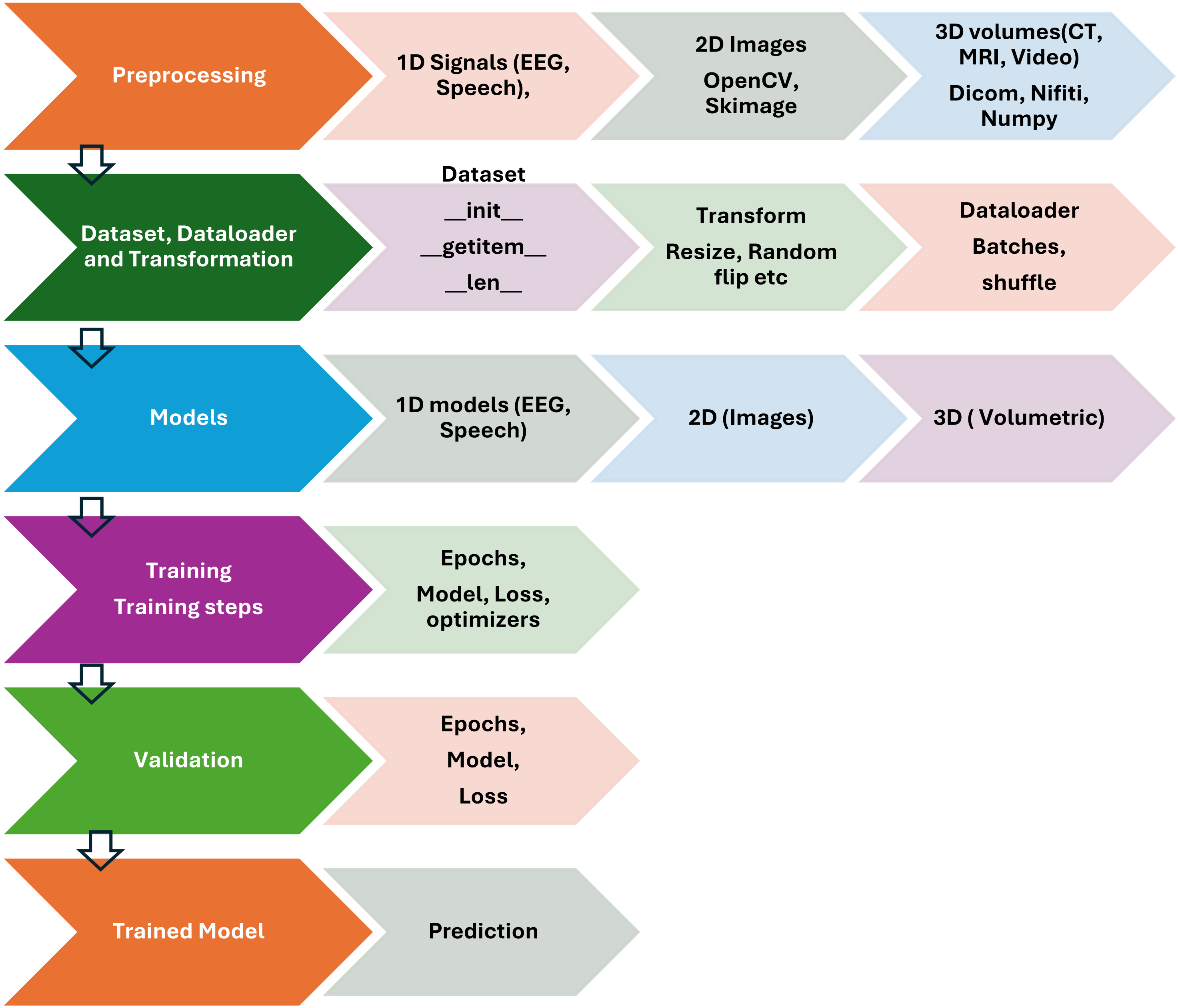
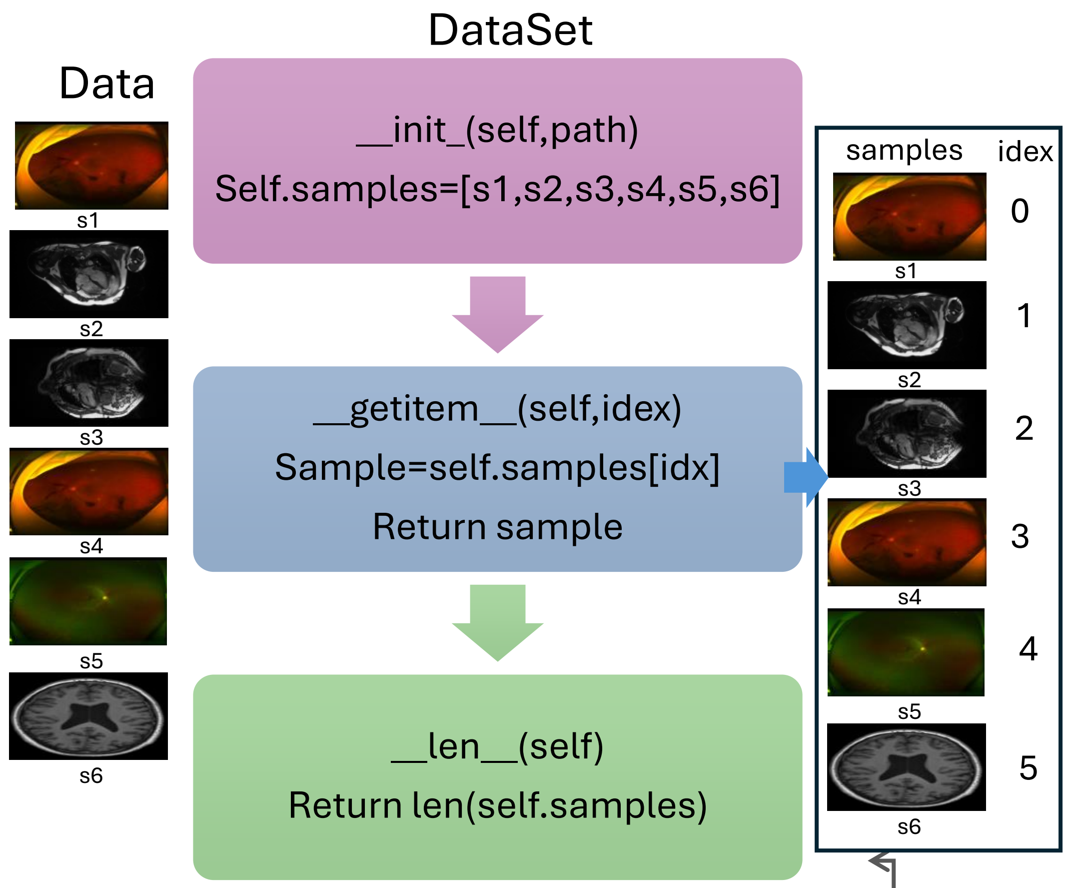
Tutorial

