2D Glioma classification

This notebook demonstrates classification of brain tumors with MONAI. To accelerate training, 2D dataset is used based on the 3D one.

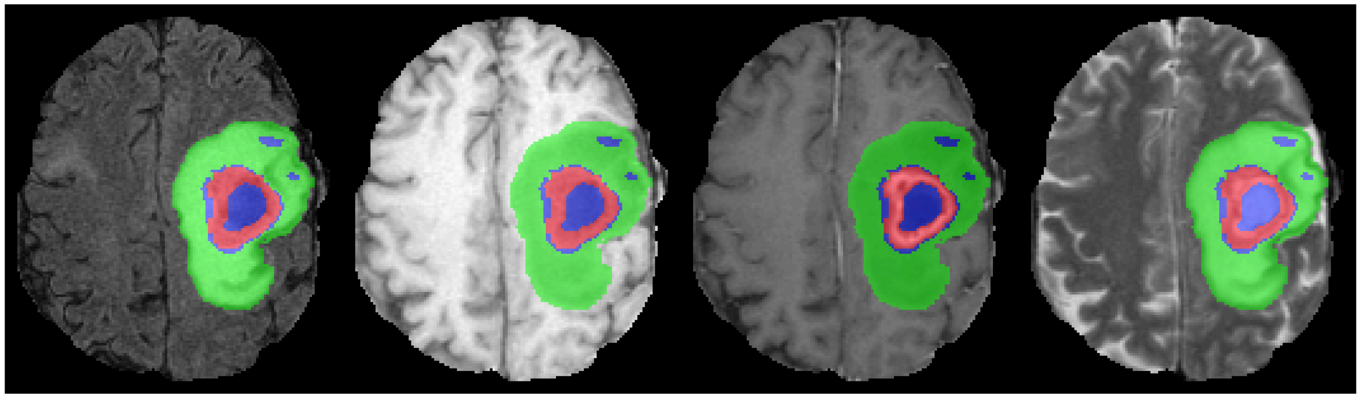
Notebook structure

Here's a rough outline of this notebook:

1. Check MONAI is installed and install if not (plus any extra dependencies)
2. Import libraries
3. Set up the data (download or create)
4. Prepare for training -- create transforms, datasets, dataloaders
5. Do some quick visualisations
6. Create our model, loss function, etc.
7. Train 🎉 🥳 🎊
8. Check results

Dataset

The dataset used here is the Decathlon 3D brain tumor dataset. We generate a 2D dataset by combining slices containing tumour, and those containing no tumourous voxels (considered healthy). We'll download the pre-computed dataset from Google Drive, but the script is available in case you're interested.



User input

Some sections have been left blank for you, the user, to complete. Suggested solutions are hidden in a collapsible text box. Feel free to rely on these as much as necessary, but you will get the most of the experience if you think it through or search the internet prior to displaying the results. Remember there are lots of ways of achieving the same goal, so it's OK if your solutions are different!

Extensions

1. We have a small amount of acceleration thanks to the `CacheDataset`, can you think of any other ways to make this training faster?

2. What methods could we use to reduce overfitting? Think about transforms and our model.
3. Could you use a different model or loss function altogether? Does this give better results?

Open in Colab

1. Check MONAI is installed

This checks if MONAI is installed, and if not installs it (plus any optional extras that might be needed for this notebook).

```
!python -c "import monai" || pip install -q "monai-weekly[nibabel, tqdm]"
!python -c "import gdown" || pip install -q "gdown"
!python -c "import matplotlib" || pip install -q matplotlib
%matplotlib inline
```

```
Traceback (most recent call last):
  File "<string>", line 1, in <module>
ModuleNotFoundError: No module named 'monai'
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.5/1.5 MB 42.6 MB/s eta 0:00:00
```

