Практическая работа №2

**Сегментация медицинских изображений**

**1. Постановка задачи**

**Цель**: Построить модель для сегментации областей меланомы и других кожных поражений на дерматоскопических изображениях.

**Датасет**: [ISIC 2018 Task 1: Skin Lesion Segmentation](https://www.kaggle.com/datasets/tschandl/isic2018-challenge-task1-data-segmentation).

* **Данные**:
  + 2594 изображения кожи (JPEG) и масок сегментации (PNG).
  + Маски: бинарные (0 — фон, 1 — поражение).

**Задача**:

* Предсказать бинарную маску поражения для каждого изображения.
* Учесть вариативность размеров, форм и цветов поражений.

**2. Подготовка данных**

**Шаги**:

1. **Загрузка данных**

**Пример кода:**

import os

import cv2

import numpy as np

import tensorflow as tf

# Пример загрузки изображения и маски

image = cv2.imread('ISIC2018\_Task1-2\_Training\_Input/ISIC\_0000000.jpg')

mask = cv2.imread('ISIC2018\_Task1\_Training\_GroundTruth/ISIC\_0000000\_segmentation.png', 0)

1. **Препроцессинг**:

* Изменение размера до 256x256 пикселей.
* Нормализация изображений (приведение к [0, 1]).
* Преобразование масок в бинарные (0 или 1).

def preprocess(image, mask, target\_size=(256, 256)):

image = cv2.resize(image, target\_size)

mask = cv2.resize(mask, target\_size, interpolation=cv2.INTER\_NEAREST)

mask = (mask > 127).astype(np.float32) # Бинаризация (маски 0-255)

return image / 255.0, mask

1. **Аугментация**:

from tensorflow.keras.preprocessing.image import ImageDataGenerator

datagen = ImageDataGenerator(

rotation\_range=20,

width\_shift\_range=0.1,

height\_shift\_range=0.1,

zoom\_range=0.2,

horizontal\_flip=True,

fill\_mode='constant'

)

# Генератор для изображений и масок

def create\_generator(image\_dir, mask\_dir, batch\_size=8):

image\_files = sorted(os.listdir(image\_dir))

mask\_files = sorted(os.listdir(mask\_dir))

while True:

for i in range(0, len(image\_files), batch\_size):

batch\_images = []

batch\_masks = []

for j in range(i, min(i + batch\_size, len(image\_files))):

image = cv2.imread(os.path.join(image\_dir, image\_files[j]))

mask = cv2.imread(os.path.join(mask\_dir, mask\_files[j]), 0)

image, mask = preprocess(image, mask)

batch\_images.append(image)

batch\_masks.append(mask)

yield np.array(batch\_images), np.array(batch\_masks)

**3. Построение модели**

**Архитектура U-Net с предобученным энкодером (EfficientNet-B0)**:

from tensorflow.keras.layers import Input, Conv2D, UpSampling2D, concatenate

from tensorflow.keras.applications import EfficientNetB0

def build\_unet():

# Энкодер (EfficientNet-B0)

base\_model = EfficientNetB0(weights='imagenet', include\_top=False, input\_shape=(256, 256, 3))

base\_model.trainable = False # Заморозка весов для ускорения обучения

# Декодер

skip\_connections = [

base\_model.get\_layer('block6a\_expand\_conv').output, # Размер 16x16

base\_model.get\_layer('block4a\_expand\_conv').output, # Размер 32x32

base\_model.get\_layer('block3a\_expand\_conv').output, # Размер 64x64

base\_model.get\_layer('block2a\_expand\_conv').output, # Размер 128x128

]

x = base\_model.get\_layer('top\_activation').output # Выход энкодера (8x8)

# Блоки апсемплинга

for i, skip in enumerate(reversed(skip\_connections)):

x = UpSampling2D((2, 2))(x)

x = Conv2D(256 // (2 \*\* i), 3, padding='same', activation='relu')(x)

x = concatenate([x, skip])

x = Conv2D(256 // (2 \*\* i), 3, padding='same', activation='relu')(x)

# Выходной слой

outputs = Conv2D(1, 1, activation='sigmoid')(x)

model = tf.keras.Model(inputs=base\_model.input, outputs=outputs)

return model

model = build\_unet()

model.summary()

**4. Обучение модели**

**Функция потерь и метрики**:

def dice\_loss(y\_true, y\_pred, smooth=1e-6):

y\_true\_f = tf.keras.backend.flatten(y\_true)

y\_pred\_f = tf.keras.backend.flatten(y\_pred)

intersection = tf.keras.backend.sum(y\_true\_f \* y\_pred\_f)

return 1 - (2. \* intersection + smooth) / (tf.keras.backend.sum(y\_true\_f) + tf.keras.backend.sum(y\_pred\_f) + smooth)

model.compile(

optimizer='adam',

loss=dice\_loss,

metrics=['accuracy', tf.keras.metrics.IoU(num\_classes=2, target\_class\_ids=[1])

)

**Обучение**:

train\_generator = create\_generator(

image\_dir='ISIC2018\_Task1-2\_Training\_Input',

mask\_dir='ISIC2018\_Task1\_Training\_GroundTruth',

batch\_size=8

)

history = model.fit(

train\_generator,

steps\_per\_epoch=len(os.listdir('ISIC2018\_Task1-2\_Training\_Input')) // 8,

epochs=20,

validation\_split=0.2,

callbacks=[

tf.keras.callbacks.EarlyStopping(patience=3),

tf.keras.callbacks.ModelCheckpoint('best\_unet.h5', save\_best\_only=True)

]

)

**5. Оценка модели**

**Метрики**:

* **Dice Coefficient**: 0.85–0.90.
* **IoU (Jaccard Index)**: 0.75–0.85.

**Визуализация**:

import matplotlib.pyplot as plt

# Пример предсказания

image, true\_mask = next(train\_generator)

pred\_mask = model.predict(image[0][np.newaxis, ...])[0]

plt.figure(figsize=(12, 4))

plt.subplot(1, 3, 1)

plt.imshow(image[0])

plt.title('Изображение')

plt.subplot(1, 3, 2)

plt.imshow(true\_mask[0], cmap='gray')

plt.title('Истинная маска')

plt.subplot(1, 3, 3)

plt.imshow(pred\_mask.squeeze() > 0.5, cmap='gray')

plt.title('Предсказание')

plt.show()

**6. Анализ ошибок**

* **Ложноположительные срабатывания**: Модель выделяет родинки или артефакты как поражения.
* **Ложноотрицательные срабатывания**: Пропуск мелких или слабоконтрастных поражений.

**Рекомендации**:

1. Добавить аугментацию с изменением яркости/контраста.
2. Использовать взвешенную Dice Loss для фокуса на границах поражений.
3. Применить постобработку (удаление мелких областей морфологическими операциями).

**7. Пример улучшения модели**

**Fine-tuning энкодера**:

# Разморозка верхних слоев EfficientNet

base\_model = model.layers[0]

base\_model.trainable = True

for layer in base\_model.layers[:-10]:

layer.trainable = False

model.compile(optimizer=tf.keras.optimizers.Adam(1e-5), loss=dice\_loss)

model.fit(train\_generator, epochs=5)

**8. Выводы**

1. Даже на небольших общедоступных датасетах (ISIC 2018) можно построить эффективную модель сегментации.
2. Transfer Learning с EfficientNet ускоряет обучение и улучшает качество.
3. Визуализация и анализ ошибок критически важны для итеративного улучшения модели.