cell_segmentation-alexandre_bailly

January 9, 2025

1 Context

Recent studies have shown that breast cancer continues to be the leading cause of death among women over the world. If detected at an early stage, it can be cured in 9 out of 10 cases.

Automated detection and segmentation of cells from images are the crucial and fundamental steps for the measurement of cellular morphology that is crucial for brest cancer diagnosis and prognosis.

In this notebook, you will learn how to train a segmentation as UNet with **monai** - a framwork based Pytorch Stadard for healthcare imaging.

1.1 Monai

MONAI is a pytorch based open source AI framework launched by NVIDIA and King's College London. It is integrated with training and modelling workflows in a native PyTorch Standard. t several places.

Install monai

[1]:
```
!pip install monai
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: monai in
/home/alexandre/.local/lib/python3.10/site-packages (1.4.0)
Requirement already satisfied: numpy<2.0,>=1.24 in
/home/alexandre/.local/lib/python3.10/site-packages (from monai) (1.24.4)
Requirement already satisfied: torch>=1.9 in
/home/alexandre/.local/lib/python3.10/site-packages (from monai) (2.4.0)
Requirement already satisfied: fsspec in
/home/alexandre/.local/lib/python3.10/site-packages (from torch>=1.9->monai)
(2024.3.1)
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.1.105 in
/home/alexandre/.local/lib/python3.10/site-packages (from torch>=1.9->monai)
(12.1.105)
Requirement already satisfied: networkx in
/home/alexandre/.local/lib/python3.10/site-packages (from torch>=1.9->monai)
(3.3)
Requirement already satisfied: nvidia-curand-cu12==10.3.2.106 in
/home/alexandre/.local/lib/python3.10/site-packages (from torch>=1.9->monai)
(10.3.2.106)
```

```
Requirement already satisfied: sympy in
/home/alexandre/.local/lib/python3.10/site-packages (from torch>=1.9->monai)
(1.13.2)
Requirement already satisfied: triton==3.0.0 in
/home/alexandre/.local/lib/python3.10/site-packages (from torch>=1.9->monai)
(3.0.0)
Requirement already satisfied: nvidia-nccl-cu12==2.20.5 in
/home/alexandre/.local/lib/python3.10/site-packages (from torch>=1.9->monai)
(2.20.5)
Requirement already satisfied: typing-extensions>=4.8.0 in
/home/alexandre/.local/lib/python3.10/site-packages (from torch>=1.9->monai)
(4.12.0)
Requirement already satisfied: nvidia-cudnn-cu12==9.1.0.70 in
/home/alexandre/.local/lib/python3.10/site-packages (from torch>=1.9->monai)
(9.1.0.70)
Requirement already satisfied: nvidia-cufft-cu12==11.0.2.54 in
/home/alexandre/.local/lib/python3.10/site-packages (from torch>=1.9->monai)
(11.0.2.54)
Requirement already satisfied: nvidia-nvtx-cu12==12.1.105 in
/home/alexandre/.local/lib/python3.10/site-packages (from torch>=1.9->monai)
(12.1.105)
Requirement already satisfied: jinja2 in
/home/alexandre/.local/lib/python3.10/site-packages (from torch>=1.9->monai)
(3.1.4)
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.1.105 in
/home/alexandre/.local/lib/python3.10/site-packages (from torch>=1.9->monai)
(12.1.105)
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.1.105 in
/home/alexandre/.local/lib/python3.10/site-packages (from torch>=1.9->monai)
(12.1.105)
Requirement already satisfied: nvidia-cublas-cu12==12.1.3.1 in
/home/alexandre/.local/lib/python3.10/site-packages (from torch>=1.9->monai)
(12.1.3.1)
Requirement already satisfied: filelock in
/home/alexandre/.local/lib/python3.10/site-packages (from torch>=1.9->monai)
(3.13.4)
Requirement already satisfied: nvidia-cusparse-cu12==12.1.0.106 in
/home/alexandre/.local/lib/python3.10/site-packages (from torch>=1.9->monai)
(12.1.0.106)
Requirement already satisfied: nvidia-cusolver-cu12==11.4.5.107 in
/home/alexandre/.local/lib/python3.10/site-packages (from torch>=1.9->monai)
(11.4.5.107)
Requirement already satisfied: nvidia-nvjitlink-cu12 in
/home/alexandre/.local/lib/python3.10/site-packages (from nvidia-cusolver-
cu12==11.4.5.107->torch>=1.9->monai) (12.6.68)
Requirement already satisfied: MarkupSafe>=2.0 in
/home/alexandre/.local/lib/python3.10/site-packages (from
jinja2->torch>=1.9->monai) (2.1.5)
```

```
Requirement already satisfied: mpmath<1.4,>=1.1.0 in
/home/alexandre/.local/lib/python3.10/site-packages (from
sympy->torch>=1.9->monai) (1.3.0)
```