2. Import libraries

```
# Copyright 2022 MONAI Consortium
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#     http://www.apache.org/licenses/LICENSE-2.0
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

from glob import glob
import matplotlib.pyplot as plt
import numpy as np
import os
import tempfile
import torch
from tqdm import tqdm, trange
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

import monai
from monai.apps import download_and_extract
from monai.data import CacheDataset, DataLoader, pad_list_data_collate, partition_dataset
from monai.networks import eval_mode
import monai.transforms as mt
from monai.utils import set_determinism

monai.config.print_config()

# Use GPU if possible, else CPU (not recommended)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

```
# Set deterministic training for reproducibility
set_determinism(seed=42)
```

```
MONAI version: 1.5.dev2446
Numpy version: 1.26.4
Pytorch version: 2.5.1+cu121
MONAI flags: HAS_EXT = False, USE_COMPILED = False, USE_META_DICT = False
MONAI rev id: 218216250ce297265400abe56ee915898d75a2ec
MONAI __file__: /usr/local/lib/python3.10/dist-packages/monai/__init__.py

Optional dependencies:
Pytorch Ignite version: NOT INSTALLED or UNKNOWN VERSION.
ITK version: NOT INSTALLED or UNKNOWN VERSION.
Nibabel version: 5.3.2
scikit-image version: 0.24.0
scipy version: 1.13.1
Pillow version: 11.0.0
Tensorboard version: 2.17.1
gdown version: 5.2.0
TorchVision version: 0.20.1+cu121
tqdm version: 4.66.6
lmdb version: NOT INSTALLED or UNKNOWN VERSION.
psutil version: 5.9.5
pandas version: 2.2.2
einops version: 0.8.0
transformers version: 4.46.2
mlflow version: NOT INSTALLED or UNKNOWN VERSION.
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
```

```
# figure out if we're running in Google Colab. Set paths accordingly.
try:
    from google.colab import drive
    drive.mount('/content/drive', force_remount=True)
    best_model_folder = "/content/drive/MyDrive/saved_models/"
    os.makedirs(best_model_folder, exist_ok=True)
except:
    best_model_folder = ""

best_model_path = os.path.join(best_model_folder, "best_model_2d_glioma_classification.pth")
```

```
Mounted at /content/drive
```

3. Setup data

You can specify a directory with the `MONAI_DATA_DIRECTORY` environment variable.
This allows you to save results and reuse downloads.
If not specified a temporary directory will be used.

2D data

We'll download the pre-computed dataset from Google Drive, but the script is available in case you're interested.

```
directory = os.environ.get("MONAI_DATA_DIRECTORY")
root_dir = tempfile.mkdtemp() if directory is None else os.path.expanduser(directory)
print(root_dir)
```

```
/tmp/tmpfbh20ybt
```

```
download_from_gdrive = True
task = "Task01_BrainTumour"
output_dir = os.path.join(root_dir, task + "2D")

if download_from_gdrive:
    resource = "https://drive.google.com/uc?id=1BB0S2PcY6yUR7TK-AeyCFoh6PyoJiH0E&export=download"
    md5 = "214a338a26778c84ddebca29822add56"
    compressed_file = os.path.join(root_dir, task + "2D.tar")
    download_and_extract(resource, compressed_file, root_dir, hash_val=md5)
else:
    %run -i ../utils/2d_slice_creator.py --path {output_dir} --download_path {root_dir} --task {task}
    pass

# get all the 2d images
images_healthy = glob(os.path.join(output_dir, "image_healthy", "*.nii.gz"))
images_tumour = glob(os.path.join(output_dir, "image_tumour", "*.nii.gz"))
data_dicts = []
data_dicts += [{"image": i, "label": [1, 0]} for i in images_healthy]
data_dicts += [{"image": i, "label": [0, 1]} for i in images_tumour]

# shuffle the data and sort into training and validation
train_files, val_files = partition_dataset(data_dicts, ratios=(8, 2), shuffle=True)
print("total num files:", len(data_dicts))
print("num training files:", len(train_files))
print("num validation files:", len(val_files))
```