Abdul Qayyum (Ph.D.)


Preprocessing
1D Signals (EEG, Speech),
2D Images
OpenCV, Skimage
3D volumes(CT, MRI, Video)
Dicom, Nifiti, Numpy
Dataset, Dataloader and Transformation
Dataset
__init__
__getitem__
__len__
Transform
Resize, Random flip etc
Dataloader
Batches,
shuffle
Models
1D models (EEG, Speech)
2D (Images)
3D (Volumetric)
Training
Training steps
Epochs,
Model, Loss, optimizers
Validation
Epochs,
Model,
Loss
Trained Model
Prediction

Dataset and DataLoader



1. Initialize Sample Paths in __init__:

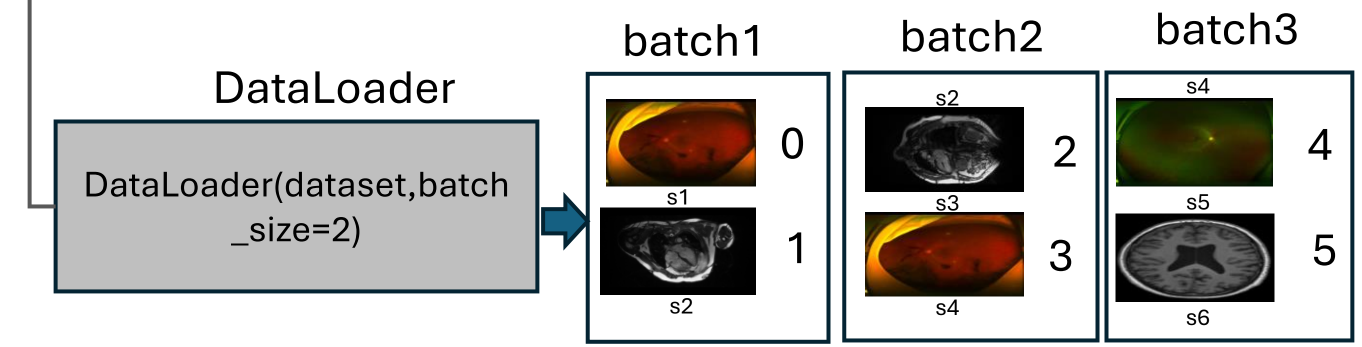
In the __init__ method of our custom dataset, we define or load the sample paths. For simplicity, we'll use a list of strings representing file paths.

