МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ
ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по индивидуальному заданию
по дисциплине «Искусственные нейронные сети»
Тема: Сегментация снимков компьютерной
томографии лёгких.

Студентгр.7382 _____ Токарев А.П.

Студентгр.7382 _____ Чигалейчик А.С.

Преподаватель _____ Жукова Н. А.

Санкт-
Петербург
2020

Цель работы.

Выделить лёгкие на снимках компьютерной томографии.

Постановка задачи.

В данном индивидуальном задании необходимо реализовать нейросеть, которая по снимку компьютерной томографии лёгких получает маску изображения, где эти самые лёгкие выделены.

Выполнение работы.

1. **Представление данных.**

В работе использовался следующий датасет: 267 снимков КТ в формате .tif, 267 масок с выделенными лёгкими в формате .tif.

Так как датасет достаточно небольшой, было решено провести аугментацию данных с помощью функции Keras ImageDataGenerator, для этого была написана следующая функция:

```
def generator(x_train, y_train, batch_size):
    data_generator = ImageDataGenerator(
                width_shift_range=0.1,
                height_shift_range=0.1,
                rotation_range=10,
                zoom_range=0.1).flow(x_train, x_train, batch_size, seed=SEED)
    mask_generator = ImageDataGenerator(
                width_shift_range=0.1,
                height_shift_range=0.1,
                rotation_range=10,
                zoom_range=0.1).flow(y_train, y_train, batch_size, seed=SEED)
    while True:
          x_batch, _ = data_generator.next()
          y_batch, _ = mask_generator.next()
          yield x_batch, y_batch
```

Рисунок 1 — Генератор данных

Как видно, сначала путём геометрических преобразований и небольшого масштабирования генерируются новые изображения КТ, затем из соответствующих им масок генерируются точно такими же преобразованиями маски для новых изображений.

Дабы убедиться в том, что маски действительно совпадают с соответствующими им изображениями, выведен несколько с помощью следующего кода:

```
image_batch, mask_batch = next(generator(x_train, y_train, 8))
fix, ax = plt.subplots(8,2, figsize=(8,20))
for i in range(8):
      ax[i,0].imshow(image_batch[i,:,:,0])
      ax[i,1].imshow(mask_batch[i,:,:,0])
plt.show()
```

Рисунок 2 — Вывод генератора

Получаем следующий вывод:

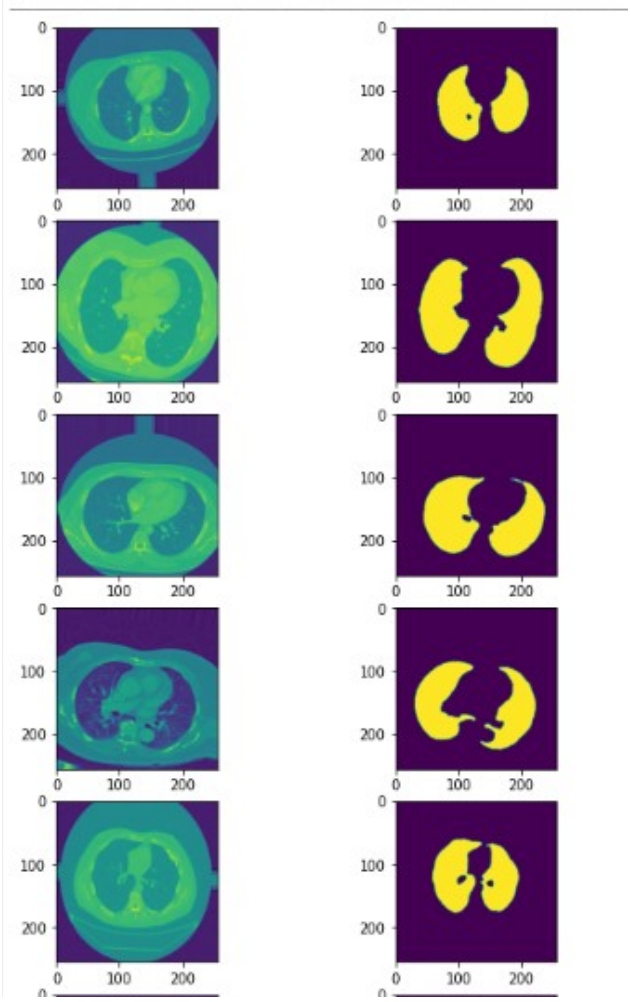

Рисунок 3 — Сгенерированные изображения и маски (часть 1).

