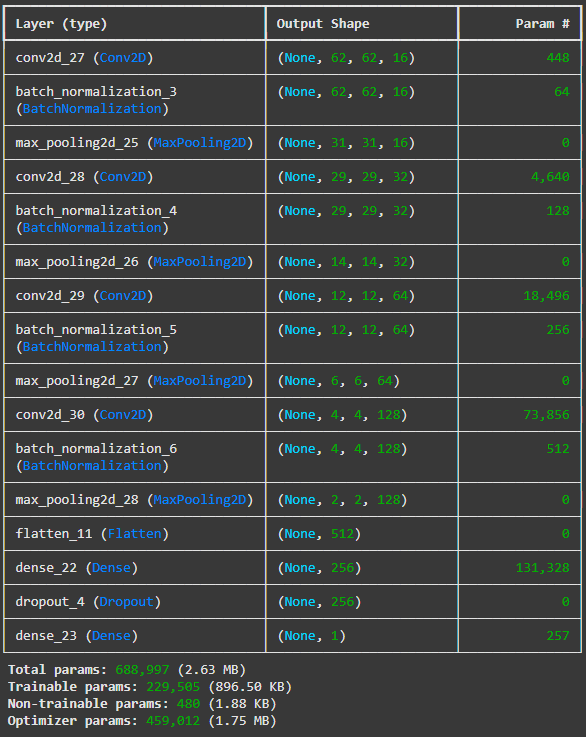


Рисунок – Архитектура прокаченной сверточной сети

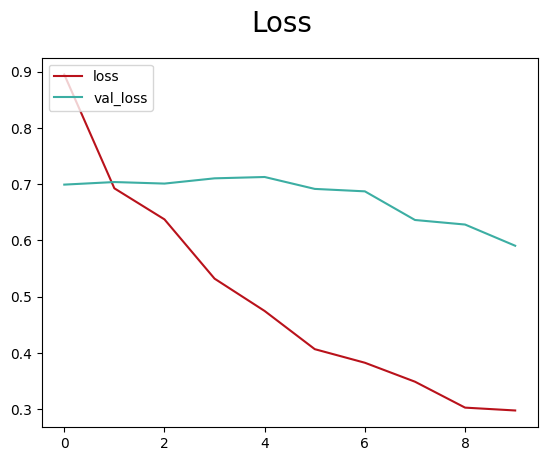


Рисунок 6 – График потерь по эпохам

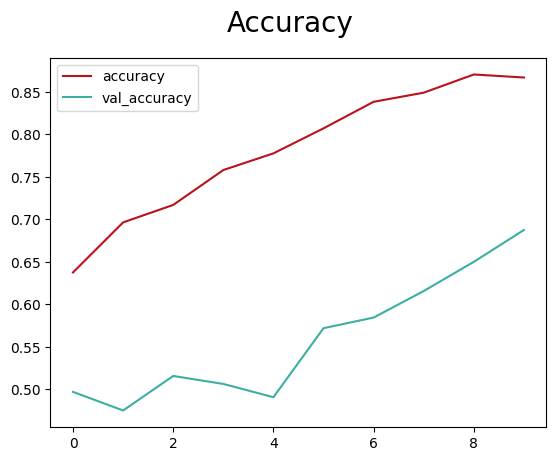


Рисунок 7 – График расчета точности по эпохам при обучении

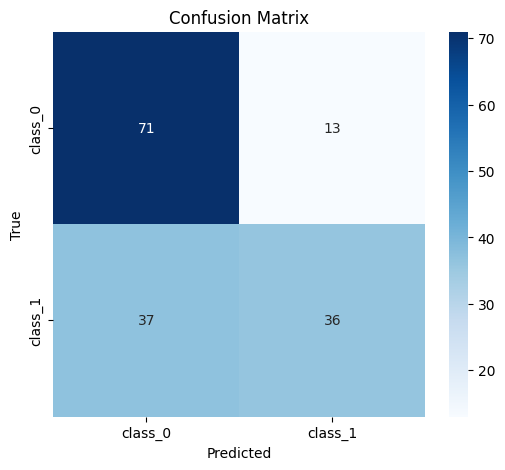


Рисунок 8 – Матрица ошибок

**Третья модель**

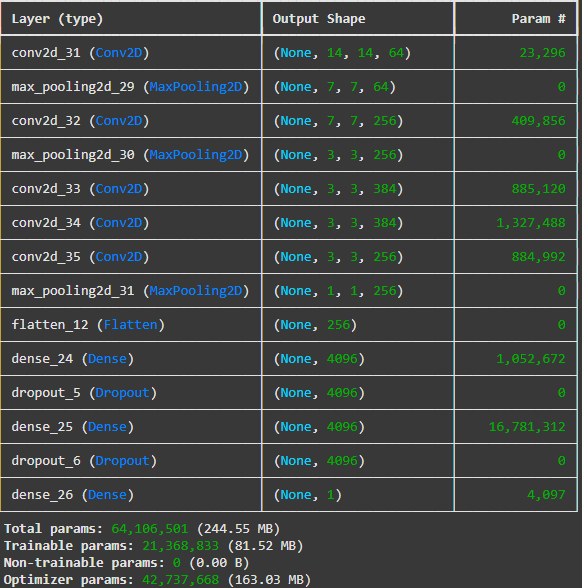


Рисунок Архитектура сверточной сети по типу alexnet

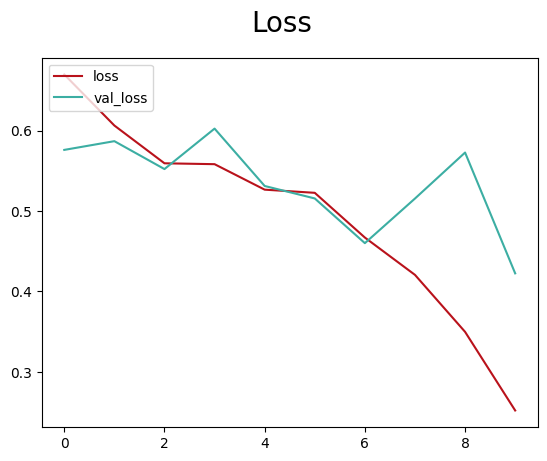


Рисунок 10 – График потерь по эпохам

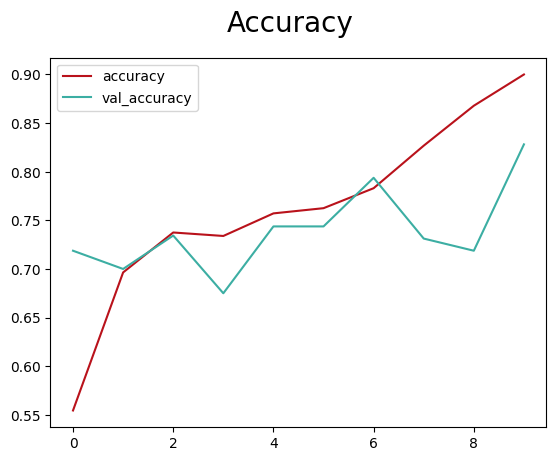


Рисунок 11 – График расчета точности по эпохам при обучении

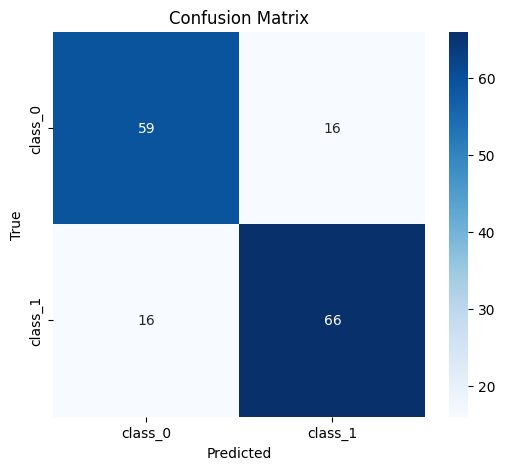


Рисунок 12 – Матрица ошибок модели 3

**8. Результаты экспериментов**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Модель | Эпох | Точность (accuracy, %) | Precision (%) | Recall (%) | Время (с) |
| Простая сверточная сеть | 10 | 74 | 89 | 81 | 627 |
| Сверточная сеть с примененим слоев нормализацией батча | 10 | 73 | 49 | 68 | 702 |
| Сверточная по типу alexnet | 10 | 80 | 80 | 80 | 897 |

Выводы:

**Сверточная сеть по типу AlexNet показала лучшие показатели по точности и сбалансированные значения precision и recall (80%), что делает её самой стабильной моделью для задачи классификации. Однако время обучение было нибольшим (897 секунд).**

**Простая сверточная сеть обеспечивает высокую precision (89%), что говорит о малом количестве ложных срабатываний. Она обучается быстрее AlexNet, но менее точна (74%) и хуже по recall.**

**Сеть с batch normalization показала неожиданно низкую precision (49%), что может говорить о плохой калибровке модели или переобучении. При этом время обучения больше, чем у простой сети, а эффективность — ниже.**

Вывод: аналогично с полносвязными сетям чем больше слоев, тем лучше. При этом сама архитекутра сверточных сетей показывает себя лучше на там же датасете, чем полносвязные сети