

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ
ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования
**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ТОМСКИЙ
ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**



Центр цифровых
образовательных технологий

09.04.01 Информатика и вычислительная техника

Семантическая сегментация изображений кошек и собак с помощью сверточного
автоэнкодера

ЛАБОРАТОРНАЯ РАБОТА № 7

по дисциплине:
Машинное обучение

Исполнитель:

студент группы

8ВМ42

Текере Ричард

Руководитель:

доцент

ОИТ, ИШИТР

Друки А.А.

ВВЕДЕНИЕ

Семантическая сегментация – это задача компьютерного зрения, при которой каждому пикселью изображения присваивается определённый класс. Иными словами, происходит выделение объектов на изображении на уровне пикселей. В контексте данной работы целью сегментации является отделить животных (кошек и собак) от фона на фотографиях. Результатом сегментации является маска – изображение-разметка, где каждый пиксель помечен классом «животное» или «фон» (а также особым классом для границы объекта). Такой подход позволяет точно очертить контуры объектов (например, формы кошки или собаки) на исходном изображении.

Для решения задачи сегментации применяются специальные нейросетевые архитектуры. Одной из самых популярных является сеть U-Net, которая представляет собой сверточный энкодер-декодер (autoencoder) с дополнительными пропусками (skip-связями) между соответствующими слоями энкодера и декодера. В данной работе, однако, используется более простой подход – сверточный автоэнкодер без skip-связей. Сверточный автоэнкодер состоит из двух частей: энкодера, последовательно сжимающего изображение в компактное представление (выделяя ключевые признаки), и декодера, который развертывает сжатое представление обратно в изображение исходного размера. В результате такой модель способна принимать на вход изображение и выдавать на выходе карту сегментации – для каждого пикселя предсказывается класс (фон, животное или граница животного).

В данной лабораторной работе используется датасет Oxford-IIIT Pet, содержащий фотографии домашних кошек и собак различных пород с ручной разметкой на уровне пикселей (так называемый trimap-аннотации). Каждое изображение датасета сопровождается маской сегментации, в которой каждому пикселью присвоена одна из трёх категорий:

1. пиксель принадлежит животному (кошке/собаке),

2. пиксель находится на границе контура животного,
3. пиксель принадлежит фону.

Цель работы

Изучить методы семантической сегментации изображений и получить навыки разработки сверточной нейросети-автоэнкодера для сегментации объектов (кошек и собак) на изображениях.

Ход работы

1. Загрузка и подготовка данных MNIST

Первым шагом мы загружаем набор данных MNIST и выполняем его предварительную обработку. Данные представляют собой изображения цифр размером 28×28 , которые удобно нормализовать в диапазон $[0,1]$ для эффективного обучения нейронной сети. Кроме того, для использования сверточных слоёв необходимо добавить измерение канала (так как изображения grayscale имеют один канал). Ниже приведен код загрузки данных, нормализации значений пикселей и преобразования размеров массивов:

```
import urllib.request, tarfile

# Ссылки на архивы изображений и аннотаций Oxford-IIIT Pet
url_images = "https://www.robots.ox.ac.uk/~vgg/data/pets/images.tar.gz"
url_masks = "https://www.robots.ox.ac.uk/~vgg/data/pets/annotations.tar.gz"

# Загрузка архивов
urllib.request.urlretrieve(url_images, "images.tar.gz")
urllib.request.urlretrieve(url_masks, "annotations.tar.gz")

# Распаковка содержимого архивов
tarfile.open("images.tar.gz").extractall()      # извлекает папку images/ с
фотографиями
tarfile.open("annotations.tar.gz").extractall() # извлекает папку annotations/ с
масками
```

Вывод:

Sample Image: great_pyrenees_21.jpg



Рисунок 1 — Пример загруженного изображения из датасета.