```
Downloading...
From (original): https://drive.google.com/uc?id=1BB0S2PcY6yUR7TK-AeyCFoh6PyoJiH0E
From (redirected): https://drive.google.com/uc?id=1BB0S2PcY6yUR7TK-AeyCFoh6PyoJiH0E&confirm=t&uuid=b38c268e-cddd-420b-809d-83bdd499525f
To: /tmp/tmp5nzqeqk1/Task01_BrainTumour2D.tar
100%|██████████| 122M/122M [00:00<00:00, 230MB/s]2024-11-20 00:19:54,328 - INFO - Downloaded: /tmp/tmpfbh20ybt/Task01_BrainTumour2D.tar

2024-11-20 00:19:54,554 - INFO - Verified 'Task01_BrainTumour2D.tar', md5: 214a338a26778c84ddebca29822add56.
2024-11-20 00:19:54,556 - INFO - Writing into directory: /tmp/tmpfbh20ybt.
total num files: 968
num training files: 774
num validation files: 194
```

4. Prepare for training -- create transforms, datasets, dataloaders

Transforms

In terms of transforms, we first load both the image and its corresponding label. We then rotate by 90 degrees, crop out superfluous zeros around the edge of the images, scale the image between 0 and 1 and then convert from numpy to pytorch.

In MONAI, we have two forms of transforms -- those that act on arrays of data, and those that act on dictionaries of data. For classification tasks, both are acceptable. We've used dictionaries here (note the `d` suffix in our transform names) –– such that each image and its corresponding class are stored together.

Datasets and dataloaders

We use the `CacheDataset` which accelerates things by pre-computing the results of our deterministic transforms. We then use a batch size of 10 in our dataloader.

```
train_transforms = val_transforms = mt.Compose(
    [
        mt.LoadImaged("image"),
        mt.Rotate90d("image"),
        mt.CropForegroundd("image", source_key="image", k_divisible=16),
        mt.ScaleIntensityd("image", channel_wise=True),
        mt.EnsureTyped(["image", "label"]),
    ]
)
```

```
/usr/local/lib/python3.10/dist-packages/monai/utils/deprecate_utils.py:321: FutureWarning: monai.transforms.croppad.dictionary CropForegroundd.__init__:allow_smaller: Current default
  warn_deprecated(argname, msg, warning_category)
```

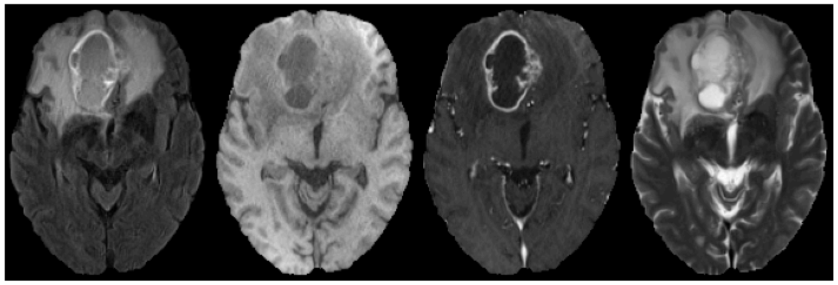
```
# The cache dataset loads in our images and keeps them in the RAM to save time on each loop.
# We use the pad_list_data_collate so that all images in our batch of data are the same size.
num_workers = 2
train_ds = CacheDataset(
    data=train_files, transform=train_transforms, cache_rate=1.0, num_workers=num_workers)
train_loader = DataLoader(train_ds, batch_size=10,
                          num_workers=num_workers, collate_fn=pad_list_data_collate)
val_ds = CacheDataset(
    data=val_files, transform=train_transforms, cache_rate=1.0, num_workers=num_workers)
val_loader = DataLoader(val_ds, batch_size=10,
                        num_workers=num_workers, collate_fn=pad_list_data_collate)
```

