МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

Федеральное государственное бюджетное образовательное

учреждение высшего образования

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»



Факультет прикладной математики и информатики

Кафедра ТПИ

Дисциплина: «Основы теории машинного обучения»

Лабораторная работа №5, модуль «Нейронные сети глубокого обучения»

**СЕГМЕНТАЦИЯ ИЗОБРАЖЕНИЙ. СЕТЬ U-NET**

Факультет: ФПМИ

Группа: ПМИМ-31

Выполнили: Монгуш Н. С., Тарулин М. А., Филипенко Ю. Д.

Проверил: Попов А. А.

Дата выполнения:

Отметка о защите:

1. ЦЕЛЬ РАБОТЫ

Изучить архитектуру нейронной сети U-net, рассмотреть её применение в задаче сегментации изображения.

1. ЗАДАНИЕ
2. Реализовать архитектуру нейронной сети U-net, заменив предобученную модель MobileNetV2 и блок pix2pix на написанные самостоятельно блоки кода. Визуализировать полученную архитектуру.
3. Обучить полученную нейронную сеть на датасете oxford\_iiit\_pet. Количество эпох, итераций и батчей выбирать в зависимости от вычислительных возможностей. Добавить в отчет график изменения значения функций ошибки в процессе обучения. Продемонстрировать результаты предсказания обученной сети.
4. Изменить значения гиперпараметров и сравнить результаты с теми, что были получены раннее.
5. ИСХОДНЫЕ ДАННЫЕ

В качестве исходных данных был выбран датасет oxford\_iiit\_pet, который состоит из изображений размером 128x128 пикселей и из маски сегментации этого изображения.



Рис. 1.1 – Изображения из датасета oxford\_iiit\_pet.

1. ПОЛУЧЕННЫЕ РЕЗУЛЬТАТЫ
   1. **Архитектура нейронной сети с 26 слоями**

Разработанная модель основана на предобученной сверточной нейросети VGG16, из которой извлекаются признаки на трех уровнях: после третьего, четвертого и пятого блоков свертки (обозначены как f3, f4, f5). На самом глубоком уровне (f5), карты признаков обрабатываются через серию сверточных слоев и слоев дропаута для уменьшения переобучения, причём последний сверточный слой уменьшает количество каналов до желаемого количества выходных каналов. Эти признаки затем увеличиваются до размера карт признаков четвертого блока (f4) с помощью транспонированной свертки.

Слияние увеличенных признаков f5 с признаками f4 и последующее увеличение результата до размера карт признаков третьего блока (f3) позволяют модели интегрировать информацию с разных уровней. Финальная транспонированная свертка затем увеличивает размер комбинированных признаков до исходных размеров входного изображения, создавая выходные данные модели. Архитектура разработанной нейронной сети изображена на рис. 2.1.