2. Retrieve Samples Using __getitem__:

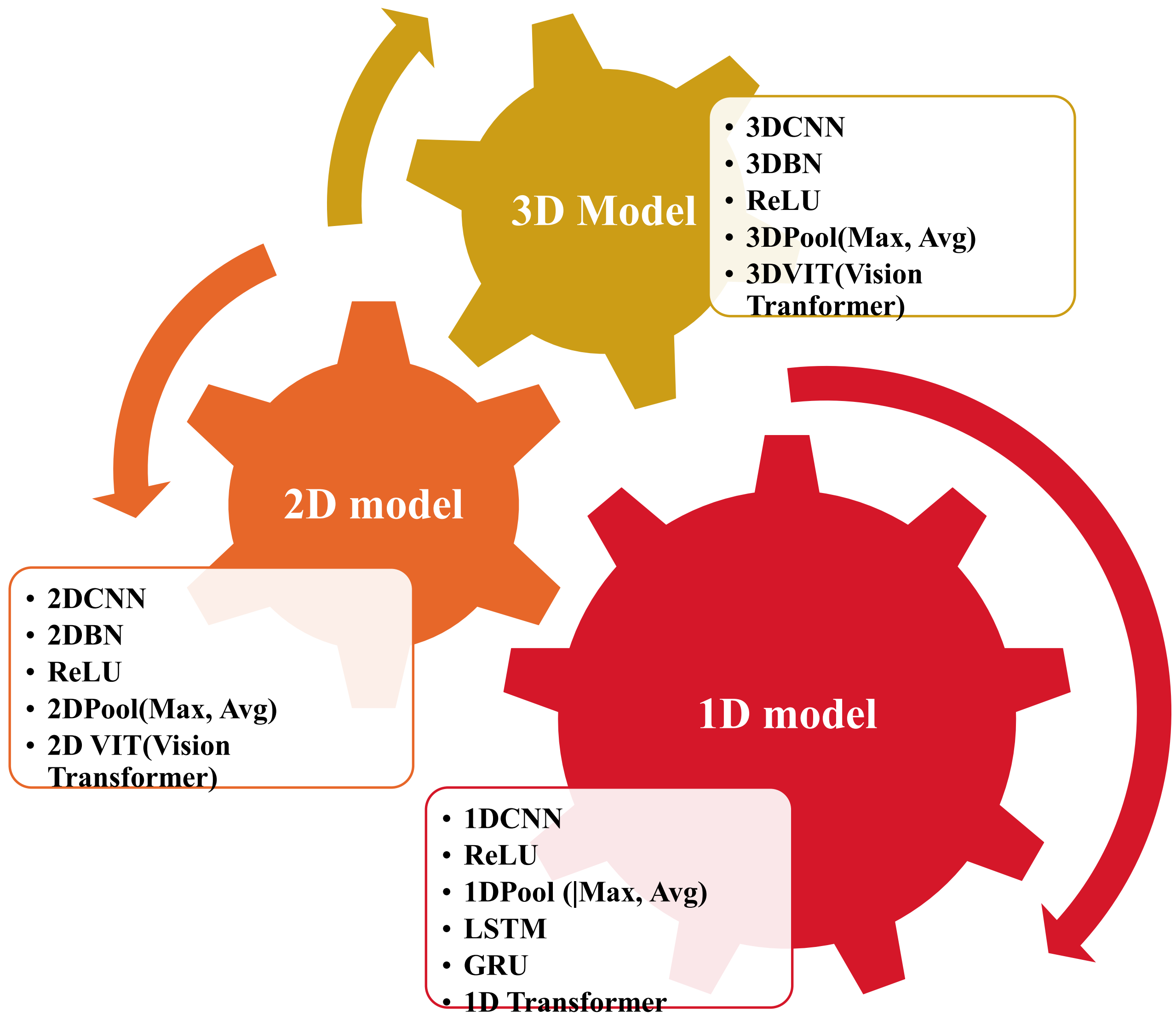
The __getitem__ method uses the index provided to retrieve a specific sample. For example, if the dataset contains ['s1', 's2', ...], calling dataset[0] will return 's1'.

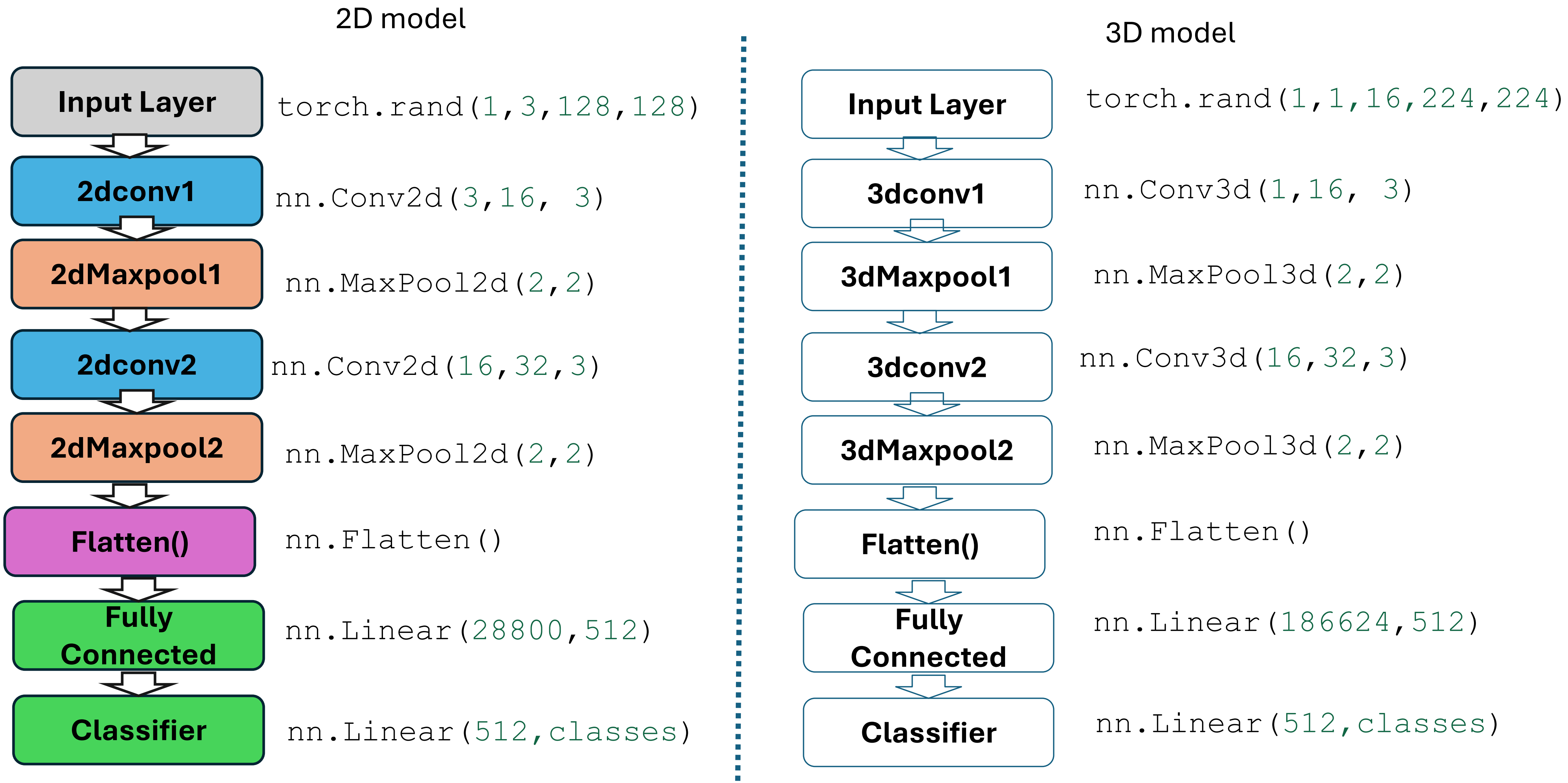
3. Divide into Batches Using DataLoader:

We use PyTorch's DataLoader to load the dataset in batches. By setting the batch_size parameter to 2, the DataLoader will automatically group samples into batches of 2.