```
Loading dataset: 100%|██████████| 774/774 [00:11<00:00, 68.01it/s]
Loading dataset: 100%|██████████| 194/194 [00:02<00:00, 77.12it/s]
```

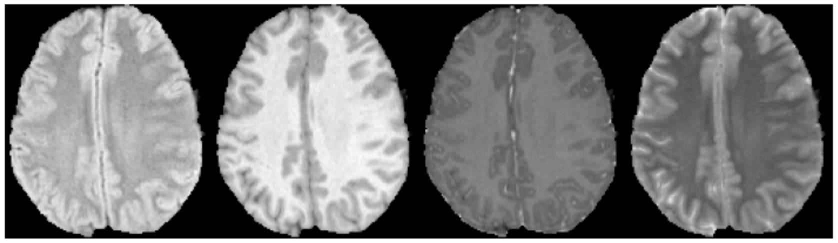
5. Display some examples

```
nrow, ncol = 5, 2
num_files = nrow * ncol
files_to_visualize = np.random.choice(train_files, size=num_files, replace=False)
fig, axes = plt.subplots(nrow, ncol, figsize=(20, 20), facecolor='white')
for f, ax in zip(files_to_visualize, axes.flatten()):
    data = train_transforms(f)
    # different modalities side by side
    img = np.concatenate(list(data["image"]), axis=1)
    im_show = ax.imshow(img, cmap="gray")
    ax.set_title(f"Has glioma? {['yes', 'no'][data['label'][0]]}")
    ax.axis("off")
```

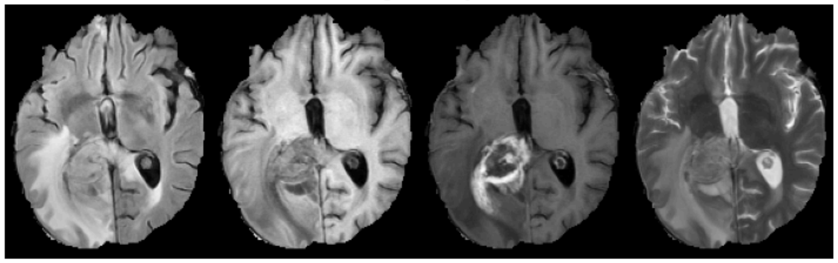
Has glioma? yes



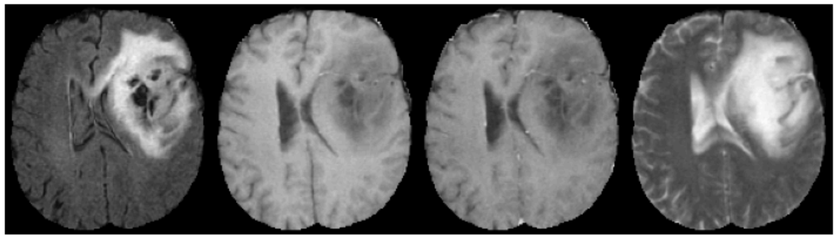
Has glioma? no



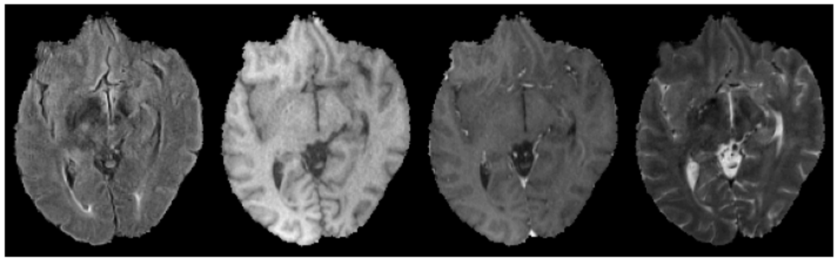