2. Предобработка данных

На этапе предобработки данные были подготовлены к подаче в нейросеть. Этот этап включает: формирование списков путей к файлам, чтение изображений и масок в память, преобразование их размеров и масштабов, а также разбиение на обучающую и тестовую выборки. Ниже описаны основные шаги предобработки. В данной работе выбрано разрешение 128×128 пикселей для ускорения обучения и снижения требуемой памяти. Каждое исходное фото и соответствующая маска были считаны и изменены до этого размера. Для чтения изображений использовалась библиотека PIL. Пример кода:

```
from PIL import Image  
import numpy as np  
import os
```

```

# Папки с изображениями и масками после распаковки
images_dir = "images/"
masks_dir = "annotations/trimaps/"

X_data = [] # список для изображений
y_data = [] # список для масок

# Проход по всем файлам изображений в директории
for filename in os.listdir(images_dir):
    if filename.endswith(".jpg"):
        # Полные пути к файлу изображения и соответствующей маске
        img_path = os.path.join(images_dir, filename)
        mask_path = os.path.join(masks_dir, filename.split('.')[0] + ".png")

        # Открытие изображения и маски
        img = Image.open(img_path)
        mask = Image.open(mask_path)
        # Преобразование размера до 128x128
        img = img.resize((128, 128))
        mask = mask.resize((128, 128))
        # Преобразование в массивы numpy
        img_array = np.array(img)
        mask_array = np.array(mask)

        X_data.append(img_array)
        y_data.append(mask_array)

```

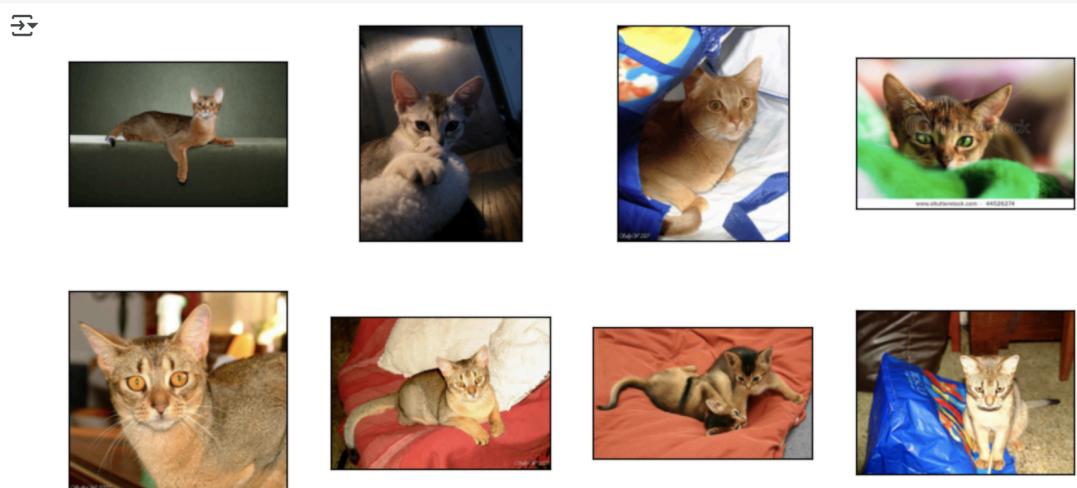


Рисунок 2 — Вывод 8 первых изображений из набора данных.

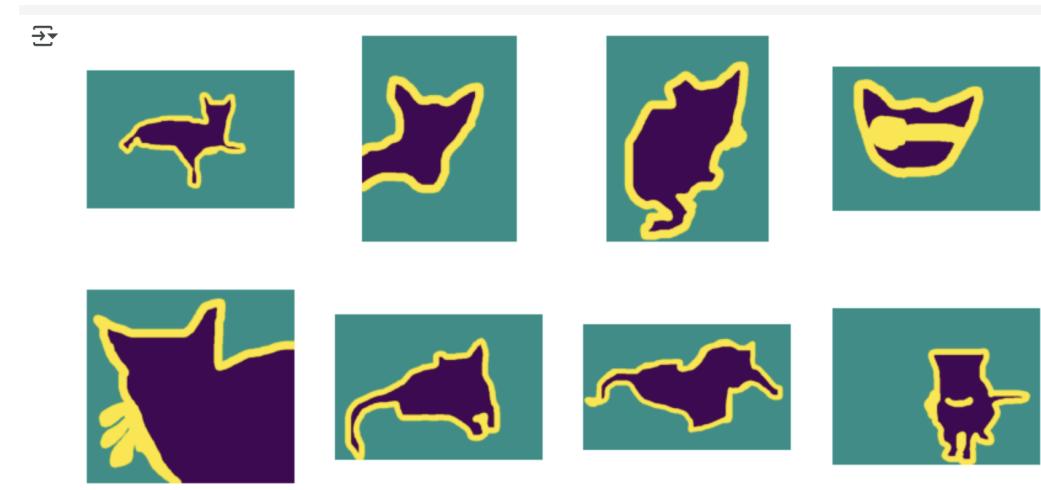


Рисунок 3 — Вывод 8 соответствующих масок сегментации.