Models


3D Model
• 3DCNN
• 3DBN
• ReLU
• 3DPool(Max, Avg)
• 3DVIT(Vision Tranformer)
2D model
• 2DCNN
• 2DBN
• ReLU
• 2DPool(Max, Avg)
• 2D VIT(Vision Transformer)
1D model
• 1DCNN
• ReLU
• 1DPool (|Max, Avg)
• LSTM
• GRU
• 1D Transformer



2D model
Input Layer
torch.rand(1,3,128,128)
2dconv1
nn.Conv2d(3,16, 3)
2dMaxpool1
nn.MaxPool2d(2,2)
2dconv2
nn.Conv2d(16,32,3)
2dMaxpool2
nn.MaxPool2d(2,2)
Flatten()
nn.Flatten()
Fully Connected
nn.Linear(28800,512)
Classifier
nn.Linear(512,classes)
3D model
Input Layer
torch.rand(1,1,16,224,224)
3dconv1
nn.Conv3d(1,16, 3)
3dMaxpool1
nn.MaxPool3d(2,2)
3dconv2
nn.Conv3d(16,32,3)
3dMaxpool2
nn.MaxPool3d(2,2)
Flatten()
nn.Flatten()
Fully Connected
nn.Linear(186624,512)
Classifier
nn.Linear(512,classes)

Loss Functions

Classification

CrossEntropyLoss

BCEWithLogitsLoss

BCELoss

NLLLoss

MSELoss

SmoothL1Loss

MarginRankingLoss

HingeEmbeddingLoss

Regression/Registration

Regression

Mean Squared Error Loss (MSELoss)

Mean Absolute Error Loss (MAE)

Huber Loss

Smooth L1 Loss

Registration

NCC

MSELoss

Grad3d

Segmentation

Cross-Entropy Loss

Dice Loss

BCE Loss

Tversky Loss

Focal Loss

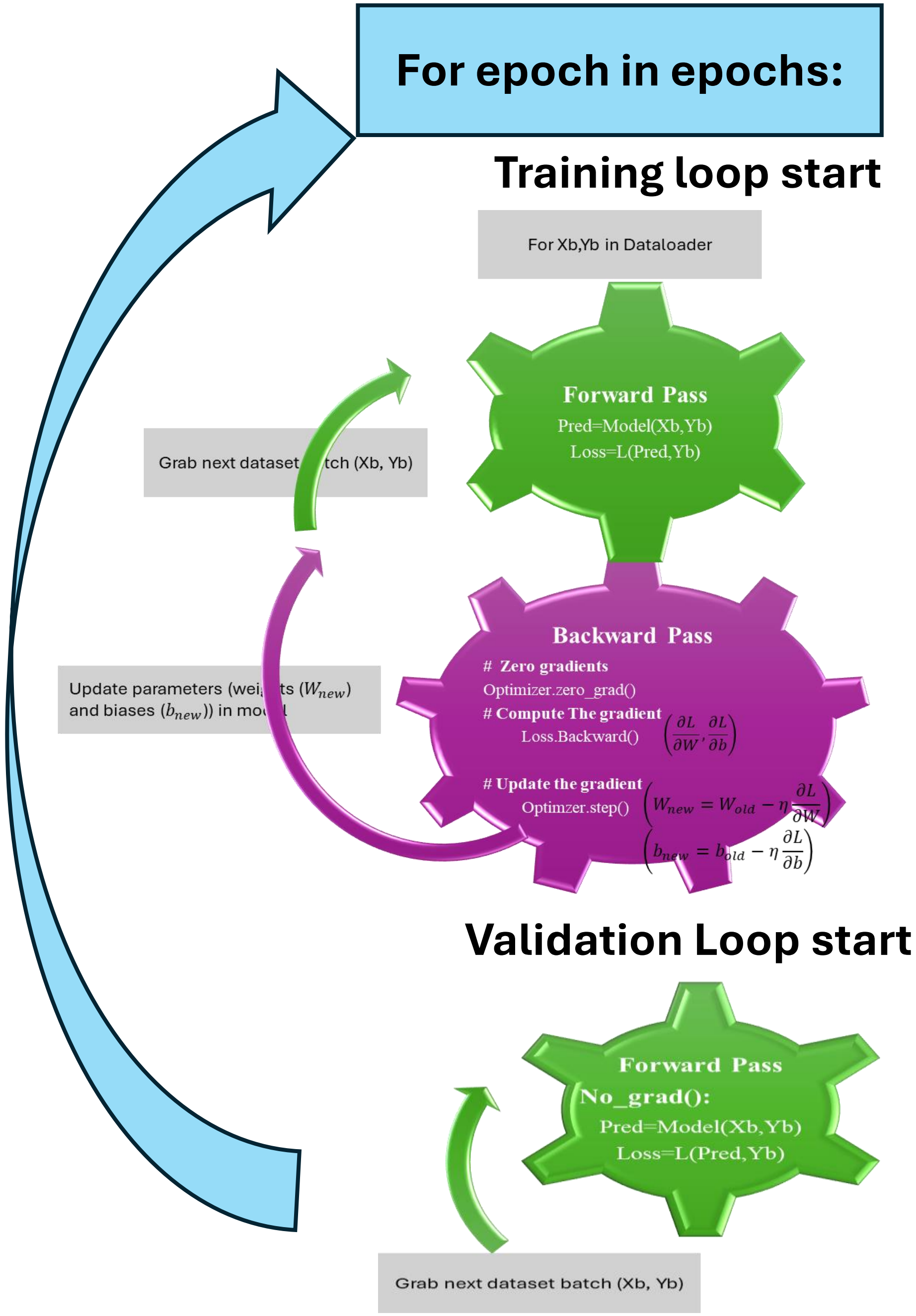
Combo Loss (Dice + BCE)