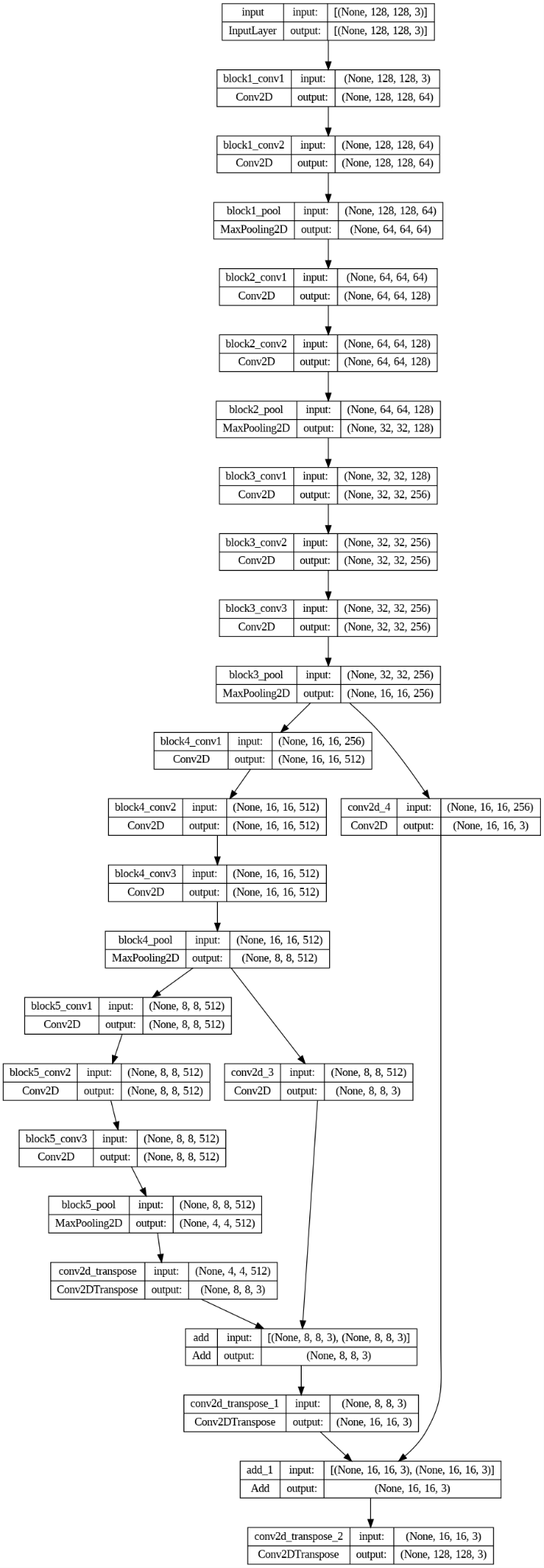


Рис. 2.1 – Архитектура разработанной нейронной сети U-net.

Произведем обучение разработанной нейронной сети на представленном выше датасете, изобразим результаты прогнозирования нейронной сети на рис. 3.1-3.5.

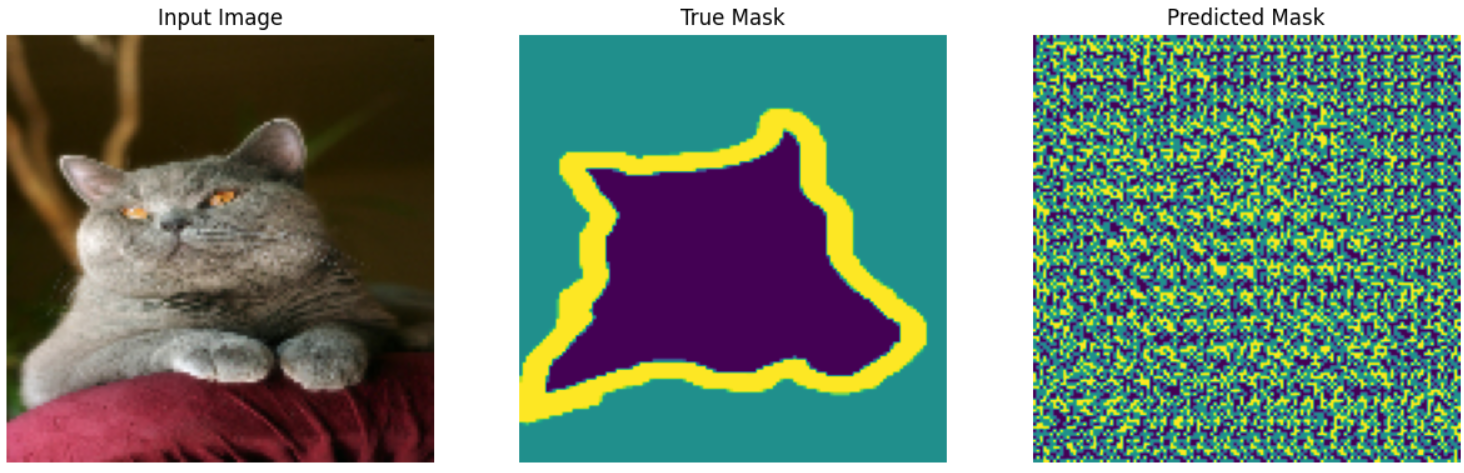


Рис. 3.1 – Результат прогнозирования I эпохи.