Check the installation by running the following cell

[2]:
```python
import monai
monai.config.print_config()
```

```
2025-01-09 00:22:23.312379: E
external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:485] Unable to register
cuFFT factory: Attempting to register factory for plugin cuFFT when one has
already been registered
2025-01-09 00:22:23.399903: E
external/local_xla/xla/stream_executor/cuda/cuda_dnn.cc:8454] Unable to register
cuDNN factory: Attempting to register factory for plugin cuDNN when one has
already been registered
2025-01-09 00:22:23.426007: E
external/local_xla/xla/stream_executor/cuda/cuda_blas.cc:1452] Unable to
register cuBLAS factory: Attempting to register factory for plugin cuBLAS when
one has already been registered
2025-01-09 00:22:23.596656: I tensorflow/core/platform/cpu_feature_guard.cc:210]
This TensorFlow binary is optimized to use available CPU instructions in
performance-critical operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild
TensorFlow with the appropriate compiler flags.
2025-01-09 00:22:24.745066: W
tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning: Could not
find TensorRT
```

```
MONAI version: 1.4.0
Numpy version: 1.24.4
Pytorch version: 2.4.0+cu121
MONAI flags: HAS_EXT = False, USE_COMPILED = False, USE_META_DICT = False
MONAI rev id: 46a5272196a6c2590ca2589029eed8e4d56ff008
MONAI __file__: /home/<username>/.local/lib/python3.10/site-
packages/monai/__init__.py

Optional dependencies:
Pytorch Ignite version: NOT INSTALLED or UNKNOWN VERSION.
ITK version: NOT INSTALLED or UNKNOWN VERSION.
Nibabel version: NOT INSTALLED or UNKNOWN VERSION.
scikit-image version: 0.22.0
scipy version: 1.13.0
Pillow version: 9.5.0
Tensorboard version: 2.17.1
gdown version: NOT INSTALLED or UNKNOWN VERSION.
TorchVision version: NOT INSTALLED or UNKNOWN VERSION.
tqdm version: 4.66.4
```

```
lmdb version: NOT INSTALLED or UNKNOWN VERSION.
psutil version: 6.0.0
pandas version: 2.2.2
einops version: NOT INSTALLED or UNKNOWN VERSION.
transformers version: 4.40.0
mlflow version: 2.17.2
pynrrd version: NOT INSTALLED or UNKNOWN VERSION.
clearml version: NOT INSTALLED or UNKNOWN VERSION.

For details about installing the optional dependencies, please visit:
    https://docs.monai.io/en/latest/installation.html#installing-the-
recommended-dependencies
```

2 Dataset

To train a model, we need to prepare some ingredients:

1. Dataset
2. Model
3. Loss function
4. Optimizer

3 I. Create Dataset

There are two ways to create your dataset: - with pytorch Dataset - with monai.data.Dataset.

In this exercise, we will create our dataset using torch.utils.data.Dataset.