3. Построение модели (энкодер и декодер автоэнкодера)

Для решения задачи сегментации разработана модель сверточного автоэнкодера, состоящая из энкодера (несколько слоёв свёртки и субдискретизации) и декодера (слои *upsampling* или транспонированной свёртки для восстановления размера изображения). Модель создана с использованием функционального API Keras – это позволяет явно указать, какие слои идут за какими, и получить на выходе объект модели .

```
def get_model(img_size, num_classes):
    inputs = keras.Input(shape=img_size + (3,))

    # Encoder
    x = layers.Conv2D(32, 3, activation="relu", padding="same")(inputs)
    x = layers.MaxPooling2D()(x)
    x = layers.Conv2D(64, 3, activation="relu", padding="same")(x)
    x = layers.MaxPooling2D()(x)
    x = layers.Conv2D(128, 3, activation="relu", padding="same")(x)
    x = layers.MaxPooling2D()(x)

    # Decoder
    x = layers.Conv2DTranspose(128, 3, strides=2, activation="relu",
                             padding="same")(x)
    x = layers.Conv2DTranspose(64, 3, strides=2, activation="relu",
                             padding="same")(x)
    x = layers.Conv2DTranspose(32, 3, strides=2, activation="relu",
                             padding="same")(x)
```

```

outputs = layers.Conv2D(num_classes, 3, activation="softmax",
padding="same")(x)
model = keras.Model(inputs, outputs)
return model

model = get_model(img_size=img_size, num_classes=3)
model.summary()

```

Model: "functional"

Layer (type)	Output Shape	Param #
input_layer_2 (InputLayer)	(None, 200, 200, 3)	0
conv2d (Conv2D)	(None, 200, 200, 32)	896
max_pooling2d (MaxPooling2D)	(None, 100, 100, 32)	0
conv2d_1 (Conv2D)	(None, 100, 100, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 50, 50, 64)	0
conv2d_2 (Conv2D)	(None, 50, 50, 128)	73,856
max_pooling2d_2 (MaxPooling2D)	(None, 25, 25, 128)	0
conv2d_transpose (Conv2DTranspose)	(None, 50, 50, 128)	147,584
conv2d_transpose_1 (Conv2DTranspose)	(None, 100, 100, 64)	73,792
conv2d_transpose_2 (Conv2DTranspose)	(None, 200, 200, 32)	18,464
conv2d_3 (Conv2D)	(None, 200, 200, 3)	867

Total params: 333,955 (1.27 MB)
Trainable params: 333,955 (1.27 MB)
Non-trainable params: 0 (0.00 B)

Рисунок 4 — Сводка модели автокодировщика, показывающая слои энкодера и декодера.

4. Компиляция и обучение модели

Перед началом обучения модель необходимо скомпилировать, указав функцию потерь (лосс-функцию) и оптимизатор. В этой работе используются:

4.1 Функция потерь: sparse_categorical_crossentropy – кроссэнтропия для случая не one-hot кодированных меток. Эта функция сравнивает для каждого пикселя предсказанное распределение по 3 классам с истинным

классом (0, 1 или 2) и вычисляет вклад ошибки. crossentropy подходит для задач многоклассовой классификации, к которым относится и сегментация (каждый пиксель – объект классификации). Мы используем sparse_ вариант, поскольку наши метки хранятся как целые числа классов, а не в виде one-hot векторов.

4.2 Оптимизатор: RMSprop – популярный алгоритм градиентного спуска с адаптивной скоростью обучения. Он хорошо зарекомендовал себя в задачах сегментации и генеративных моделях. В компиляции использован оптимизатор RMSprop с параметрами по умолчанию (начальная скорость обучения 0.001).