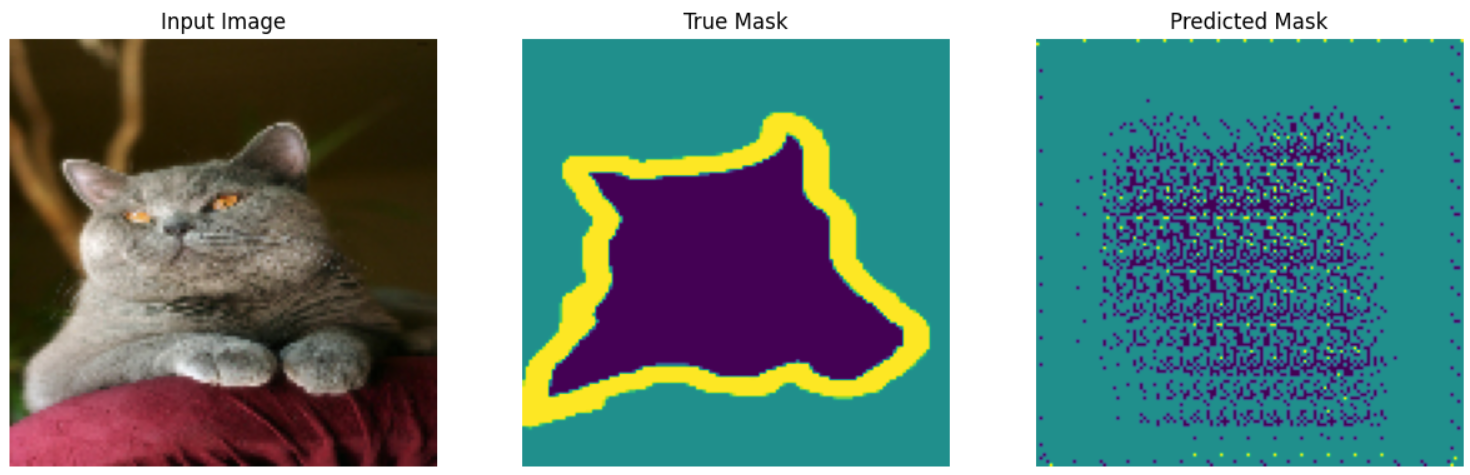


Рис. 3.2 – Результат прогнозирования II эпохи.

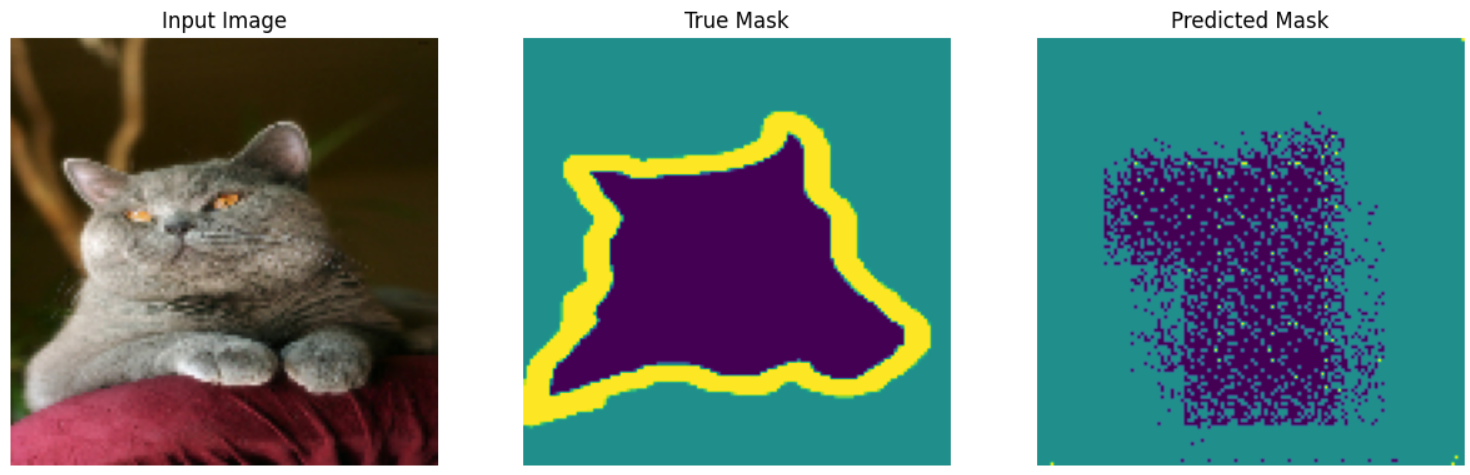


Рис. 3.3 – Результат прогнозирования III эпохи.

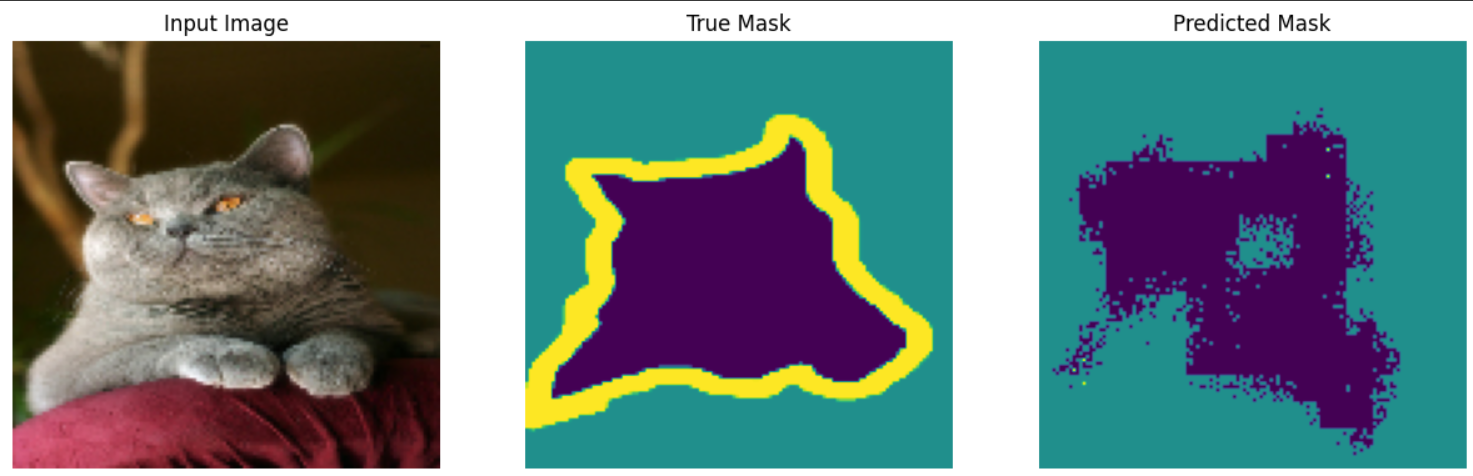


Рис. 3.4 – Результат прогнозирования IV эпохи.