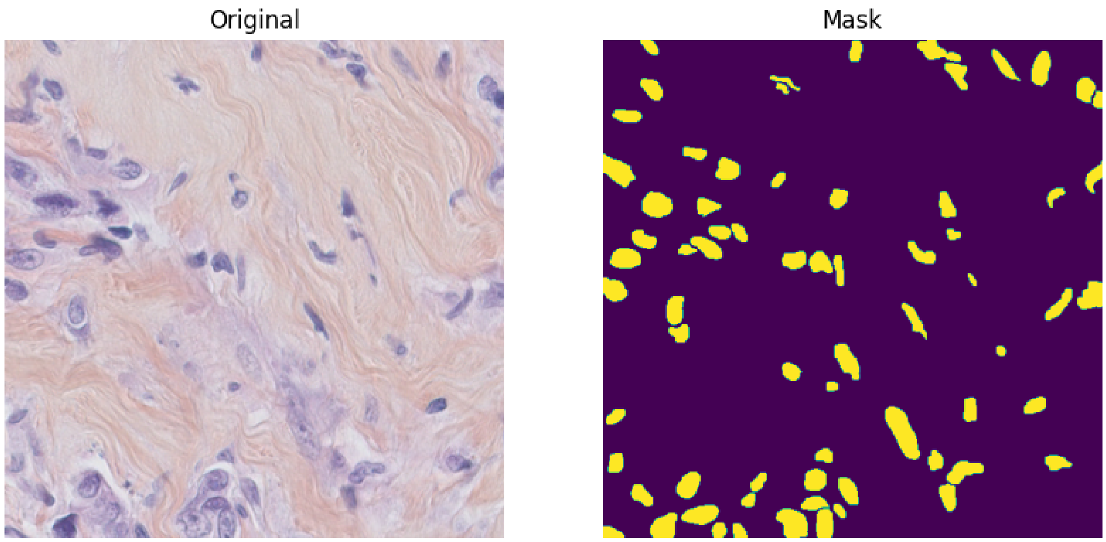
3.1 1. List all files in folder

Download the dataset from https://zenodo.org/record/1175282#.YMn_Qy-FDox

Notice that there are two kind of folder : original cell picture folder and mask folders. Using your file explorer or some code, display one image and the corresponding image

[3]:
```python
import matplotlib.pyplot as plt
from PIL import Image

image = 'TNBC_NucleiSegmentation/Slide_01/01_1.png'
mask = 'TNBC_NucleiSegmentation/GT_01/01_1.png'

plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.imshow(Image.open(image))
plt.title('Original')
plt.axis('off')

plt.subplot(1, 2, 2)
```

```
plt.imshow(Image.open(mask))
plt.title('Mask')
plt.axis('off')

plt.show()
```



3.2 2. Define a transform

When you load your data, you need to define some transformation. For example, we want to convert image to the format [num_channels, spatial_dim_1, spatial_dim_2] because monai/pytorch use this format. We'll also need to convert the images to PyTorch tensors with transforms.ToTensor()

The following code lets you load image and the labels and define several steps to transform the data.

[4]:
```
# CHANGED IMPORT AND GIVEN CODE DUE TO DEPRECATED FUNCTIONS
import torch
from monai.transforms import Compose, LoadImage, ToTensor,␣
 ↪NormalizeIntensity,EnsureChannelFirst
from monai.transforms import ScaleIntensity

from monai.data import PILReader

image_trans = Compose(
    [
        LoadImage(image_only=True, reader = PILReader(converter=lambda image:␣
 ↪image.convert("RGB"))),
        EnsureChannelFirst(),
        NormalizeIntensity(),
```

```
        ToTensor(),
    ])


label_trans = Compose(
    [
        LoadImage(image_only=True),
        EnsureChannelFirst(),
        ScaleIntensity(),
        ToTensor(),
    ])
```

3.3 3. Create dataset

The following class CellDataset allows us to create our dataset from "image_files" and "label_files" where: - "image_files" is a list of image names - "label_files" is the list of segmentation names respectively.

"im_trans" and "label_trans" are respectively the transforms for the images and their labels.

[5]:
```
import torch

class CellDataset(torch.utils.data.Dataset):
    def __init__(self, image_files, label_files, im_trans, label_trans):
        self.image_files = image_files
        self.label_files = label_files
        self.im_trans = im_trans
        self.label_trans = label_trans

    def __len__(self):
        return(len(self.image_files))

    def __getitem__(self, index):
        return self.im_trans(self.image_files[index]), self.label_trans(self.
 ↪label_files[index])
#Changed the ast line as well
```

By using this class, create your training dataset et your test dataset. Remember to check if your dataset is loaded correctly.