Optimizers

Optimizers:

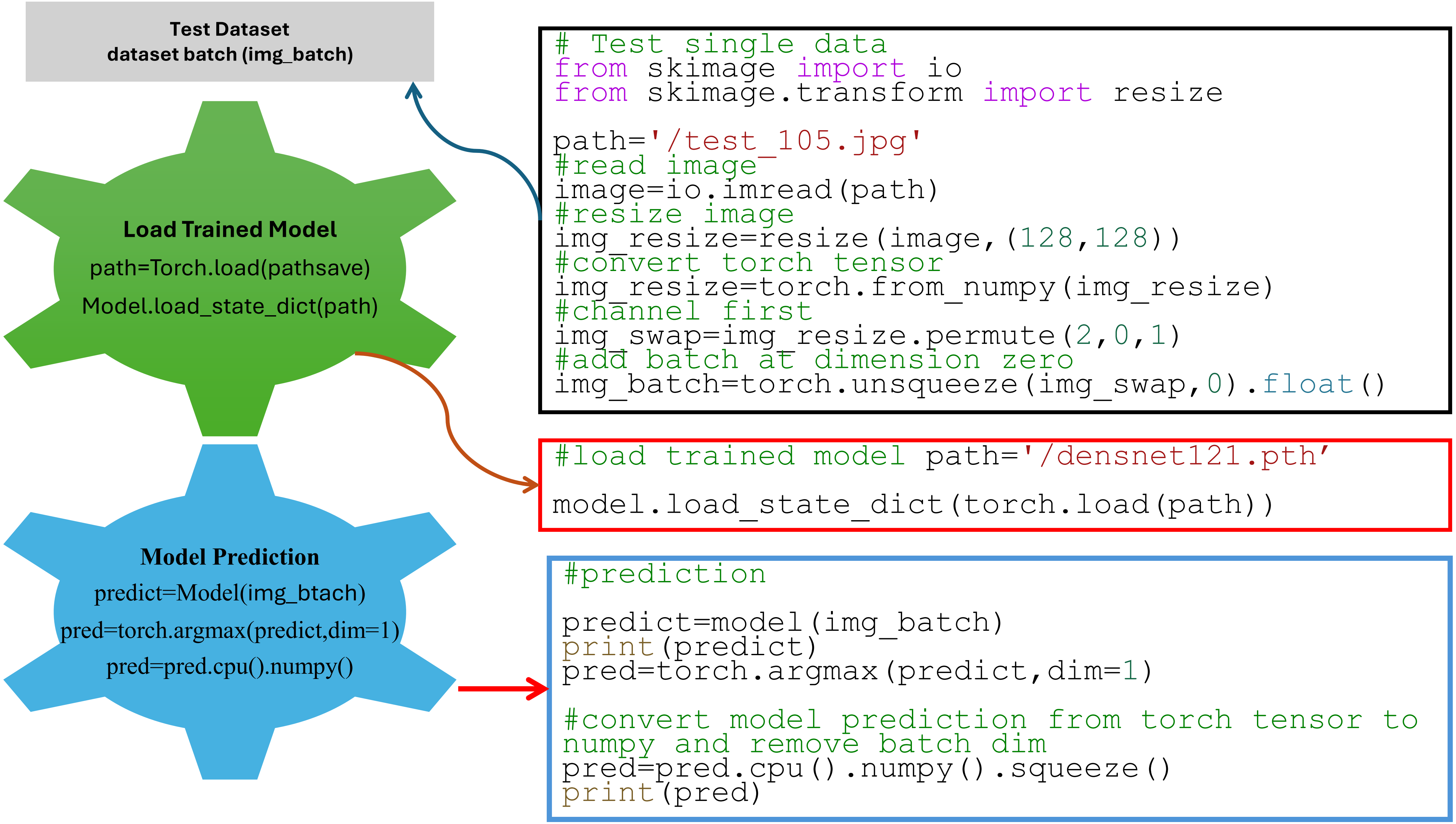
1. **SGD** (with or without momentum)
2. **Adam**
3. **RMSprop**
4. **Adagrad**
5. **Adadelta**
6. **AdamW**
7. **LBFGS**
8. **ASGD**
9. **SparseAdam**