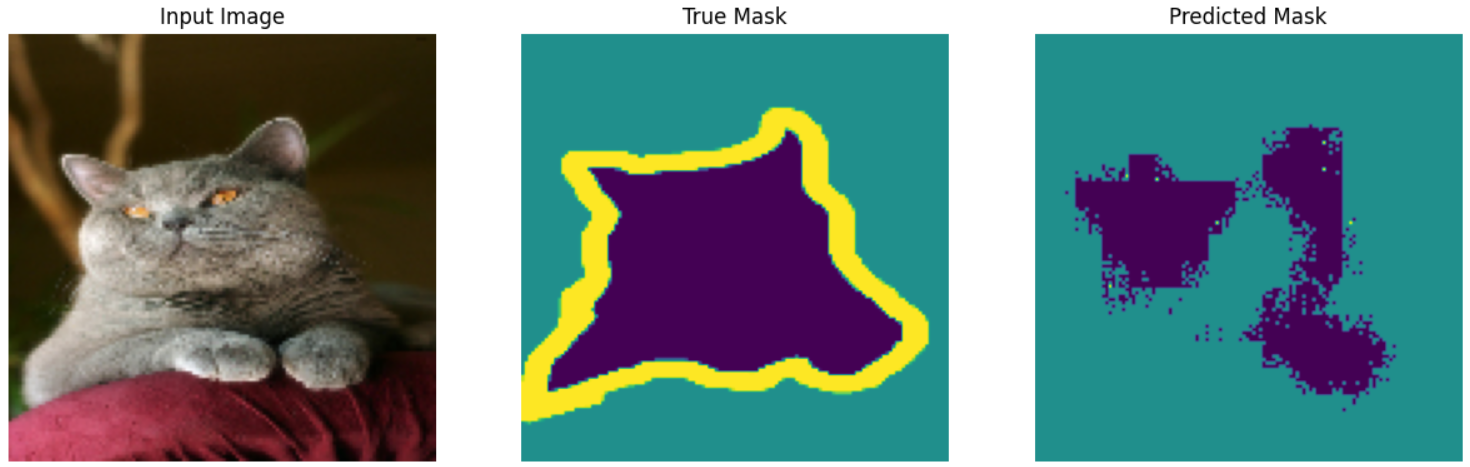


Рис. 3.5 – Результат прогнозирования V эпохи.

На рис. 4.1 изобразим график ошибок обучения (синяя линия) и ошибок валидации (красная линия) в зависимости от количества эпох обучения. Время обучения на 5 эпохах составило 3 ч. 35 мин.

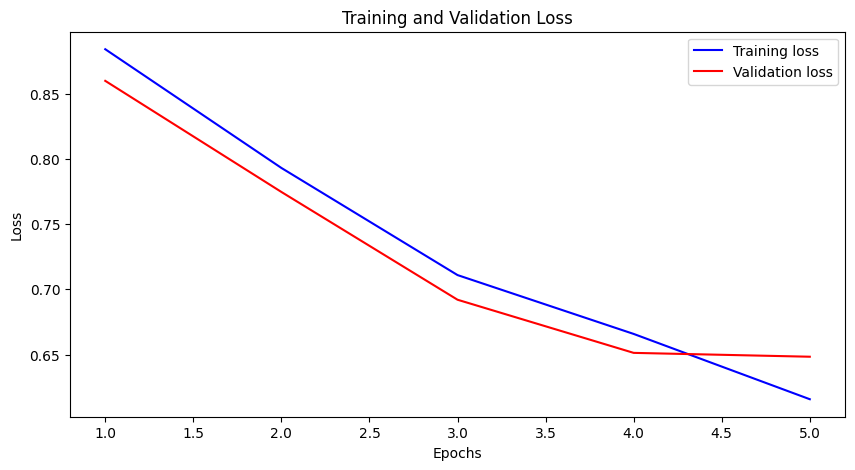


Рис. 4.1 – График ошибок на тренировочной и обучающей выборке.

