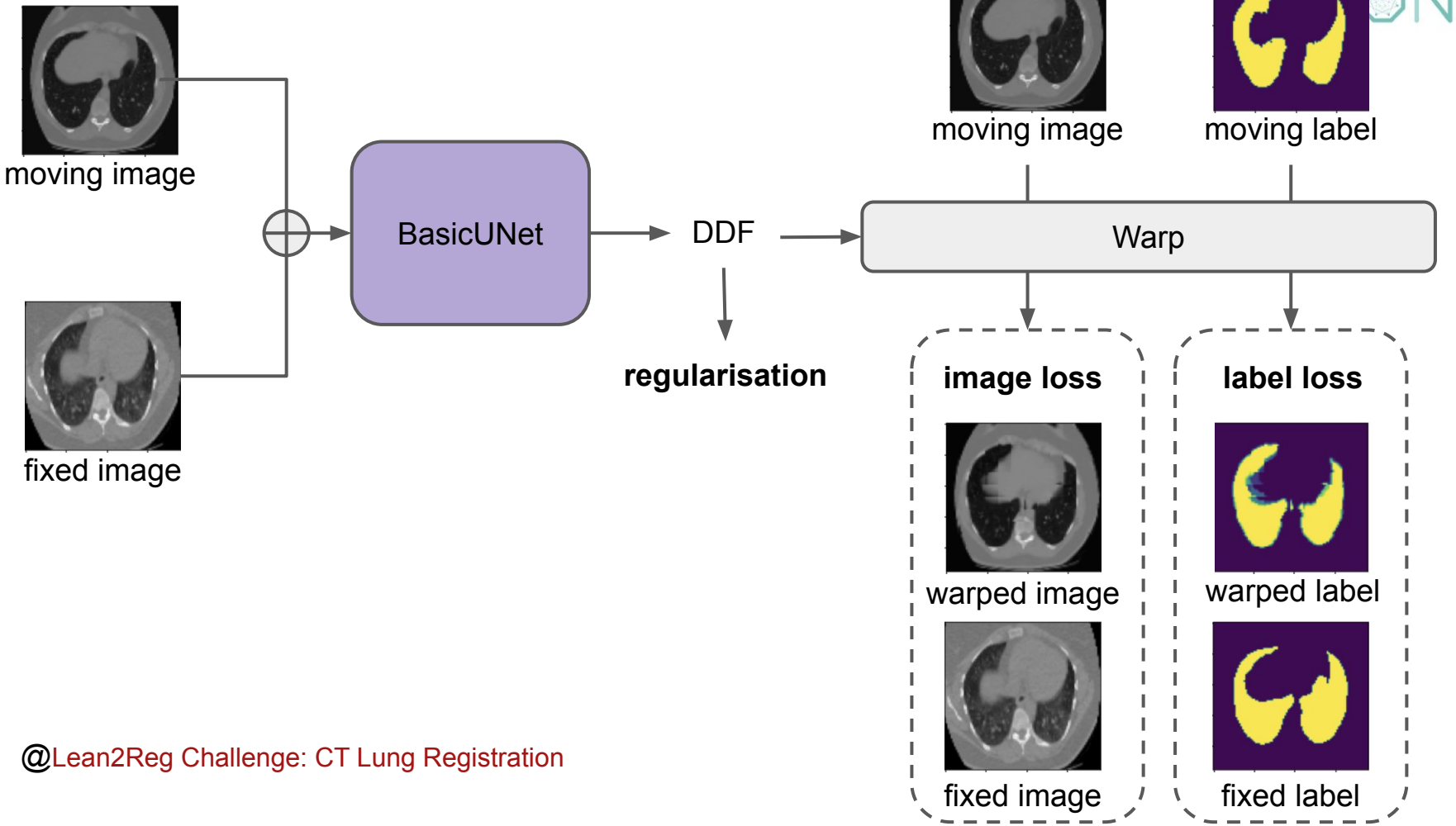
# Medical Image Registration features in MONAI

# Registration

The process of transforming different image datasets into one coordinate system with matched imaging contents (i.e. corresponding anatomical/pathological features/ROIs).