[8]:
```
import os
import glob

image_dirs = [dir_name for dir_name in sorted(os.
 ↪listdir('TNBC_NucleiSegmentation'))
              if dir_name.startswith('Slide')]
mask_dirs = [dir_name for dir_name in sorted(os.
 ↪listdir('TNBC_NucleiSegmentation'))
```

```python
            if dir_name.startswith('GT')]

image = []
for dir_name in image_dirs:
    image.extend(sorted(glob.glob(f'TNBC_NucleiSegmentation/{dir_name}/*.png')))

mask = []
for dir_name in mask_dirs:
    mask.extend(sorted(glob.glob(f'TNBC_NucleiSegmentation/{dir_name}/*.png')))

# If the prints are not the same then the data is incorrectly loaded
print("Nb image files : ", len(image))
print("Nb mask files : ", len(mask))

# If the prints are not the same then the data is incorrectly loaded
print("Nb image dirs : ", len(image_dirs))
print("Nb mask dirs : ", len(mask_dirs))


# Split
num_train = int(0.8 * len(image))
train_images, test_images = image[:num_train], image[num_train:]
train_labels, test_labels = mask[:num_train], mask[num_train:]

# Train
train_dataset = CellDataset(
    image_files=train_images,
    label_files=train_labels,
    im_trans=image_trans,
    label_trans=label_trans
)

# Test
test_dataset = CellDataset(
    image_files=test_images,
    label_files=test_labels,
    im_trans=image_trans,
    label_trans=label_trans
)

# We should have 50 and 50 for Nb image files and Nb mask files and 11 for the␣
↪dirs
```

```
Nb image files :  50
Nb mask files :  50
Nb image dirs :  11
Nb mask dirs :  11
```

3.4 4. DataLoader

With the your dataset loaded, you have to pass it to a DataLoader. The torch.utils.data.DataLoader takes a dataset and returns batches of images and the corresponding labels. You can set various parameters like the batch size and if the data is shuffled after each epoch.

The following code let you create a data loader for the train dataset, do the same to create a test_loader on the test_dataset. Name it **test_load**

[18]:
```
train_loader = torch.utils.data.DataLoader(
    train_dataset,
    batch_size=32,
    shuffle=True
)

test_load = torch.utils.data.DataLoader(
    test_dataset,
    batch_size=32,
    shuffle=True
)
```

3.5 5. Now, time to check your dataloader.

Execute the code following to check if your dataloader works correctly

[19]:
```
import monai
im, seg = monai.utils.misc.first(train_load)
im.shape
```

[19]:
```
torch.Size([32, 3, 512, 512])
```

4 II. Build your segmentation model with monai

Monai already has a UNet model architecture : https://docs.monai.io/en/stable/networks.html#unet

By using the monai.networks.nets module, build a UNet model for segmentation task in 2D. You'll have to choose the following parameters for the model:

1. dimensions (number of spatial dimensions)
2. in_channels (number of input channel)
3. out_channels (number of output channel)
4. channels
5. strides

[20]:
```
from monai.networks.nets import UNet
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

model = UNet(
    spatial_dims=2,
```

```
    in_channels=3,
    out_channels=1,
    channels=(32, 64, 128, 256),
    strides=(2, 2, 2, 2),
).to(device)

print(f"Use: {device}")
```

```
Use: cpu
```

```
/home/alexandre/.local/lib/python3.10/site-
packages/monai/networks/nets/unet.py:130: UserWarning: `len(strides) >
len(channels) - 1`, the last 1 values of strides will not be used.
  warnings.warn(f"`len(strides) > len(channels) - 1`, the last {delta} values of
strides will not be used.")
```

5 III. Define your loss function and optimizer

For a segmentation prob, we usually use DiceLoss. Using monai.losses.DiceLoss, define your loss function and store it in the variable named **loss_function**. The option sigmoid = True should be used.

[21]:
```
from monai.losses import DiceLoss

loss_function = DiceLoss(
    sigmoid=True
)
```

With torch.optim, define an optimizer for your model. Use the Adam optimiser

[24]:
```
from torch import optim

optimizer = optim.Adam(
    model.parameters(),
    lr=1e-4,
    eps=1e-8,
    weight_decay=1e-5
)
```

6 IV. Trainning the model

This time, we have all ingredients to train a segmentation model: a model, an optimizer, train_loader and a loss function.

Monai use a standard PyTorch program style for training a deep learning model.