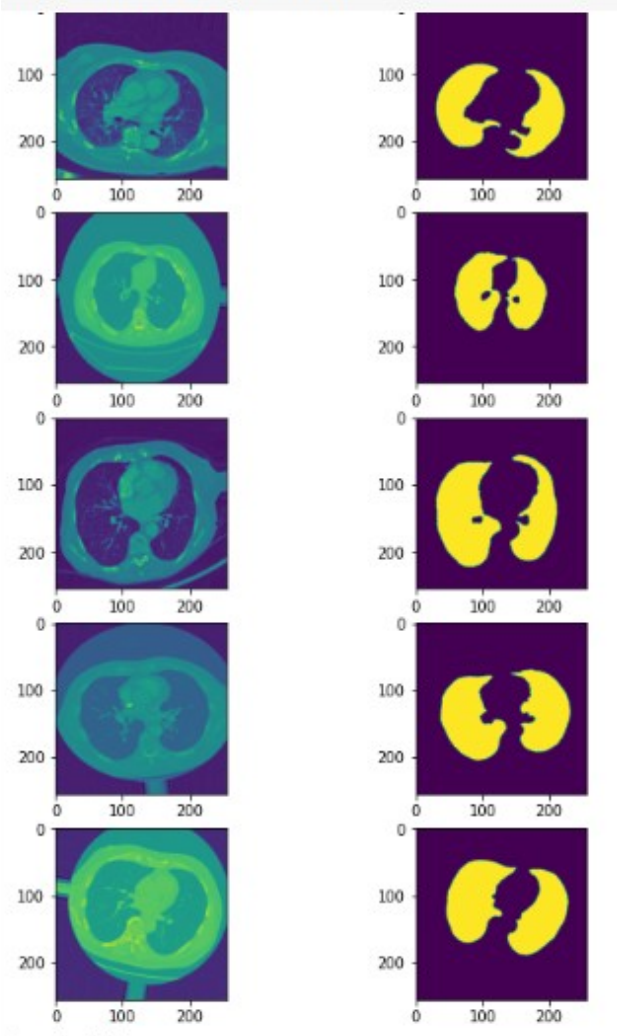

Рисунок 4 — Сгенерированные изображения и маски (часть 2).

Непосредственно подгрузка данных и их выравнивание по размеру осуществляется следующим образом с помощью утилиты openCV (пакет cv2):

```
IMAGE_LIB = './data/2d_images/'
MASK_LIB = './data/2d_masks/'
IMG_HEIGHT, IMG_WIDTH = 256, 256
SEED=42
all_images = [x for x in sorted(os.listdir(IMAGE_LIB)) if x[-4:] == '.tif']
x_data = np.empty((len(all_images), IMG_HEIGHT, IMG_WIDTH), dtype='float32')
for i, name in enumerate(all_images):
    im = cv2.imread(IMAGE_LIB + name,
cv2.IMREAD_UNCHANGED).astype("int16").astype('float32')
    im = cv2.resize(im, dsize=(IMG_WIDTH, IMG_HEIGHT), interpolation=cv2.INTER_LANCZOS4)
    im = (im - np.min(im)) / (np.max(im) - np.min(im))
    x_data[i] = im
y_data = np.empty((len(all_images), IMG_HEIGHT, IMG_WIDTH), dtype='float32')
for i, name in enumerate(all_images):
    im = cv2.imread(MASK_LIB + name, cv2.IMREAD_UNCHANGED).astype('float32')/255.
    im = cv2.resize(im, dsize=(IMG_WIDTH, IMG_HEIGHT), interpolation=cv2.INTER_NEAREST)
    y_data[i] = im
```

Рисунок 5 — Функция загрузки и подготовки данных

Проверим, что получили после обработки:

```
fig, ax = plt.subplots(1,2, figsize = (8,4))
ax[0].imshow(x_data[0], cmap='gray')
ax[1].imshow(y_data[0], cmap='gray')
plt.show()
```

Рисунок 6 — Вывод примера обработанных данных

Получаем следующее изображение:

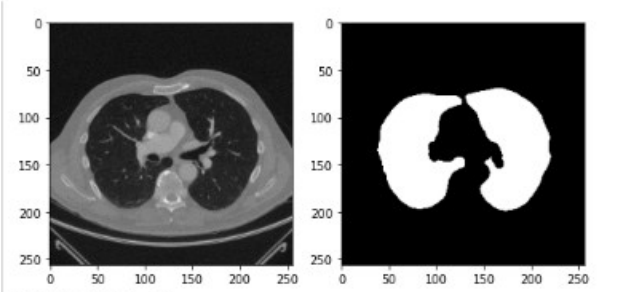

Рисунок 7 — Обработанное изображение и его маска

Итак, данные подготовлены.