- ➔ across time-point
- ➔ across modality
- ➔ across patient
- ➔ ...

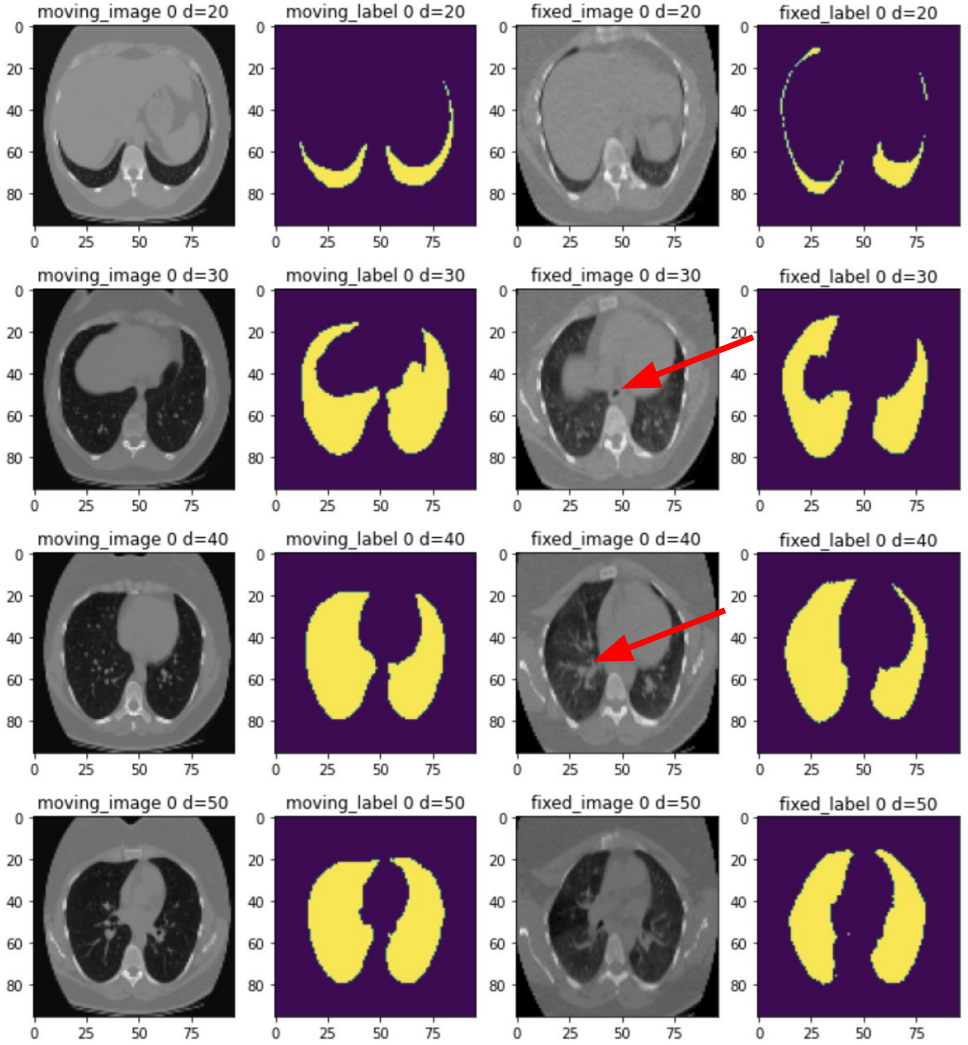



@Lean2Reg Challenge: CT Lung Registration

```
# initialise data_dict
data_dicts = [{"fixed_image": $FIXED_IMAGE_PATH$,
               "moving_image": $MOVING_IMAGE_PATH$,
               "fixed_label": $FIXED_LABEL_PATH$,
               "moving_label": $MOVING_LABEL_PATH$}]
```

```
data_dicts = [{
        "fixed_image": os.path.join(data_dir,"scans/case_%03d_exp.nii.gz" % idx),
        "moving_image": os.path.join(data_dir,"scans/case_%03d_insp.nii.gz" % idx),
        "fixed_label": os.path.join(data_dir,"lungMasks/case_%03d_exp.nii.gz" % idx),
        "moving_label": os.path.join(data_dir,"lungMasks/case_%03d_insp.nii.gz" % idx),
    }for idx in range(1, 21)
]

train_files, val_files = data_dicts[:18], data_dicts[18:]

train_ds = CacheDataset(data=train_files, transform=train_transforms)
train_loader = DataLoader(train_ds, batch_size=1, shuffle=True, num_workers=4)
val_ds = CacheDataset(data=val_files, transform=val_transforms)
val_loader = DataLoader(val_ds, batch_size=1, num_workers=4)