Has glioma? yes



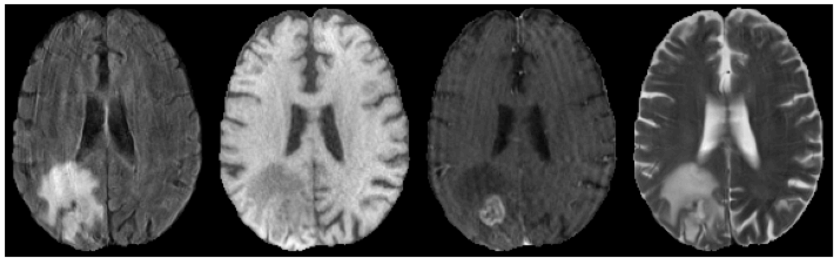
Has glioma? yes



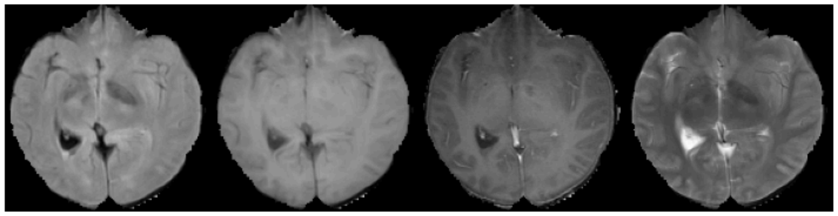
Has glioma? no



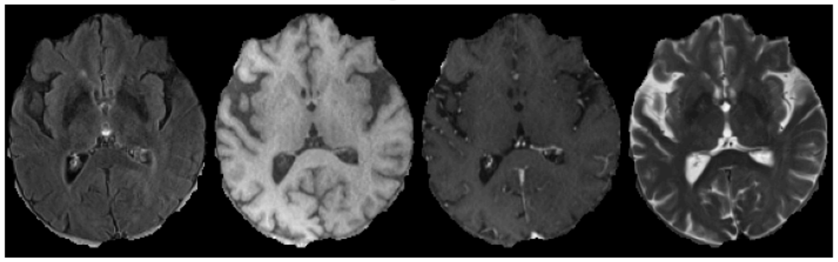
Has glioma? yes



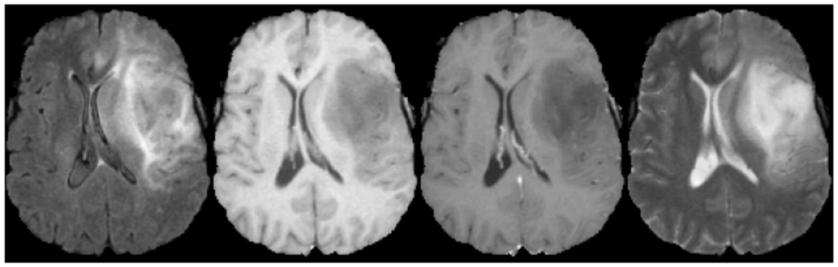
Has glioma? no



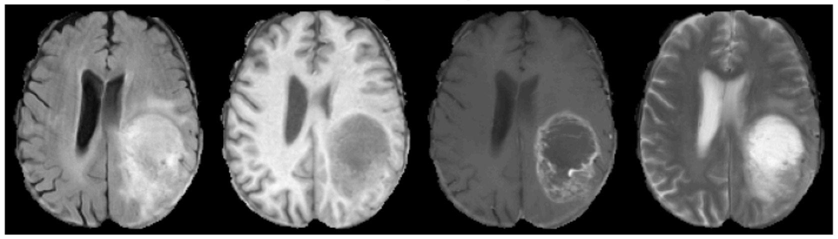
Has glioma? no



Has glioma? yes



Has glioma? yes



6. Create the model, loss function, etc.

The model

Which network do you want to use? How many channels should go in and how many should come out?

▼ 👉 Click to see an example model! 👈

```
model = monai.networks.nets.DenseNet121(spatial_dims=2, in_channels=4, out_channels=2)
```

The loss function

Which loss function should you use? Remember that you're doing classification. How many channels are you dealing with? What kind of activation is needed?

▼ 👉 Click to see an example loss function! 👈