Training and Validation Loop

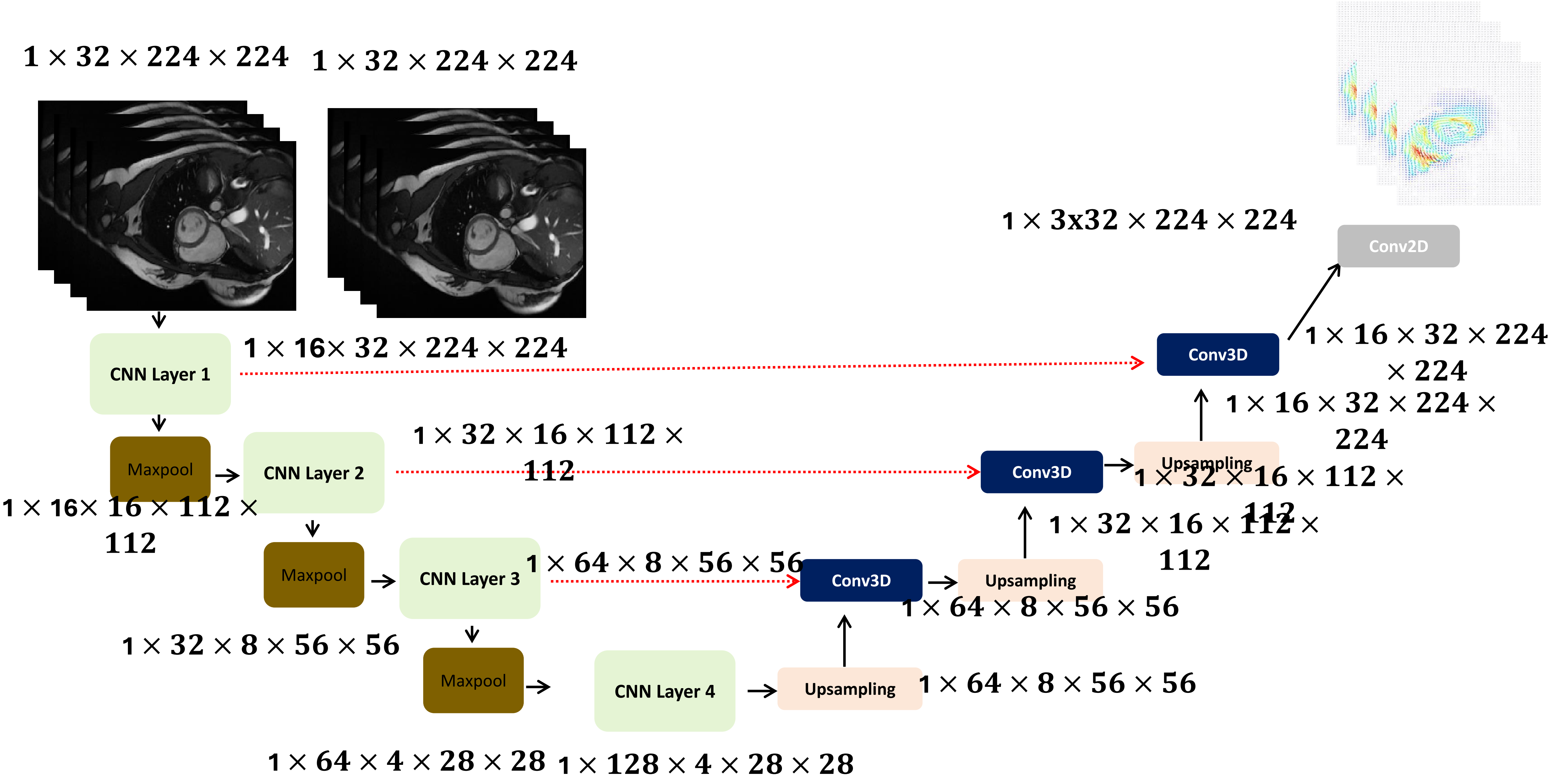



```python
import torch
import torch.nn as nn
import torch.optim as optim

# Define a simple neural network with a single fully connected layer
class SimpleNN(nn.Module):
    def __init__(self):
        super(SimpleNN, self).__init__()
        self.fc = nn.Linear(in_features=5, out_features=3)  # Single FC layer
    def forward(self, x):
        x = self.fc(x)  # Output from the FC layer
        return x

# Instantiate the model
model = SimpleNN()
# Define a loss function (e.g., Cross-Entropy Loss)
criterion = nn.CrossEntropyLoss()
# Define an optimizer (e.g., SGD)
optimizer = optim.SGD(model.parameters(), lr=0.01)
# Create a random input tensor (batch_size=3, input_features=5)
input_tensor = torch.randn(3, 5)
# Create a random target tensor (batch_size=3, number of classes=3)
target_tensor = torch.randint(0, 3, (3,))
# Zero the gradients before the backward pass
optimizer.zero_grad()
# Forward pass: Get model output
output = model(input_tensor)
# Compute the loss
loss = criterion(output, target_tensor)
# Backward pass: Compute gradients
loss.backward()
# Print the gradients for the FC layer
print("Gradients for fc weights:", model.fc.weight.grad)
print("Gradients for fc biases:", model.fc.bias.grad)
# Optimizer step (update the model parameters based on gradients)
optimizer.step()
```