2. Подключение GPU

Для ускорения работы сети была предпринята попытка использовать видеокарту, для этого были обновлены драйверы nvidia, установлены драйверы CUDA и необходимые для этого библиотеки, но из — за множества ошибок несовпадения версий (tensorflow и cuda) было принято решение использовать сервис google collab, где подключить GPU можно в несколько простых шагов.

3. Выбор метрики

На основе изученной информации было установлено, что для оценки работы нейронных сетей при задаче сегментации изображений часто используется мера Жаккара, которая задаётся следующей формулой:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}.$$

Иначе говоря, множество пересечений правильных масок и полученных масок делится на множество их объединений.

Реализуем данную метрику:

```
def dice_coef(y_true, y_pred):
    y_true_f = K.flatten(y_true)
    y_pred_f = K.flatten(y_pred)
    intersection = K.sum(y_true_f * y_pred_f)
    return (2. * intersection + K.epsilon()) / (K.sum(y_true_f) + K.sum(y_pred_f) + K.epsilon())
```

Рисунок 8 — Метрика Жаккара

4. Выбор архитектуры

Для выполнения поставленной задачи была выбрана свёрточная нейронная сеть по типу U-Net.

Общая схема данной сети представлена на картинке:

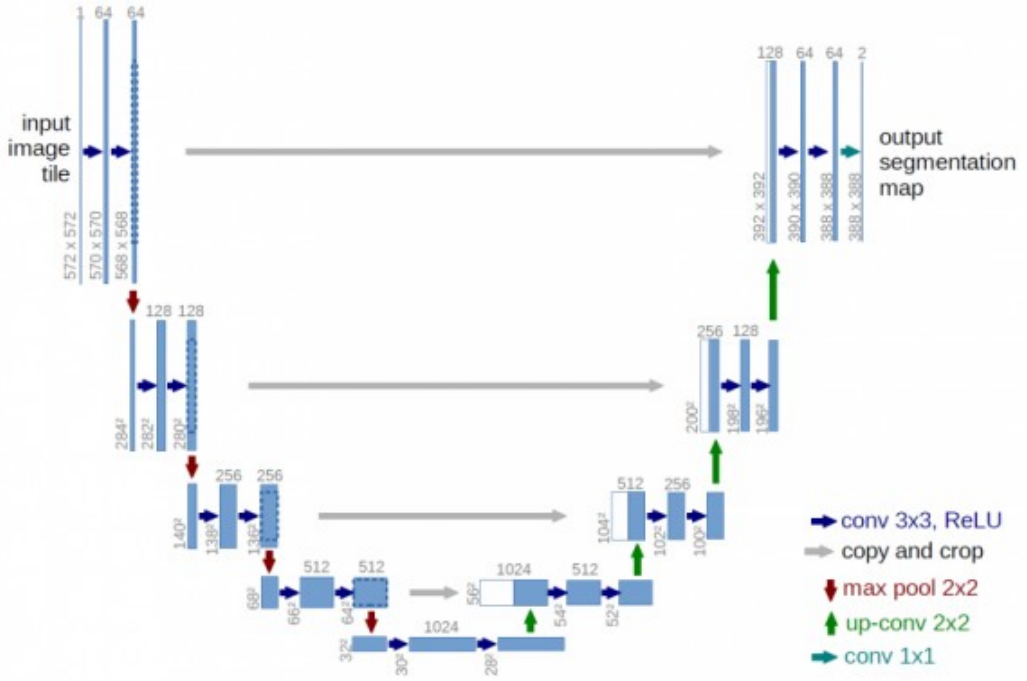

Рисунок 9 — U-Net архитектура

Изначально было решено использовать следующую архитектуру:

```
input_layer = Input(shape=x_train.shape[1:])
c1 = Conv2D(filters=8, kernel_size=(3, 3), activation='relu', padding='same')(input_layer)
l = MaxPool2D(strides=(2, 2))(c1)
c2 = Conv2D(filters=16, kernel_size=(3, 3), activation='relu', padding='same')(l)
l = MaxPool2D(strides=(2, 2))(c2)
c3 = Conv2D(filters=32, kernel_size=(3, 3), activation='relu', padding='same')(l)
l = MaxPool2D(strides=(2, 2))(c3)
c4 = Conv2D(filters=32, kernel_size=(1, 1), activation='relu', padding='same')(l)