The general process with Monai/Pytorch just for one learning step as follows:

1. Load input and label of each batch.

2. Zero accumulated gradients with optimizer.zero_grad()
3. Compute the output from the model
4. Calculate the loss
5. Perform backprop with loss.backward()
6. Update the optimizer with optimizer.step()

Complete the following code so that it do the training

[25]:
```
epoch_loss_values = list()
for epoch in range(2):

    model.train()
    epoch_loss = 0
    step = 0
    for batch_data in train_loader:
        step += 1
        inputs, labels = batch_data[0].to(device), batch_data[1].to(device)
        optimizer.zero_grad()

        #compute the model predictions using the model variable and inputs
        predictions = model(inputs)

        # compute the loss using the loss function, the predictions and labels
        loss = loss_function(predictions, labels)

        # use the backward method of the loss variable to  compute the gradient␣
 ↪of the loss used to find the minimum of the loss function

        # call the step method of the optimizer
        optimizer.step()

        epoch_loss += loss.item()
        epoch_len = len(train_dataset) // train_loader.batch_size
        print(f"{step}/{epoch_len}, train_loss: {loss.item():.4f}")


    epoch_loss /= step
    epoch_loss_values.append(epoch_loss)
    print(f"epoch {epoch + 1} average loss: {epoch_loss:.4f}")
```

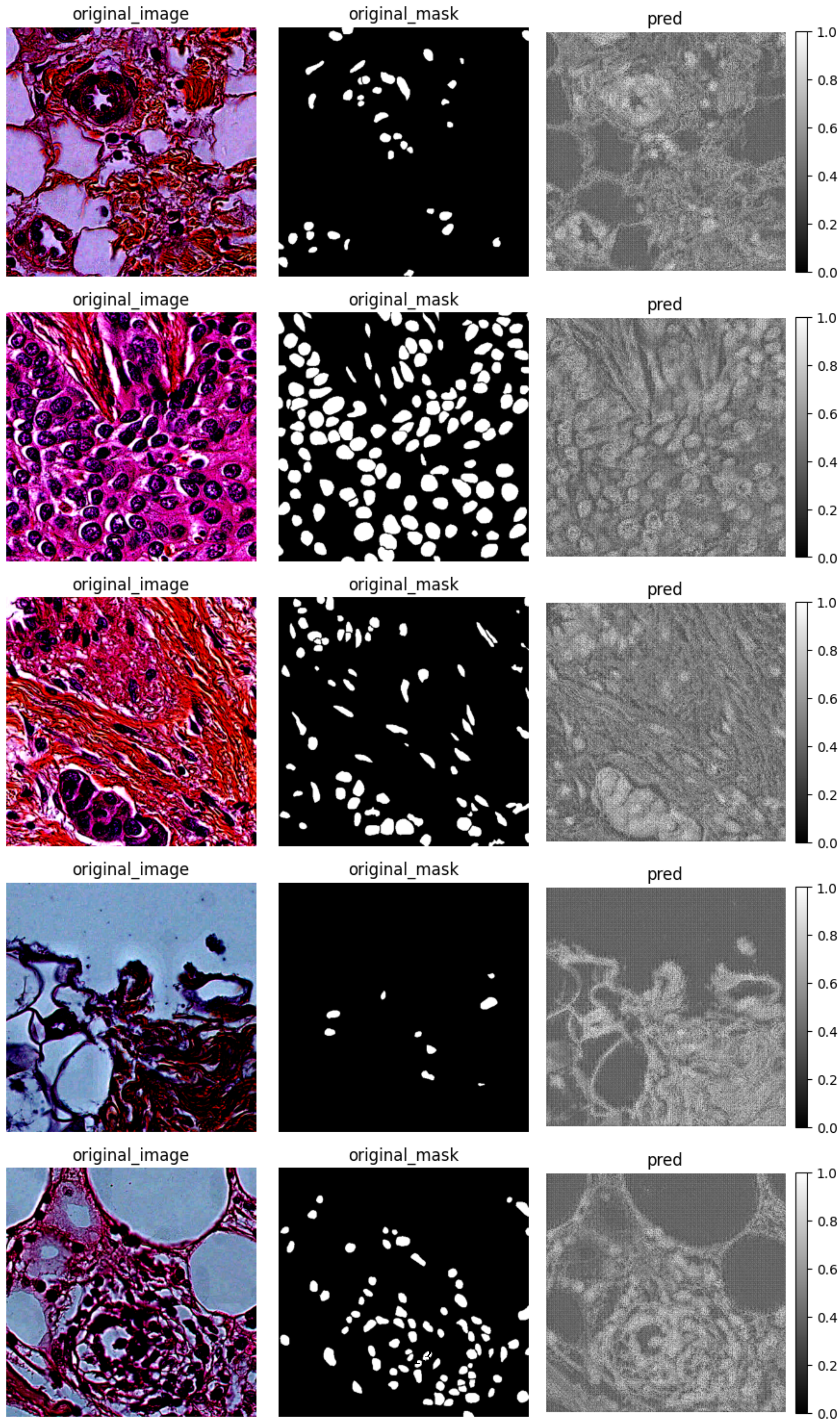
```
1/1, train_loss: 0.7710
2/1, train_loss: 0.7896
epoch 1 average loss: 0.7803
1/1, train_loss: 0.7674
2/1, train_loss: 0.8039
epoch 2 average loss: 0.7857
```