* 1. **Архитектура нейронной сети с 18 слоями**

Разработанная архитектура, включает в себя кодировщик и декодировщик, соединённые сквозными связями для передачи пространственной информации и улучшения точности сегментации. Кодировщик построен из сверточных слоёв и слоёв максимального пулинга, которые уменьшают размер карт признаков и увеличивают их глубину, начиная с 64 фильтров на первом уровне и заканчивая 256 фильтрами на дне U-образной структуры.

Декодировщик использует транспонированные сверточные слои для увеличения размера карт признаков и сквозные соединения с картами кодировщика для сохранения деталей, каждый уровень содержит два сверточных слоя для обработки признаков. Финальный сверточный слой с функцией активации sigmoid создаёт карту вероятностей, классифицируя каждый пиксель.

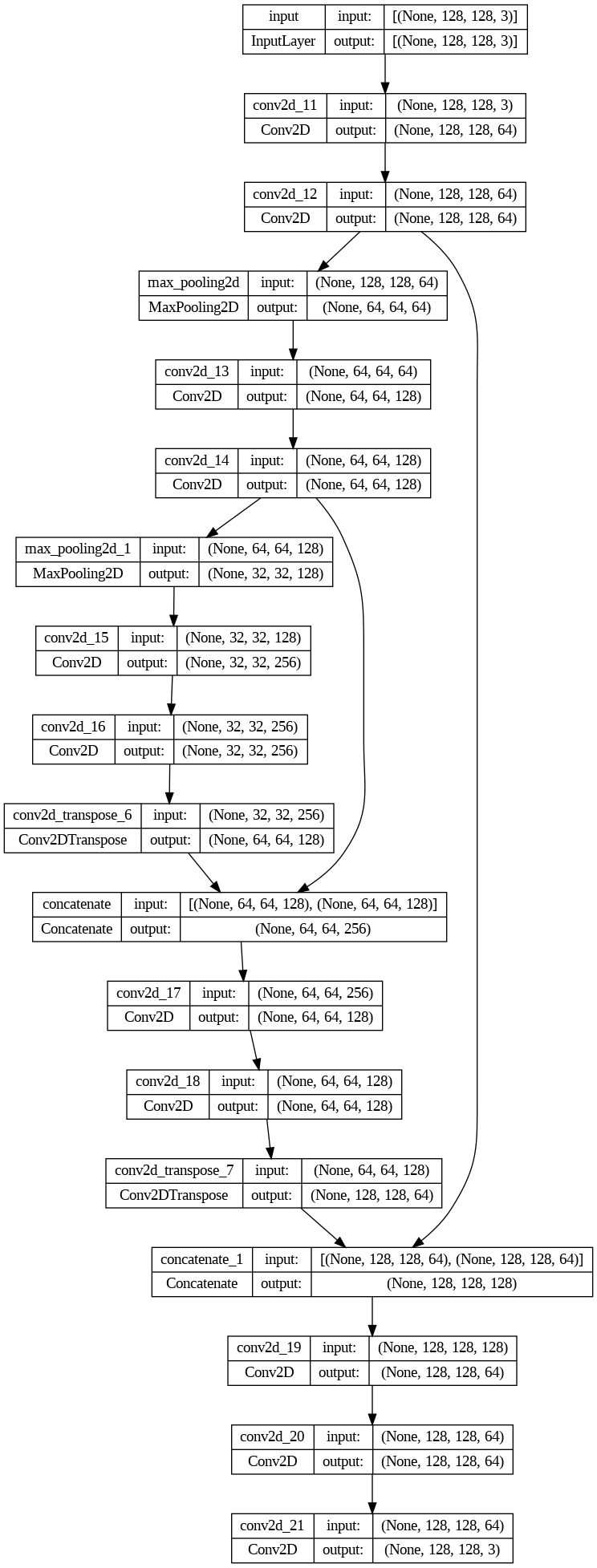


Рис. 5.1 – Архитектура разработанной нейронной сети U-net.

Произведем обучение разработанной нейронной сети, изобразим результаты прогнозирования нейронной сети на рис. 6.1-6.5.

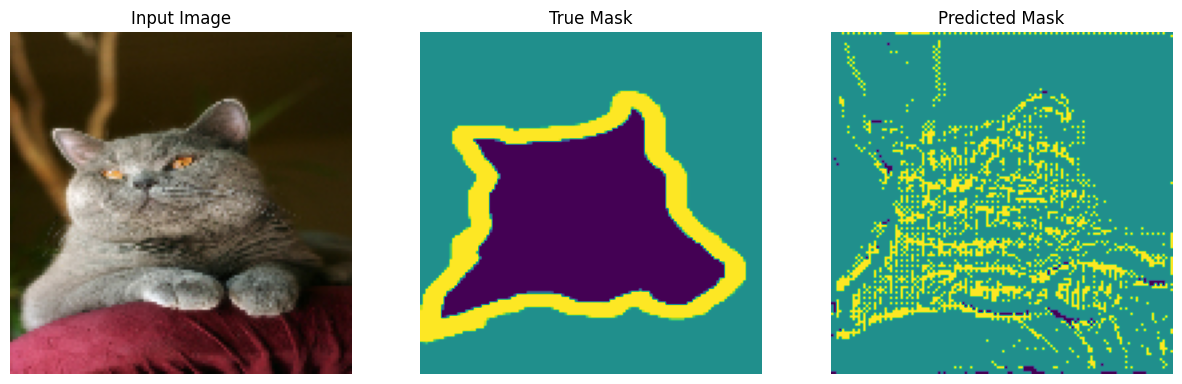


Рис. 6.1 – Результат прогнозирования первой эпохи.