```

→ *Dictionary Transforms*


moving_image 0 d=20
moving_label 0 d=20
fixed_image 0 d=20
fixed_label 0 d=20
moving_image 0 d=30
moving_label 0 d=30
fixed_image 0 d=30
fixed_label 0 d=30
moving_image 0 d=40
moving_label 0 d=40
fixed_image 0 d=40
fixed_label 0 d=40
moving_image 0 d=50
moving_label 0 d=50
fixed_image 0 d=50
fixed_label 0 d=50

```
# initialise model
model = $MODEL_OBJECT$.to(device)
# initialise warp layer
warp_layer = Warp().to(device)
# initialise optimizer
optimizer = torch.optim.$OPTIMIZER$($TRAINABLE_PARAMETERS$, $LEARNING_RATE$)
```

- ➔ *GlobalNet*
- ➔ *LocalNet*

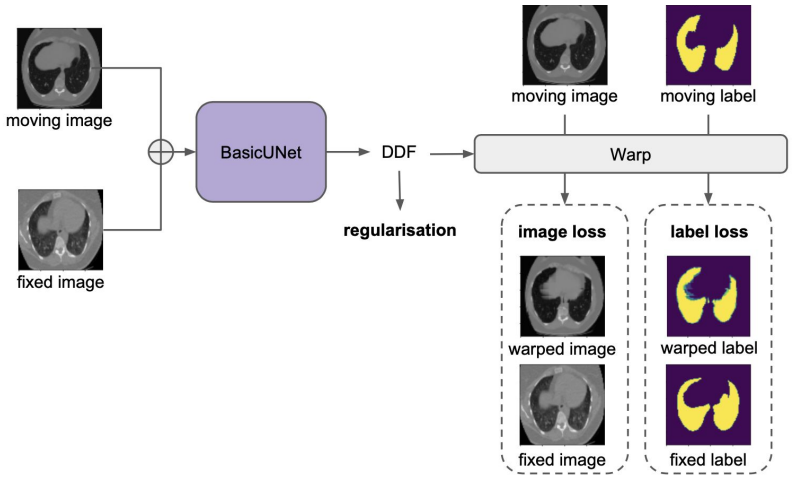
```
model = LocalNet(
    spatial_dims=3,
    in_channels=2,
    out_channels=3,
    num_channel_initial=32,
    extract_levels=[0, 1, 2, 3],
    out_activation=None,
    out_kernel_initializer="zeros").to(device)
warp_layer = Warp().to(device)
optimizer = torch.optim.Adam(model.parameters(), 1e-5)
```

```
# initialise loss
image_loss_fn = $IMAGE_LOSS_OBJECT$
label_loss_fn = $LABEL_LOSS_OBJECT$
regularization_fn = $REGULARISATION_LOSS_OBJECT$
```