Display the prediction of your model on several image

[29]:
```python
import matplotlib.pyplot as plt
import torch

model.eval()

# Get a batch of images and masks from test_load
with torch.no_grad():
    batch_data = next(iter(test_load))
    images, masks = batch_data[0].to(device), batch_data[1].to(device)

    # Predict
    predictions = model(images)
    predictions = torch.sigmoid(predictions)

    # Move tensors to CPU
    images = images.cpu()
    masks = masks.cpu()
    predictions = predictions.cpu()

num_images = 5
fig, axes = plt.subplots(num_images, 3, figsize=(9, 3 * num_images))

for idx in range(num_images):
    # Display the original image
    axes[idx, 0].imshow(images[idx].permute(1, 2, 0))
    axes[idx, 0].set_title('original_image')
    axes[idx, 0].axis('off')

    # Display the original mask
    im1 = axes[idx, 1].imshow(masks[idx][0], cmap='gray')
    axes[idx, 1].set_title('original_mask')
    axes[idx, 1].axis('off')

    # Display the prediction
    im2 = axes[idx, 2].imshow(predictions[idx][0], cmap='gray', vmin=0, vmax=1)
    axes[idx, 2].set_title('pred')
    axes[idx, 2].axis('off')
    plt.colorbar(im2, ax=axes[idx, 2], fraction=0.046, pad=0.04)
plt.tight_layout()
plt.show()
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers). Got range [-8.309181..2.351774].
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers). Got range [-6.013909..2.5418687].
Clipping input data to the valid range for imshow with RGB data ([0..1] for
```

```
floats or [0..255] for integers). Got range [-6.8392878..2.5710113].
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers). Got range [-5.775525..1.7985137].
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers). Got range [-5.8697805..1.7427459].
```


original_image
original_mask
pred
1.0
0.8
0.6
0.4
0.2
0.0

Train another architecture (either another Unet architecture or find another segmentation model in the available models of Monai). Compare the results with the first model

[30]:
```
new_unet_model = UNet(
    spatial_dims=2,
    in_channels=3,
    out_channels=1,
    channels=(16, 32, 64, 128, 256),
    strides=(2, 2, 2, 2),
    num_res_units=2,
    act="PRELU",
    norm="INSTANCE",
    dropout=0.2,
    bias=True,
    adn_ordering="NDA",
).to(device)

epoch_loss_values = list()
for epoch in range(2):

    model.train()
    epoch_loss = 0
    step = 0
    for batch_data in train_loader:
        step += 1
        inputs, labels = batch_data[0].to(device), batch_data[1].to(device)
        optimizer.zero_grad()

        #compute the model predictions using the model variable and inputs
        predictions = model(inputs)

        # compute the loss using the loss function, the predictions and labels
        loss = loss_function(predictions, labels)

        # use the backward method of the loss variable to  compute the gradient␣
 ↪of the loss used to find the minimum of the loss function

        # call the step method of the optimizer
        optimizer.step()

        epoch_loss += loss.item()
        epoch_len = len(train_dataset) // train_loader.batch_size
        print(f"{step}/{epoch_len}, train_loss: {loss.item():.4f}")
```

```
    epoch_loss /= step
    epoch_loss_values.append(epoch_loss)
    print(f"epoch {epoch + 1} average loss: {epoch_loss:.4f}")
```

```
1/1, train_loss: 0.7767
2/1, train_loss: 0.7667
epoch 1 average loss: 0.7717
1/1, train_loss: 0.7848
2/1, train_loss: 0.7344
epoch 2 average loss: 0.7596
```

[ ]:
```
# I got slightly worse results. It could be because of too few epoch, because␣
↪of a too high dropout or because of the many other parameters I added.
```