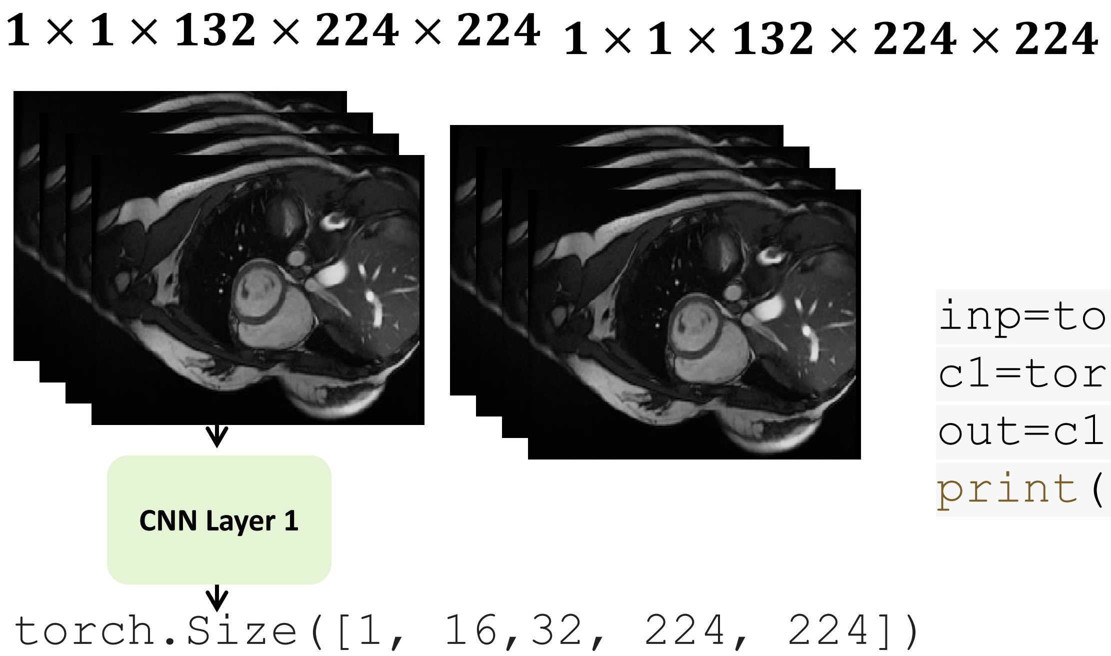
Testing/Validation


Test Dataset
dataset batch (img_batch)
Load Trained Model
path=Torch.load(pathsave)
Model.load_state_dict(path)
Model Prediction
predict=Model(img_btach)
pred=torch.argmax(predict,dim=1)
pred=pred.cpu().numpy()
# Test single data
from skimage import io
from skimage.transform import resize
path='/test_105.jpg'
#read image
image=io.imread(path)
#resize image
img_resize=resize(image,(128,128))
#convert torch tensor
img_resize=torch.from_numpy(img_resize)
#channel first
img_swap=img_resize.permute(2,0,1)
#add batch at dimension zero
img_batch=torch.unsqueeze(img_swap,0).float()
#load trained model path='/densnet121.pth'
model.load_state_dict(torch.load(path))
#prediction
predict=model(img_batch)
print(predict)
pred=torch.argmax(predict,dim=1)
#convert model prediction from torch tensor to numpy and remove batch dim
pred=pred.cpu().numpy().squeeze()
print(pred)

UNet Step by Step for Registraion

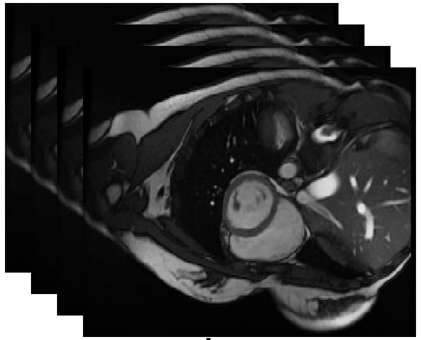


$1 \times 32 \times 224 \times 224$
$1 \times 32 \times 224 \times 224$
$1 \times 3x32 \times 224 \times 224$
Conv2D
CNN Layer 1
$1 \times 16\times 32 \times 224 \times 224$
Conv3D
$1 \times 16 \times 32 \times 224 \times 224$
$1 \times 16 \times 32 \times 224 \times 224$
Maxpool
CNN Layer 2
$1 \times 32 \times 16 \times 112 \times 112$
Conv3D
Upsampling
$1 \times 32 \times 16 \times 112 \times 112$
$1 \times 16\times 16 \times 112 \times 112$
$1 \times 32 \times 16 \times 112 \times 112$
Maxpool
CNN Layer 3
$1 \times 64 \times 8 \times 56 \times 56$
Conv3D
Upsampling
$1 \times 64 \times 8 \times 56 \times 56$
$1 \times 32 \times 8 \times 56 \times 56$
Maxpool
CNN Layer 4
Upsampling
$1 \times 64 \times 8 \times 56 \times 56$
$1 \times 64 \times 4 \times 28 \times 28$
$1 \times 128 \times 4 \times 28 \times 28$




```
inp=torch.rand(1, 2,32,224,224)
c1=torch.nn.Conv3d(3,16,3,padding=1)
out=c1(inp)
print(out.shape)
```

$\mathbf{1 \times 1 \times 132 \times 224 \times 224}$



CNN Layer 1

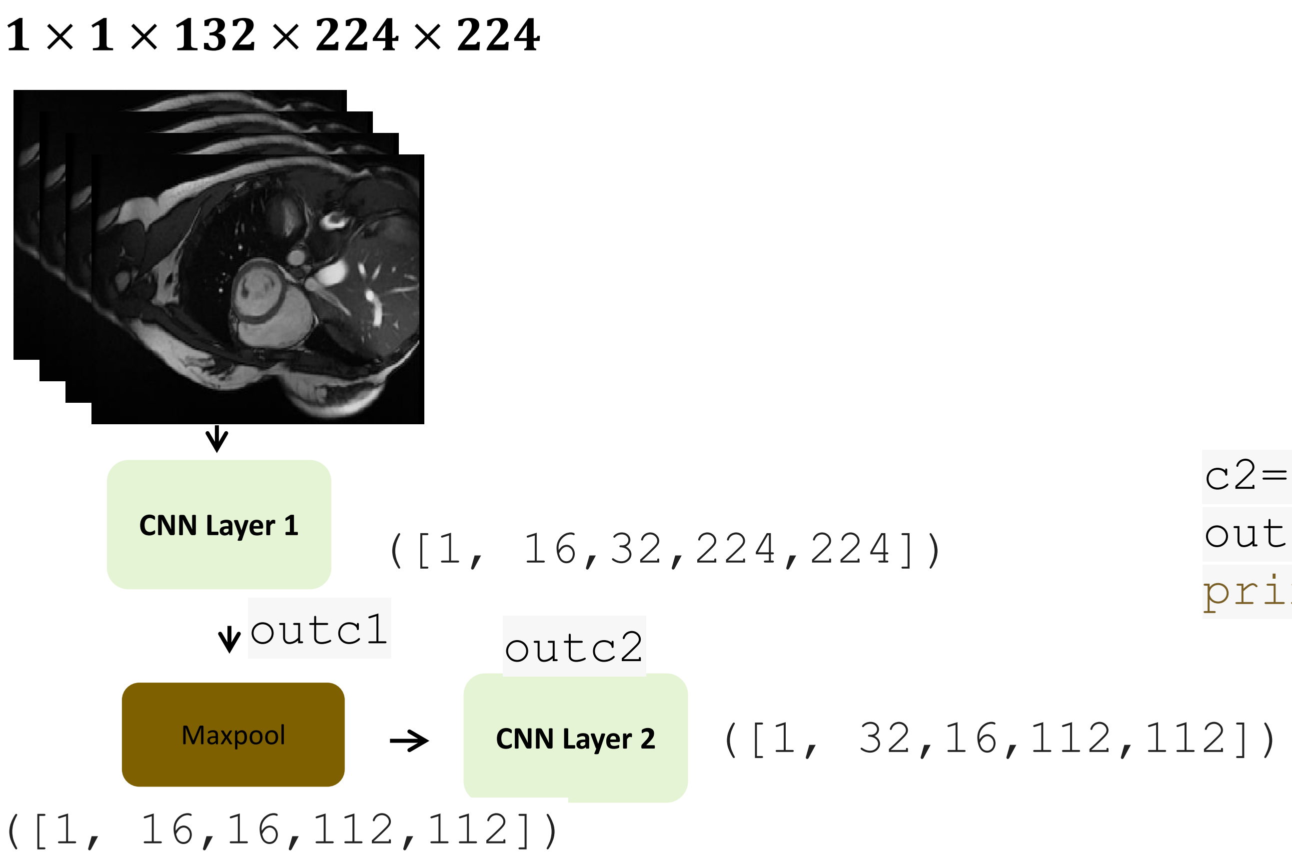
([1, 16,32,224,224])