Компиляция модели и обучение на обучающей выборке:

```
model.compile(optimizer="rmsprop", loss="sparse_categorical_crossentropy")

history = model.fit(
    input_imgs[:-1000], targets[:-1000],
    epochs=10,
    batch_size=64,
    validation_data=(input_imgs[-1000:], targets[-1000:])
)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.legend()
plt.title("Training and Validation Loss")
plt.show()
```

Вывод:

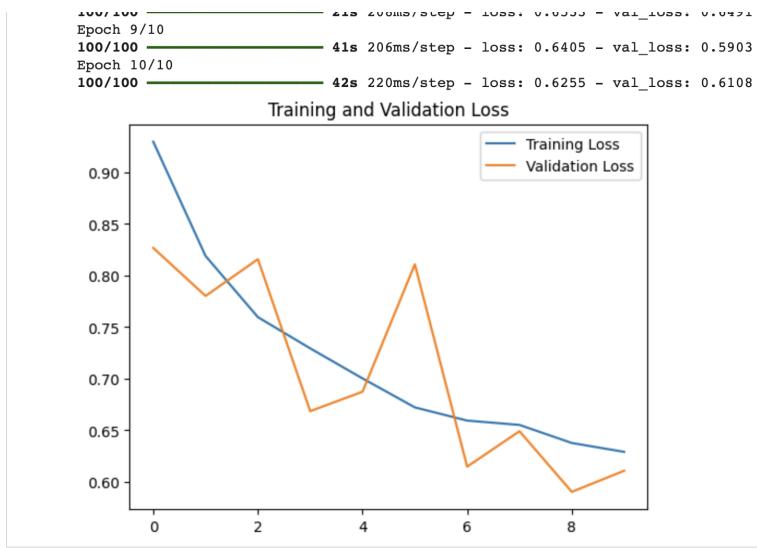


Рисунок 5 — Компиляция и обучение модели.

5. Проверка сегментации

Проведение предсказания на изображении из валидационной выборки:

```
i = 20

test_image = input_imgs[-1000:][i]
true_mask = targets[-1000:][i]
predicted_mask = model.predict(np.expand_dims(test_image, 0))[0]

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

plt.subplot(1,3,1)
plt.title("Input Image")
plt.imshow(test_image.astype("uint8"))
plt.axis("off")

plt.subplot(1,3,2)
plt.title("True Mask")
plt.imshow(true_mask.squeeze(), cmap='gray')
plt.axis("off")

plt.subplot(1,3,3)
plt.title("Predicted Mask")
plt.imshow(np.argmax(predicted_mask, axis=-1), cmap='gray')
plt.axis("off")

plt.show()
```

Вывод:

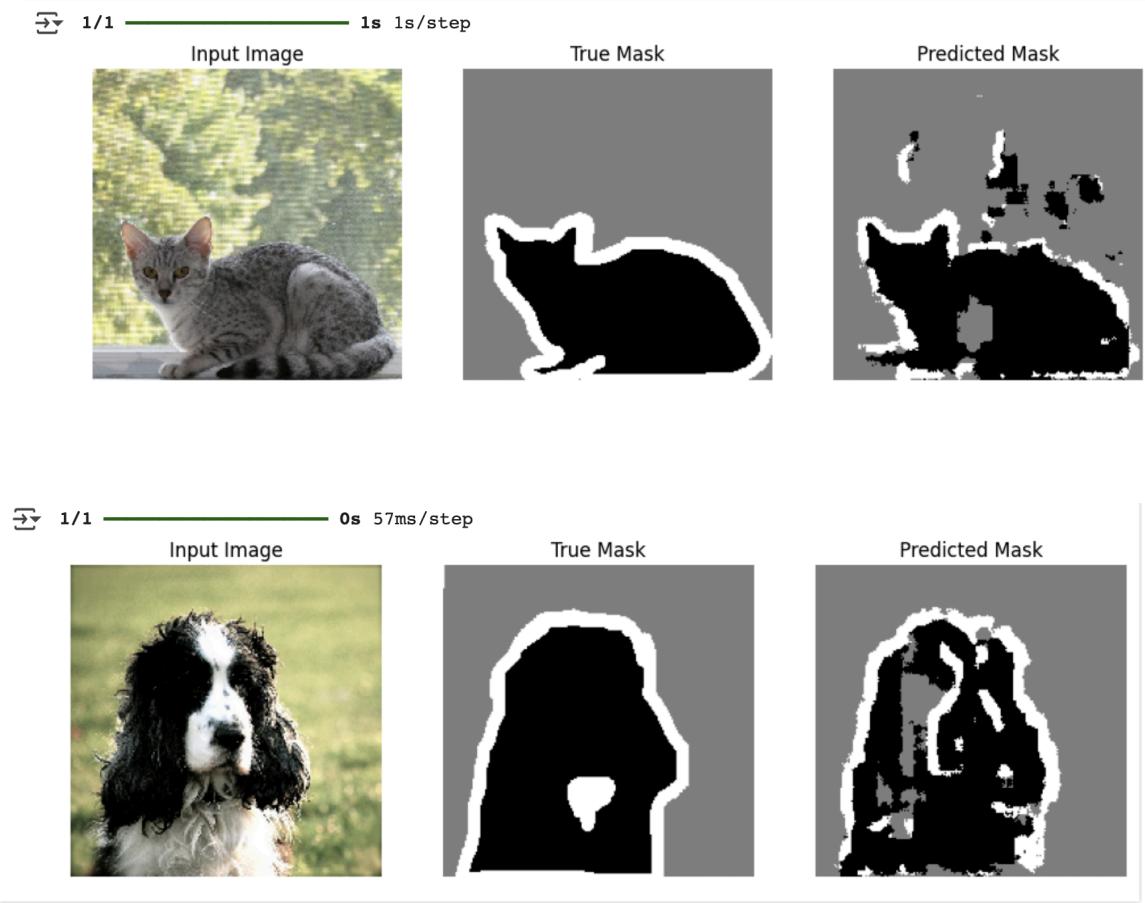


Рисунок 6 — исходное изображение, целевая маска и результат предсказания.

Заключение

В ходе выполнения лабораторной работы студент ознакомился с задачей семантической сегментации изображений и практическими методами её решения с помощью нейронных сетей. Был изучен процесс подготовки специальных данных для сегментации – загрузка реального датасета (Oxford-IIIT Pet), преобразование и нормализация изображений и пиксельных масок. Студент приобрёл навык построения нейросетевых моделей через функциональное API Keras, научился создавать и объединять слой входа, сверточные, pooling и транспонированные сверточные слои в единую архитектуру энкодер-декодер.

Был успешно реализован и обучен сверточный автоэнкодер, решающий задачу сегментации объектов (кошек и собак на фотографиях). В процессе обучения были получены навыки настройки параметров обучения (функции потерь *sparse_categorical_crossentropy* для многоклассовой сегментации, оптимизатор RMSprop и др.), мониторинга метрик (анализ кривых потерь и точности) и выявления возможного переобучения.

Анализ результатов сегментации на тестовых данных позволил оценить качество работы модели: студент научился интерпретировать полученные маски, сравнивать их с эталоном, замечать характерные ошибки (например, неточности по контуру) и понимать причины их возникновения с точки зрения устройства сети.

