MONAI-to-MONAI Deploy (english version)

Deploying a MONAI Model with MONAI Deploy: A Step-by-Step Guide

If you have developed a medical AI model using **MONAI**, you can deploy it using **MONAI Deploy** to make it available for inference in clinical or research environments. This guide provides a **detailed, step-by-step explanation** of how to **convert, package, and deploy** your MONAI model using MONAI Deploy.

📌 Overview of the Deployment Process

1. **Train a model using MONAI** (or use a pre-trained model).
2. **Convert the trained model** into a deployable format (`.pt`, `.onnx`, `.torchscript`).
3. **Create a MONAI Deploy App (MDA)**:
   - Define an **application structure**.
   - Implement an **Inference Operator**.
4. **Package and deploy the app** using MONAI Deploy App SDK.
5. **Run inference** using MONAI Deploy.

🔷 Step 1: Train or Load a MONAI Model

If you haven't already trained a MONAI model, you can train a simple UNet model for segmentation.

Example: Train and Save a MONAI Model

```
import monai
import torch
from monai.networks.nets import UNet
from monai.transforms import Compose, EnsureChannelFirst, R
esize, ToTensor
```

```
from monai.data import Dataset, DataLoader

# Create a simple UNet model
model = UNet(
    spatial_dims=2,
    in_channels=1,
    out_channels=2,
    channels=(16, 32, 64, 128, 256),
    strides=(2, 2, 2, 2),
    num_res_units=2
)

# Save the model
torch.save(model.state_dict(), "monai_model.pth")
print("Model saved as monai_model.pth")
```

👉 **Note:**

- You can also export the model to **ONNX** or **TorchScript** for deployment.

🔷 Step 2: Convert the MONAI Model to a Deployable Format

Before deploying, you need to **convert** the MONAI model to a format compatible with MONAI Deploy.

Convert to ONNX

```
import torch
import onnx

# Load the trained model
model = UNet(
    spatial_dims=2,
    in_channels=1,
    out_channels=2,
    channels=(16, 32, 64, 128, 256),
    strides=(2, 2, 2, 2),
    num_res_units=2
```

```
)
model.load_state_dict(torch.load("monai_model.pth"))
model.eval()

# Create dummy input
dummy_input = torch.randn(1, 1, 128, 128)

# Convert to ONNX format
onnx_path = "monai_model.onnx"
torch.onnx.export(model, dummy_input, onnx_path, input_name
s=['input'], output_names=['output'])
print(f"Model exported to {onnx_path}")
```

👉 **Supported Formats:**

- `.pth` → PyTorch
- `.onnx` → ONNX
- `.torchscript` → TorchScript (for optimized inference)

🔷 Step 3: Create a MONAI Deploy App (MDA)

MONAI Deploy uses a modular structure with **Operators** to handle inference, preprocessing, and postprocessing.

📁 Project Structure

```
monai_deploy_app/
├── app.py                    # Main MONAI Deploy App
├── operators/
│   ├── inference_operator.py  # Defines the inference proc
ess
│   └── transform_operator.py  # Preprocessing & Postproces
sing
└── configs/
    └── settings.json          # Configuration file
```

📌 3.1 Install MONAI Deploy SDK

```
pip install monai-deploy-app-sdk
```

📌 3.2 Define the MONAI Deploy Application

Create `app.py` inside `monai_deploy_app/`:

```
# app.py - MONAI Deploy App for Model Inference
from monai.deploy.core import Application, resource
from operators.inference_operator import InferenceOperator
from operators.transform_operator import TransformOperator

class MonaiDeployApp(Application):
    """
    MONAI Deploy Application for AI model inference.
    """
    @resource(cpu=4, gpu=1, memory="4Gi")
    def compose(self):
        # Define pipeline components
        transform_op = TransformOperator()
        inference_op = InferenceOperator()

        # Define workflow pipeline
        self.add_flow(transform_op, inference_op)

if __name__ == "__main__":
    app = MonaiDeployApp()
    app.run()
```

📌 3.3 Create an Inference Operator

Create `operators/inference_operator.py`:

```
# inference_operator.py - Inference Operator
import torch
import numpy as np
from monai.deploy.core import Operator, ExecutionContext, I
OType
```

```
from monai.deploy.operators import DICOMDataLoader, DICOMSe
riesToVolumeConverter

class InferenceOperator(Operator):
    """
    Performs model inference using MONAI Deploy.
    """
    def __init__(self):
        super().__init__()
        self.model_path = "monai_model.onnx"
        self.device = "cuda" if torch.cuda.is_available() e
lse "cpu"
        self.model = torch.jit.load(self.model_path).to(sel
f.device)

    def compute(self, context: ExecutionContext):
        # Load input image
        image = context.input.get().asnumpy()
        image_tensor = torch.from_numpy(image).float().unsq
ueeze(0).to(self.device)

        # Perform inference
        output = self.model(image_tensor)

        # Convert result to numpy and send output
        result = output.cpu().detach().numpy()
        context.output.set(result)
```

📌 3.4 Preprocessing & Postprocessing Operator

Create `operators/transform_operator.py`:

```
# transform_operator.py - Preprocessing & Postprocessing
import numpy as np
from monai.deploy.core import Operator, ExecutionContext
from monai.transforms import Compose, EnsureChannelFirst, R
esize, ScaleIntensity
```

```
class TransformOperator(Operator):
    """
    Preprocessing and postprocessing operator for MONAI Deploy.
    """
    def __init__(self):
        super().__init__()
        self.transforms = Compose([
            EnsureChannelFirst(),
            ScaleIntensity(),
            Resize((128, 128))
        ])

    def compute(self, context: ExecutionContext):
        image = context.input.get().asnumpy()
        processed_image = self.transforms(image)
        context.output.set(processed_image)
```

🔷 Step 4: Package and Deploy the Model

Once the application is defined, we can **package** and **deploy** it.

📌 4.1 Package the MONAI Deploy App

```
monai-deploy package monai_deploy_app/ -o monai_model_package
```

This will generate a deployable containerized package.

📌 4.2 Deploy the MONAI Deploy App

You can run the model on a **local machine** or a **Kubernetes cluster**.

Run Locally

```
monai-deploy run monai_model_package --input test_image.dcm --output output_result.dcm
```

Run on Kubernetes

```
kubectl apply -f monai_deploy_app.yaml
```

🔷 Step 5: Run Inference

After deployment, you can run inference on a test image.

```
monai-deploy infer monai_model_package --input test_image.dcm --output result.dcm
```

👉 **Expected Output:**

- The model will process the `test_image.dcm` file and generate an output (`result.dcm`).
- The pipeline applies **preprocessing → inference → postprocessing**.

🎯 Conclusion

You have successfully **developed, converted, and deployed** a MONAI model using **MONAI Deploy**! 🚀

✅ **Trained a MONAI model**

✅ **Converted the model for deployment**

✅ **Created a MONAI Deploy App**

✅ **Packaged and deployed the model**

✅ **Ran inference on real medical images**

Now, your AI model is ready to be integrated into hospital PACS, cloud environments, or edge devices. 🎉

🔗 Additional Resources

- 📖 MONAI Deploy Docs
- 💾 MONAI GitHub
- 📘 NVIDIA Clara & MONAI Deploy

Let me know if you need further clarification! 🚀🔥