l = concatenate([UpSampling2D(size=(2, 2))(c4), c3], axis=-1)
l = Conv2D(filters=32, kernel_size=(2, 2), activation='relu', padding='same')(l)
l = concatenate([UpSampling2D(size=(2, 2))(l), c2], axis=-1)
l = Conv2D(filters=24, kernel_size=(2, 2), activation='relu', padding='same')(l)
l = concatenate([UpSampling2D(size=(2, 2))(l), c1], axis=-1)
l = Conv2D(filters=16, kernel_size=(2, 2), activation='relu', padding='same')(l)
l = Conv2D(filters=64, kernel_size=(1, 1), activation='relu')(l)
output_layer = Conv2D(filters=1, kernel_size=(1, 1), activation='sigmoid')(l)
```

```
model = Model(input_layer, output_layer)
```

Рисунок 10 — Архитектура 1

Она включает в себя три операции максимального объединения на сужающемся пути, каждой из которых предшествует одна свёртка с матрицей 3x3.

На расширяющимся пути трём операциям максимального объединения соответствуют три операции повышающей дискретизации, за которыми следуют свёртка и объединение с соответствующим образом обрезанной картой свойств из сужающегося пути.

Просмотрим графики метрики Жаккара и потерь:

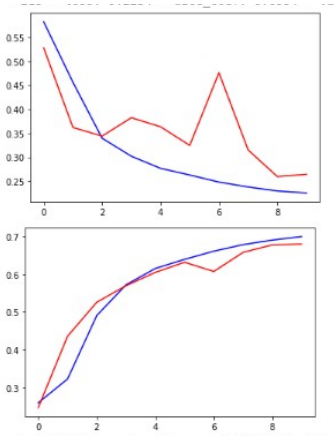

Рисунок 11 — график потерь и меры Жаккара (1).

Наблюдаем скачок потерь, попробуем встроить слой dropout, а так же обучать 30 эпох.

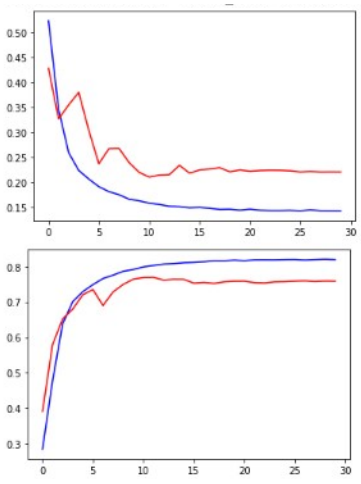

Рисунок 12 — график потерь и меры Жаккара (2).

Скачок чуть сгладился, в целом показатели меры Жаккара увеличились, что же касается продолжительности обучения — 10 эпох достаточно, далее наблюдается переобучение.

Попробуем увеличить точность работы, встроив дополнительные свёртки перед каждой операцией объединения, расширения. Получим следующие результаты:

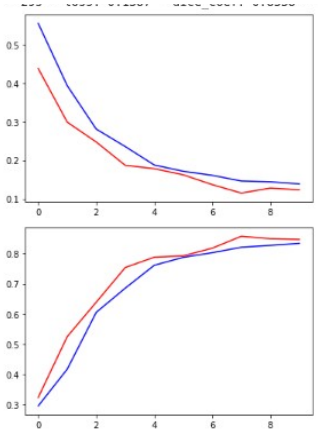

Рисунок 13 — график потерь и меры Жаккара (3).

Наблюдаем существенный прирост точности.