Приложение А

[Google Colab Link](#)

```
# Импорты
import os
import random
import tarfile
import matplotlib.pyplot as plt
import numpy as np
from tensorflow.keras.utils import load_img, img_to_array
from tensorflow import keras
from tensorflow.keras import layers

# Монтирование Google Drive
from google.colab import drive
drive.mount('/content/drive')

# Распаковка архивов
images_tar_path = '/content/drive/MyDrive/Pets/images.tar.gz'
annotations_tar_path = '/content/drive/MyDrive/Pets/annotations.tar.gz'

with tarfile.open(images_tar_path) as tar:
    tar.extractall(path='/content')

with tarfile.open(annotations_tar_path) as tar:
    tar.extractall(path='/content')

# Проверка: вывод одного случайного изображения
sample_image_path = os.path.join("/content/images",
random.choice(os.listdir("/content/images")))
img = load_img(sample_image_path)

plt.figure(figsize=(5,5))
plt.imshow(img)
plt.title(f'Sample Image: {os.path.basename(sample_image_path)}')
plt.axis('off')
plt.show()

# Предобработка данных
input_dir = "images/"
target_dir = "annotations/trimaps/"

input_img_paths = sorted([
    os.path.join(input_dir, fname)
    for fname in os.listdir(input_dir)
    if fname.endswith(".jpg")
])

target_paths = sorted([
    os.path.join(target_dir, fname)
])
```

```

for fname in os.listdir(target_dir)
    if fname.endswith(".png") and not fname.startswith(".")
[])
# Вывод примеров изображений
plt.figure(figsize=(10,10))
for i in range(8):
    plt.subplot(4,4,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.imshow(load_img(input_img_paths[i]))
    plt.show()

# Вывод примеров масок
def display_target(target_array):
    normalized_array = (target_array.astype("uint8") - 1) * 127
    plt.axis("off")
    plt.imshow(normalized_array[:, :, 0])

plt.figure(figsize=(10,10))
for i in range(8):
    plt.subplot(4,4,i+1)
    plt.xticks([])
    plt.yticks([])
    img = img_to_array(load_img(target_paths[i], color_mode="grayscale"))
    display_target(img)
    plt.show()

# Загрузка изображений и масок в массивы
img_size = (200, 200)
num_imgs = len(input_img_paths)

random.Random(1337).shuffle(input_img_paths)
random.Random(1337).shuffle(target_paths)

def path_to_input_image(path):
    return img_to_array(load_img(path, target_size=img_size))

def path_to_target(path):
    img = img_to_array(load_img(path, target_size=img_size, color_mode="grayscale"))
    img = img.astype("uint8") - 1
    return img

input_imgs = np.zeros((num_imgs,) + img_size + (3,), dtype="float32")
targets = np.zeros((num_imgs,) + img_size + (1,), dtype="uint8")

for i in range(num_imgs):
    input_imgs[i] = path_to_input_image(input_img_paths[i])
    targets[i] = path_to_target(target_paths[i])

# Вывод одного примера

```

```

plt.figure(figsize=(6,3))
plt.subplot(1,2,1)
plt.imshow(input_imgs[0].astype("uint8"))
plt.title("Input Image")
plt.axis("off")

plt.subplot(1,2,2)
plt.imshow(targets[0].squeeze(), cmap="gray")
plt.title("Target Mask")
plt.axis("off")
plt.show()

# Построение модели автоэнкодера
def get_model(img_size, num_classes):
    inputs = keras.Input(shape=img_size + (3,))

    # Encoder
    x = layers.Conv2D(32, 3, activation="relu", padding="same")(inputs)
    x = layers.MaxPooling2D()(x)
    x = layers.Conv2D(64, 3, activation="relu", padding="same")(x)
    x = layers.MaxPooling2D()(x)
    x = layers.Conv2D(128, 3, activation="relu", padding="same")(x)
    x = layers.MaxPooling2D()(x)

    # Decoder
    x = layers.Conv2DTranspose(128, 3, strides=2, activation="relu", padding="same")(x)
    x = layers.Conv2DTranspose(64, 3, strides=2, activation="relu", padding="same")(x)
    x = layers.Conv2DTranspose(32, 3, strides=2, activation="relu", padding="same")(x)

    outputs = layers.Conv2D(num_classes, 3, activation="softmax", padding="same")(x)
    model = keras.Model(inputs, outputs)
    return model

model = get_model(img_size=img_size, num_classes=3)
model.summary()

# Компиляция и обучение модели
model.compile(optimizer="rmsprop", loss="sparse_categorical_crossentropy")

history = model.fit(
    input_imgs[:-1000], targets[:-1000],
    epochs=10,
    batch_size=64,
    validation_data=(input_imgs[-1000:], targets[-1000:]))
)

# График ошибки
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.legend()
plt.title("Training and Validation Loss")

```

```
plt.show()

# Проверка сегментации
i = 20

test_image = input_imgs[-1000:][i]
true_mask = targets[-1000:][i]
predicted_mask = model.predict(np.expand_dims(test_image, 0))[0]

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

plt.subplot(1,3,1)
plt.title("Input Image")
plt.imshow(test_image.astype("uint8"))
plt.axis("off")

plt.subplot(1,3,2)
plt.title("True Mask")
plt.imshow(true_mask.squeeze(), cmap='gray')
plt.axis("off")

plt.subplot(1,3,3)
plt.title("Predicted Mask")
plt.imshow(np.argmax(predicted_mask, axis=-1), cmap='gray')
plt.axis("off")
plt.show()
```