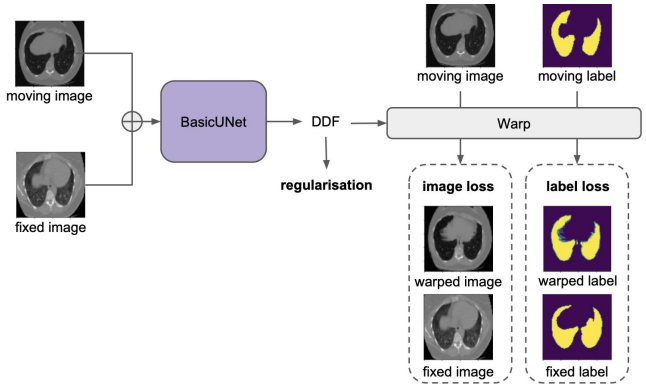
➔ *LocalNormalizedCorrelationLoss*
➔ *GlobalMutualInformationLoss*

```
image_loss_fn = MSELoss()
label_loss_fn = DiceLoss()
label_loss_fn = MultiScaleLoss(label_loss, scales=[0, 1, 2, 4, 8, 16])
regularization_fn = BendingEnergyLoss()
```

```python
# define forward pass
def forward(batch_data, model):
    fixed_image = batch_data["fixed_image"].to(device)
    fixed_label = batch_data["fixed_label"].to(device)
    moving_image = batch_data["moving_image"].to(device)
    moving_label = batch_data["moving_label"].to(device)
    # predict DDF through LocalNet
    ddf = model(torch.cat((moving_image, fixed_image), dim=1))
    # warp moving image and label with the predicted ddf
    pred_image = warp_layer(moving_image, ddf)
    pred_label = warp_layer(moving_label, ddf)
    return pred_image, pred_label, ddf
```



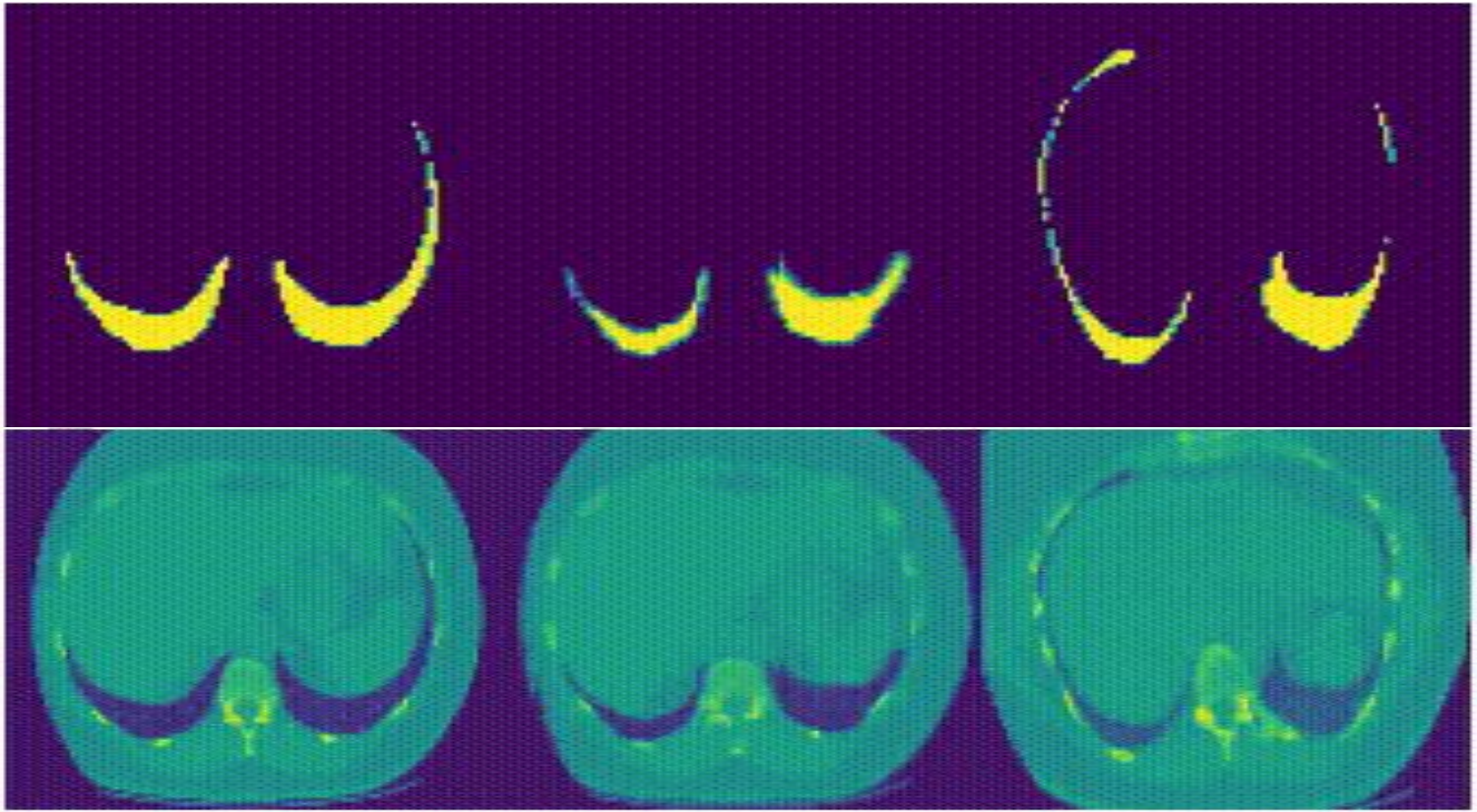
```
# train
for batch_data in train_loader:
        step += 1
        optimizer.zero_grad()
        pred_image, pred_label, ddf = forward(batch_data, model)

        fixed_image = batch_data["fixed_image"].to(device)
        image_loss = image_loss_fn(pred_image, fixed_image) * $IMAGE_LOSS_WEIGHT$
        fixed_label = batch_data["fixed_label"].to(devic)
        label_loss = label_loss_fn(pred_label, fixed_label) * $LABEL_LOSS_WEIGHT$
        regularisation_loss = regularization_fn(ddf) * $REGULARISATION_WEIGHT$
        loss = image_loss + label_loss + regularisation_loss

        loss.backward()
        optimizer.step()
        epoch_loss += loss.item()
```



```
# validate
for batch_data in val_loader:
        pred_image, pred_label, ddf = forward(batch_data, model)
        # update dice metric
        fixed_label = batch_data["fixed_label"].to(device)
        dice_metric(y_pred=val_pred_label, y=val_fixed_label)
```

```
best_metric = 0
dice_metric = DiceMetric(include_background=True, reduction="mean", get_not_nans=False)
for epoch in range(max_epochs):
    model.train()
    $TRAIN$
    # validate every val_interval epochs
    if (epoch + 1) % val_interval == 0 or epoch == 0:
        model.eval()
        with torch.no_grad():
            $VALIDATE$
            metric = dice_metric.aggregate().item()
            dice_metric.reset()
            metric_values.append(metric)
        # save state_dict when new best resulat achieved
        if metric > best_metric:
            best_metric = metric
            torch.save(model.state_dict(), os.path.join(root_dir,"best_metric_model.pth"))
```