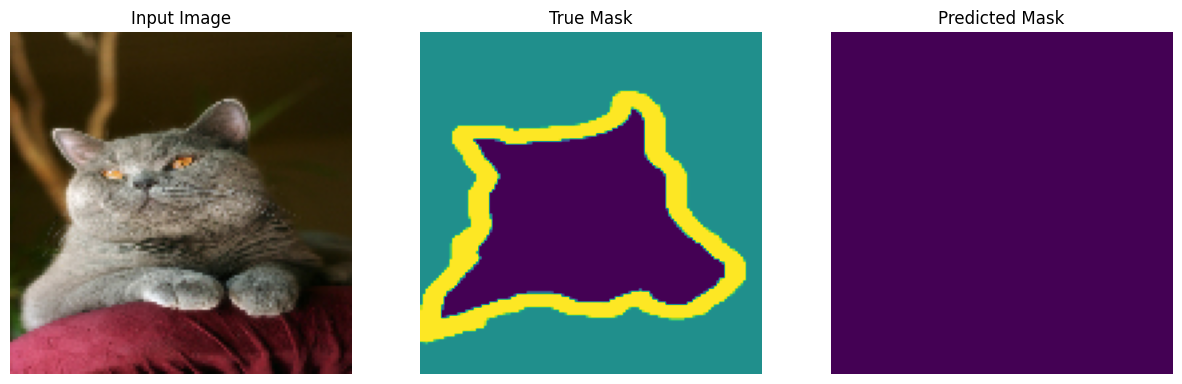


Рис. 6.2 – Результат прогнозирования II эпохи.

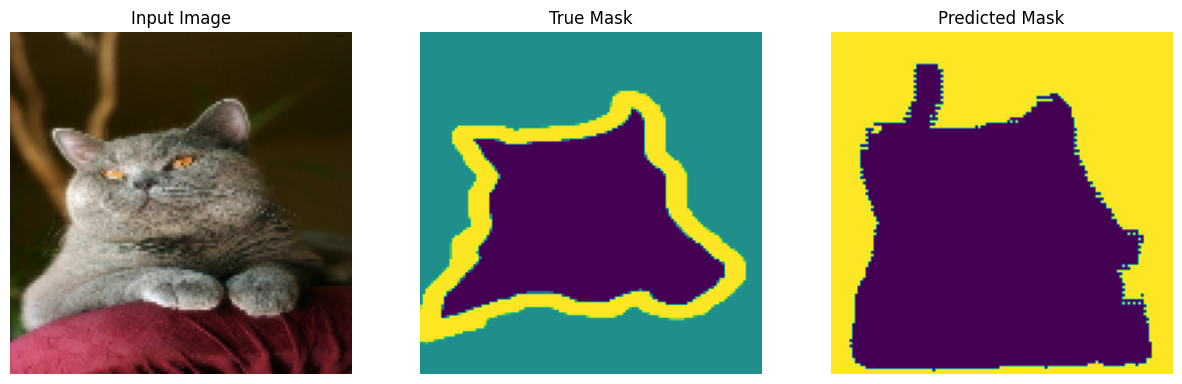


Рис. 6.3 – Результат прогнозирования III эпохи.

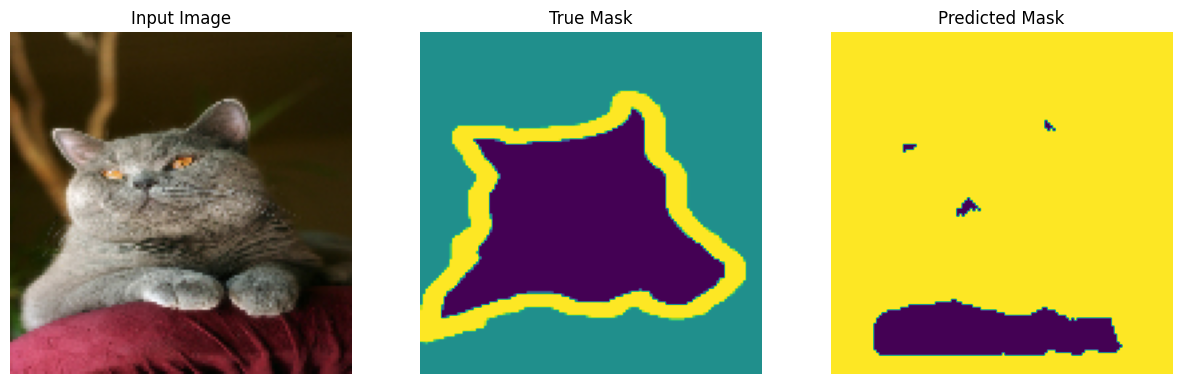


Рис. 6.4 – Результат прогнозирования IV эпохи.