```
loss_function = torch.nn.CrossEntropyLoss()
```

```
# Insert your model here

#model = monai.networks.nets.SENet154(spatial_dims = 2, in_channels = 4)
#model = monai.networks.nets.SEResNet50(spatial_dims=2, in_channels=4)
#model = monai.networks.nets.SEResNext101(spatial_dims=2, in_channels=4)
model = monai.networks.nets.ResNet(spatial_dims=2, n_input_channels=4, block = 'bottleneck' , layers=[3, 4, 6, 3], block_inplanes=[64, 128, 256, 512])
model.to(device)
```

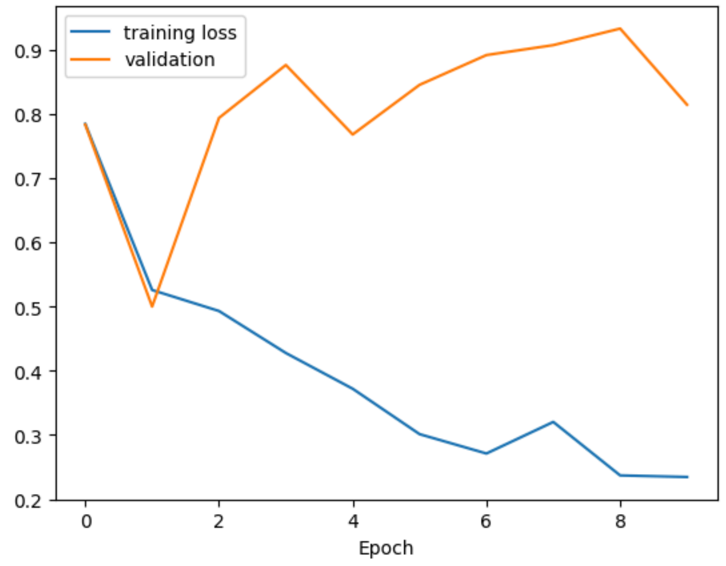
```
# Insert your loss function here
loss_function = torch.nn.CrossEntropyLoss()


# Helper function for inferring during validation
def infer_seg(images, model):
    return model(images).argmax(0)


# use Adam optimizer
optimizer = torch.optim.Adam(model.parameters(), 1e-3)
```

7. Train!

```
# Start a typical PyTorch training
max_epochs = 10
val_interval = 1
best_metric = -1
best_metric_epoch = -1
losses, metrics = [], []

tr = trange(max_epochs)
for epoch in tr:
    epoch_loss = 0

    # Training phase
    model.train()
    for batch_data in train_loader:
        inputs, labels = batch_data["image"], batch_data["label"]
        inputs, labels = inputs.to(device), labels.to(device)

        # Convert one-hot encoded labels to class indices, if necessary
        if len(labels.shape) > 1 and labels.shape[1] > 1:  # Check if labels are one-hot encoded
            labels = labels.argmax(dim=1)  # Convert to class indices

        # If labels have extra dimensions, squeeze them
        labels = labels.squeeze()  # Ensures labels are 1D tensor of class indices

        optimizer.zero_grad()
        outputs = model(inputs)

        # CrossEntropyLoss already applies LogSoftmax, so no need for Softmax
        loss = loss_function(outputs, labels.long())  # Ensure labels are long type for CrossEntropyLoss
        loss.backward()
        optimizer.step()

        epoch_loss += loss.item()

    epoch_loss /= len(train_loader)
    losses.append(epoch_loss)

    # Validation phase
    if (epoch + 1) % val_interval == 0:
        model.eval()  # Set model to evaluation mode
        num_correct = 0
        metric_count = 0
        with torch.no_grad():  # Disable gradient calculation during validation
            for val_data in val_loader:
                val_images, val_labels = val_data["image"], val_data["label"]
                val_images, val_labels = val_images.to(device), val_labels.to(device)