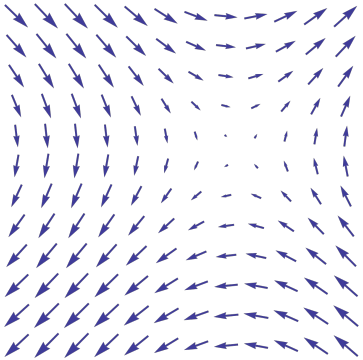
moving fixed pred



# other deformation predictions

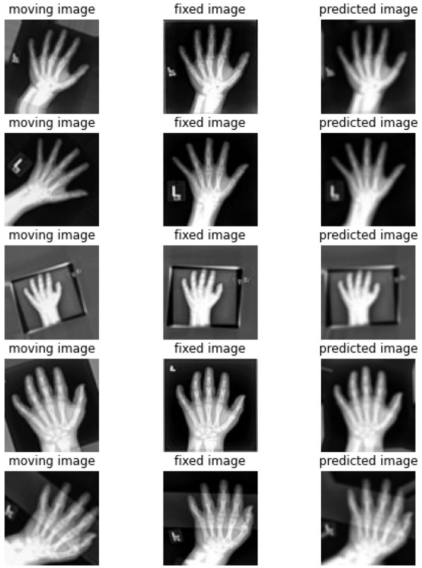
DVF
(dense velocity field)

➔ *DVF2DDF*



affine transformation

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ 1 & 1 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



MONAI+

# acknowledgement