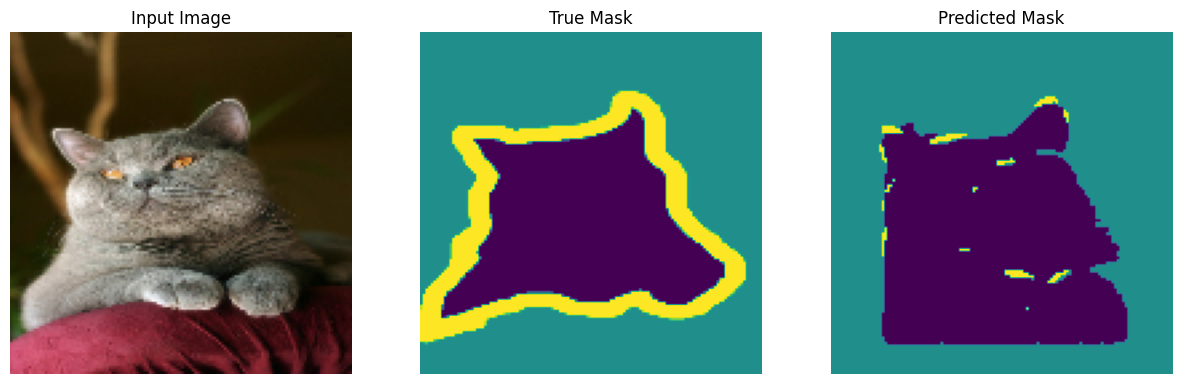


Рис. 6.5 – Результат прогнозирования V эпохи.

1. ВЫВОД

В ходе работы были сравнены результаты предсказания 2 архитектур нейронной сети с различными слоями. Вторая архитектура нейронной сети, с меньшим количеством слоев, по результатам 5 эпох обучения показала результаты предсказания лучше, чем архитектура с 23 слоями.

Архитектура с 18 слоями имеет прямой подход с явно определенными шагами кодирования и декодирования, используя свертки и транспонированные свертки. Архитектура с 23 слоями использует предварительно обученную сеть для извлечения признаков и их объединения на разных уровнях, чтобы создать сегментированные карты.

1. КОД ПРОГРАММЫ

pip install git+https://github.com/tensorflow/examples.git

import tensorflow as tf

import tensorflow\_datasets as tfds

tfds.disable\_progress\_bar()

from IPython.display import clear\_output

import matplotlib.pyplot as plt

dataset, info = tfds.load('oxford\_iiit\_pet:3.\*.\*', with\_info=True)

def load\_image\_train(datapoint):

  input\_image = tf.image.resize(datapoint['image'], (128, 128))

  input\_mask = tf.image.resize(datapoint['segmentation\_mask'], (128, 128))

  if tf.random.uniform(()) > 0.5:

    input\_image = tf.image.flip\_left\_right(input\_image)

    input\_mask = tf.image.flip\_left\_right(input\_mask)

  input\_image, input\_mask = normalize(input\_image, input\_mask)

  return input\_image, input\_mask

def load\_image\_test(datapoint):

  input\_image = tf.image.resize(datapoint['image'], (128, 128))

  input\_mask = tf.image.resize(datapoint['segmentation\_mask'], (128, 128))

  input\_image, input\_mask = normalize(input\_image, input\_mask)

  return input\_image, input\_mask

TRAIN\_LENGTH = info.splits['train'].num\_examples

BUFFER\_SIZE = 1000

BATCH\_SIZE = 128

STEPS\_PER\_EPOCH = TRAIN\_LENGTH // BATCH\_SIZE

train = dataset['train'].map(load\_image\_train, num\_parallel\_calls=tf.data.experimental.AUTOTUNE)

test = dataset['test'].map(load\_image\_test)

train\_dataset = train.cache().shuffle(BUFFER\_SIZE).batch(BATCH\_SIZE).repeat()

train\_dataset = train\_dataset.prefetch(buffer\_size=tf.data.experimental.AUTOTUNE)

test\_dataset = test.batch(BATCH\_SIZE)

def display(display\_list):

  plt.figure(figsize=(15, 15))

  title = ['Input Image', 'True Mask', 'Predicted Mask']

  for i in range(len(display\_list)):

    plt.subplot(1, len(display\_list), i+1)

    plt.title(title[i])

    plt.imshow(tf.keras.preprocessing.image.array\_to\_img(display\_list[i]))

    plt.axis('off')

  plt.show()

for image, mask in train.take(3):

  sample\_image, sample\_mask = image, mask

display([sample\_image, sample\_mask])

from tensorflow.keras.models import Model

from tensorflow.keras.layers import \*

from tensorflow.keras.applications.vgg16 import \*

def unet\_model(image\_size, ch\_in=3, ch\_out=3):

    inputs = Input(shape=(\*image\_size, ch\_in), name='input')