Попробуем теперь добавить ещё одну операцию объединения, соответствующую ей операцию расширения и дополнительное объединение карт свойств, получим следующую архитектуру:

```
input_layer = Input(shape=x_train.shape[1:])
c1 = Conv2D(filters=8, kernel_size=(3, 3), activation='relu', padding='same')(input_layer)
c1 = Conv2D(filters=8, kernel_size=(3, 3), activation='relu', padding='same')(c1)
l = MaxPool2D(strides=(2, 2))(c1)
c2 = Conv2D(filters=16, kernel_size=(3, 3), activation='relu', padding='same')(l)
c2 = Conv2D(filters=16, kernel_size=(3, 3), activation='relu', padding='same')(c2)
l = MaxPool2D(strides=(2, 2))(c2)
c3 = Conv2D(filters=32, kernel_size=(3, 3), activation='relu', padding='same')(l)
c3 = Conv2D(filters=32, kernel_size=(3, 3), activation='relu', padding='same')(c3)
l = MaxPool2D(strides=(2, 2))(c3)
c4 = Conv2D(filters=32, kernel_size=(1, 1), activation='relu', padding='same')(l)
c4 = Conv2D(filters=32, kernel_size=(1, 1), activation='relu', padding='same')(c4)
l = MaxPool2D(strides=(2, 2))(c4)
c5 = Conv2D(filters=32, kernel_size=(1, 1), activation='relu', padding='same')(l)
c5 = Conv2D(filters=32, kernel_size=(1, 1), activation='relu', padding='same')(c5)
l = concatenate([UpSampling2D(size=(2, 2))(c5), c4], axis=-1)
l = Conv2D(filters=32, kernel_size=(2, 2), activation='relu', padding='same')(l)
l = Conv2D(filters=32, kernel_size=(2, 2), activation='relu', padding='same')(l)
l = concatenate([UpSampling2D(size=(2, 2))(l), c3], axis=-1)
l = Conv2D(filters=24, kernel_size=(2, 2), activation='relu', padding='same')(l)
l = Conv2D(filters=24, kernel_size=(2, 2), activation='relu', padding='same')(l)
l = concatenate([UpSampling2D(size=(2, 2))(l), c2], axis=-1)
l = Conv2D(filters=24, kernel_size=(2, 2), activation='relu', padding='same')(l)
l = Conv2D(filters=24, kernel_size=(2, 2), activation='relu', padding='same')(l)
l = concatenate([UpSampling2D(size=(2, 2))(l), c1], axis=-1)
l = Conv2D(filters=16, kernel_size=(2, 2), activation='relu', padding='same')(l)
l = Conv2D(filters=16, kernel_size=(2, 2), activation='relu', padding='same')(l)
l = Conv2D(filters=64, kernel_size=(1, 1), activation='relu')(l)
l = Dropout(0.5)(l)
output_layer = Conv2D(filters=1, kernel_size=(1, 1), activation='sigmoid')(l)
model = Model(input_layer, output_layer)
```

Рисунок 14 — Конечная архитектура.

Получим следующие графики работы:

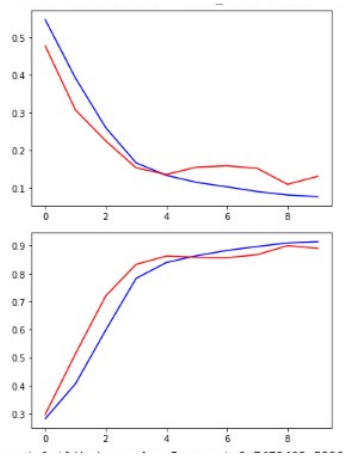

Рисунок 15 — график потерь и меры Жаккара (4).

Данная архитектура, хоть и является довольно громоздкой, опять же, даёт существенный прирост точности (+0.05), выходя на значения меры Жаккара около 0.9 при обучении в течение 9 эпох (10 оказалась так же избыточной).

4. Использование колбэков.

В работе был использован callback Keras ModelCheckpoint наиболее удачной конфигурации сети:

```
weight_saver = ModelCheckpoint('lung.h5', monitor='val_dice_coef',
                               save_best_only=True, save_weights_only=True)
```

Рисунок 16 — Колбэк weight_saver

5. Разделение ролей

Чигалейчик А.С. - поиск, обработка и аугментация данных.

Токарев А.П. - мера Жаккара и архитектура сети.

6. Вывод

В ходе данной работы была разработана нейронная сеть, решающая задачу сегментации изображений КТ лёгких, проведена подготовка и аугментация данных, найдена оптимальная архитектура сети по типу u-net, написана собственная метрика (метрика Жаккара), использован колбэк для сохранения лучшей конфигурации сети.