                # Convert one-hot encoded labels to class indices, if necessary
                if len(val_labels.shape) > 1 and val_labels.shape[1] > 1:
                    val_labels = val_labels.argmax(dim=1)  # Convert to class indices

                val_labels = val_labels.squeeze()  # Ensure labels are 1D tensor of class indices

                val_outputs = model(val_images)

                # Compare the predicted class (argmax) with the true class
                value = torch.eq(val_outputs.argmax(dim=1), val_labels)
                metric_count += len(value)
                num_correct += int(value.sum().item())

        metric = num_correct / metric_count
        metrics.append(metric)

        # If the current validation metric is better, save the model
        if metric > best_metric:
            best_metric = metric
            best_metric_epoch = epoch + 1
            torch.save(model.state_dict(), best_model_path)

    tr.set_description(f"Loss: {epoch_loss:.4f}, best val metric: {best_metric:.4f} at epoch {best_metric_epoch}")
```

```
Loss: 0.2345, best val metric: 0.9330 at epoch 9: 100%|██████████| 10/10 [04:17<00:00, 25.71s/it]
```

```
fig, ax = plt.subplots(facecolor='white')
ax.plot(losses, label="training loss")
ax.plot(metrics, label="validation")
ax.set_xlabel("Epoch")
_ = ax.legend()
```



```
model.load_state_dict(torch.load(best_model_path))
_ = model.eval()
```

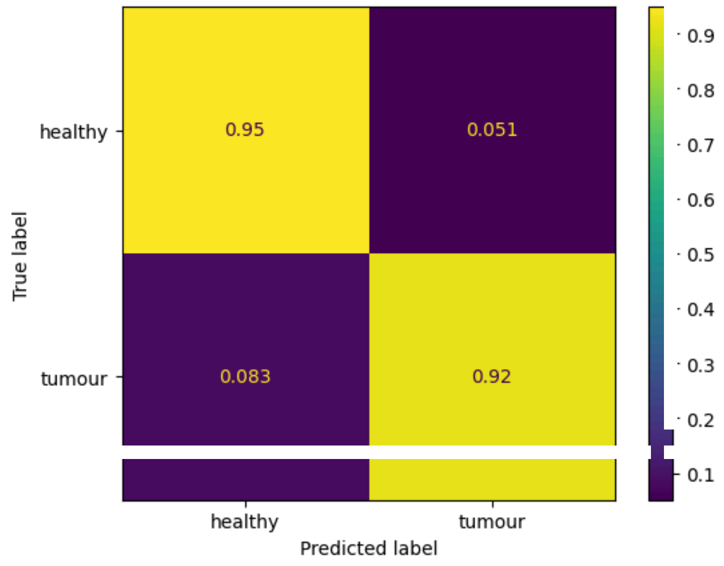
```
<ipython-input-106-df4a6896f45c>:1: FutureWarning: You are using `torch.load` with `weights_only=False` (the current default value), which uses the default pickle module implicitly. I
  model.load_state_dict(torch.load(best_model_path))
```

8. Check classifications

Loop over validation files, and plot a confusion matrix so we get a feeling for how we're doing!

```
y_pred = torch.tensor([], dtype=torch.float32, device=device)
y = torch.tensor([], dtype=torch.long, device=device)

for data in tqdm(val_loader):
    images, labels = data["image"].to(device), data["label"].to(device
    outputs = model(images).detach()
    y_pred = torch.cat([y_pred, outputs], dim=0)
    y = torch.cat([y, labels], dim=0)

y_pred = y_pred.argmax(dim=1)
y = y.argmax(dim=1)

cm = confusion_matrix(
    y.cpu().numpy(),
    y_pred.cpu().numpy(),
    normalize='true',
)
disp = ConfusionMatrixDisplay(
    confusion_matrix=cm,
    display_labels=["healthy", "tumour"],
)
```

```
100%|██████████| 20/20 [00:01<00:00, 12.44it/s]
```



Extensions