    # Сжимающий путь (Кодировщик)

    conv1 = Conv2D(64, 3, activation='relu', padding='same')(inputs)

    conv1 = Conv2D(64, 3, activation='relu', padding='same')(conv1)

    pool1 = MaxPooling2D(pool\_size=(2, 2))(conv1)

    conv2 = Conv2D(128, 3, activation='relu', padding='same')(pool1)

    conv2 = Conv2D(128, 3, activation='relu', padding='same')(conv2)

    pool2 = MaxPooling2D(pool\_size=(2, 2))(conv2)

    # Дно U-образной архитектуры

    conv3 = Conv2D(256, 3, activation='relu', padding='same')(pool2)

    conv3 = Conv2D(256, 3, activation='relu', padding='same')(conv3)

    # Расширяющий путь (Декодировщик)

    up1 = Conv2DTranspose(128, 3, strides=(2, 2), padding='same')(conv3)

    merge1 = concatenate([conv2, up1])

    conv4 = Conv2D(128, 3, activation='relu', padding='same')(merge1)

    conv4 = Conv2D(128, 3, activation='relu', padding='same')(conv4)

    up2 = Conv2DTranspose(64, 3, strides=(2, 2), padding='same')(conv4)

    merge2 = concatenate([conv1, up2])

    conv5 = Conv2D(64, 3, activation='relu', padding='same')(merge2)

    conv5 = Conv2D(64, 3, activation='relu', padding='same')(conv5)

    # Финальный слой для карт сегментации

    outputs = Conv2D(ch\_out, 1, activation='sigmoid', padding='same')(conv5)

    # Создание модели

    model = Model(inputs=inputs, outputs=outputs)

    return model

strategy = tf.distribute.MirroredStrategy()

with strategy.scope():

  model = unet\_model(image\_size=(128, 128))

  # compiling the model with the optimizer, loss function and acc metrics

  model.compile(optimizer='adam',

              loss=tf.keras.losses.SparseCategoricalCrossentropy(from\_logits=True),

              metrics=['accuracy'])

tf.keras.utils.plot\_model(model, show\_shapes=True)

def create\_mask(pred\_mask):

  pred\_mask = tf.argmax(pred\_mask, axis=-1)

  pred\_mask = pred\_mask[..., tf.newaxis]

  return pred\_mask[0]

def show\_predictions(dataset=None, num=1):

  if dataset:

    for image, mask in dataset.take(num):

      pred\_mask = model.predict(image)

      display([image[0], mask[0], create\_mask(pred\_mask)])

  else:

    display([sample\_image, sample\_mask,

             create\_mask(model.predict(sample\_image[tf.newaxis, ...]))])

show\_predictions()

class DisplayCallback(tf.keras.callbacks.Callback):

  def on\_epoch\_end(self, epoch, logs=None):

    clear\_output(wait=True)

    show\_predictions()

    print ('\nSample Prediction after epoch {}\n'.format(epoch+1))

 # ReduceLROnPlateau callback

reduce\_lr = tf.keras.callbacks.ReduceLROnPlateau(monitor='val\_loss', factor=0.2,

                              patience=5, min\_lr=0.001)

# EarlyStopping callback

early\_stopping = tf.keras.callbacks.EarlyStopping(monitor='loss', patience=8)

EPOCHS = 5

VAL\_SUBSPLITS = 5

VALIDATION\_STEPS = info.splits['test'].num\_examples//BATCH\_SIZE//VAL\_SUBSPLITS

model\_history = model.fit(train\_dataset, epochs=EPOCHS,

                          steps\_per\_epoch=STEPS\_PER\_EPOCH,

                          validation\_steps=VALIDATION\_STEPS,

                          validation\_data=test\_dataset,

                          callbacks=[DisplayCallback(), reduce\_lr, early\_stopping])

# Получаем историю обучения

train\_loss = model\_history.history['loss']

val\_loss = model\_history.history['val\_loss']

epochs = range(1, len(train\_loss) + 1)

# Строим графики

plt.plot(epochs, train\_loss, 'bo', label='Training loss')

plt.plot(epochs, val\_loss, 'r', label='Validation loss')

plt.title('Training and validation loss')

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.legend()

plt.show()