`outc1`

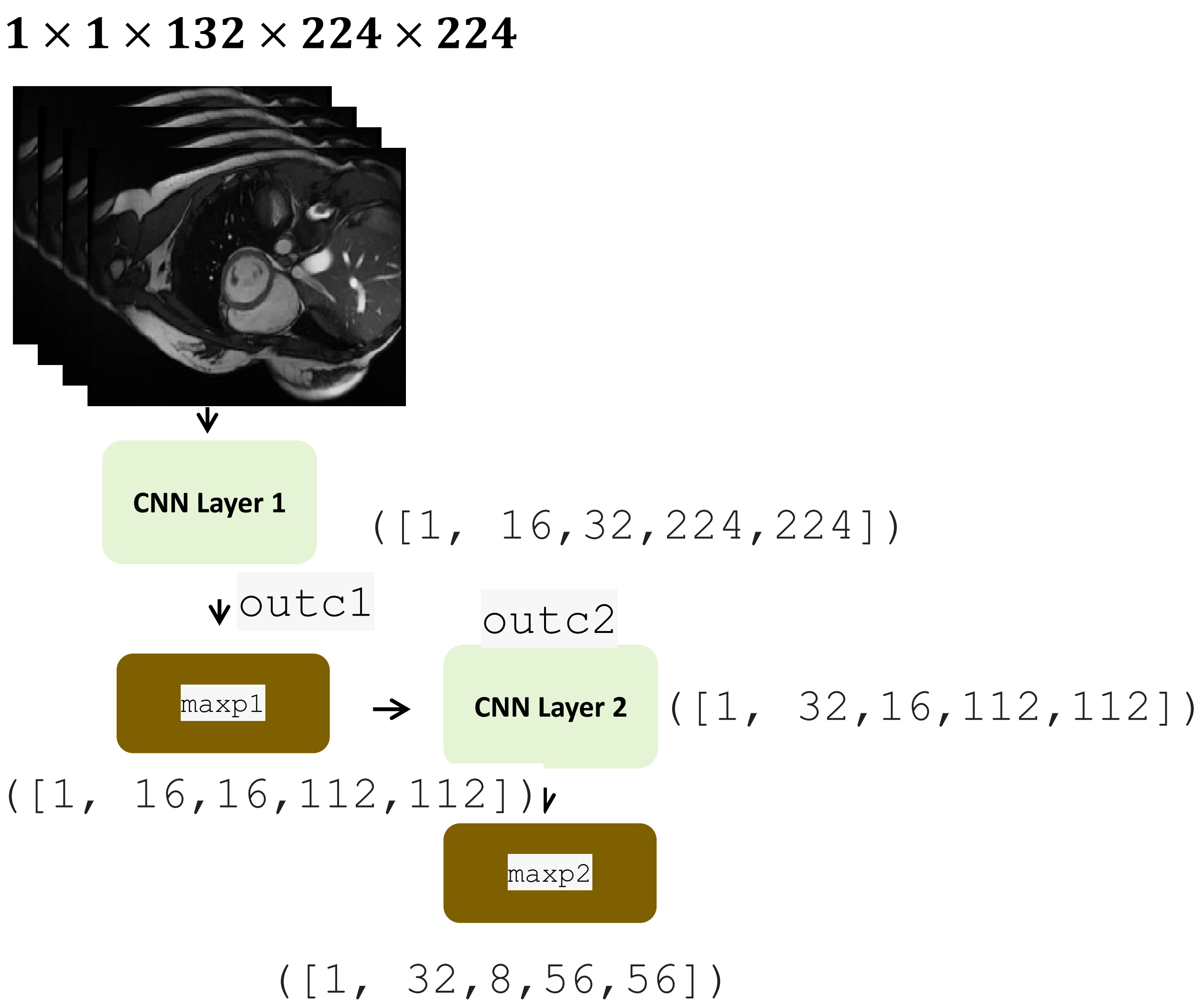
Maxpool

([1, 16,16,112,112])

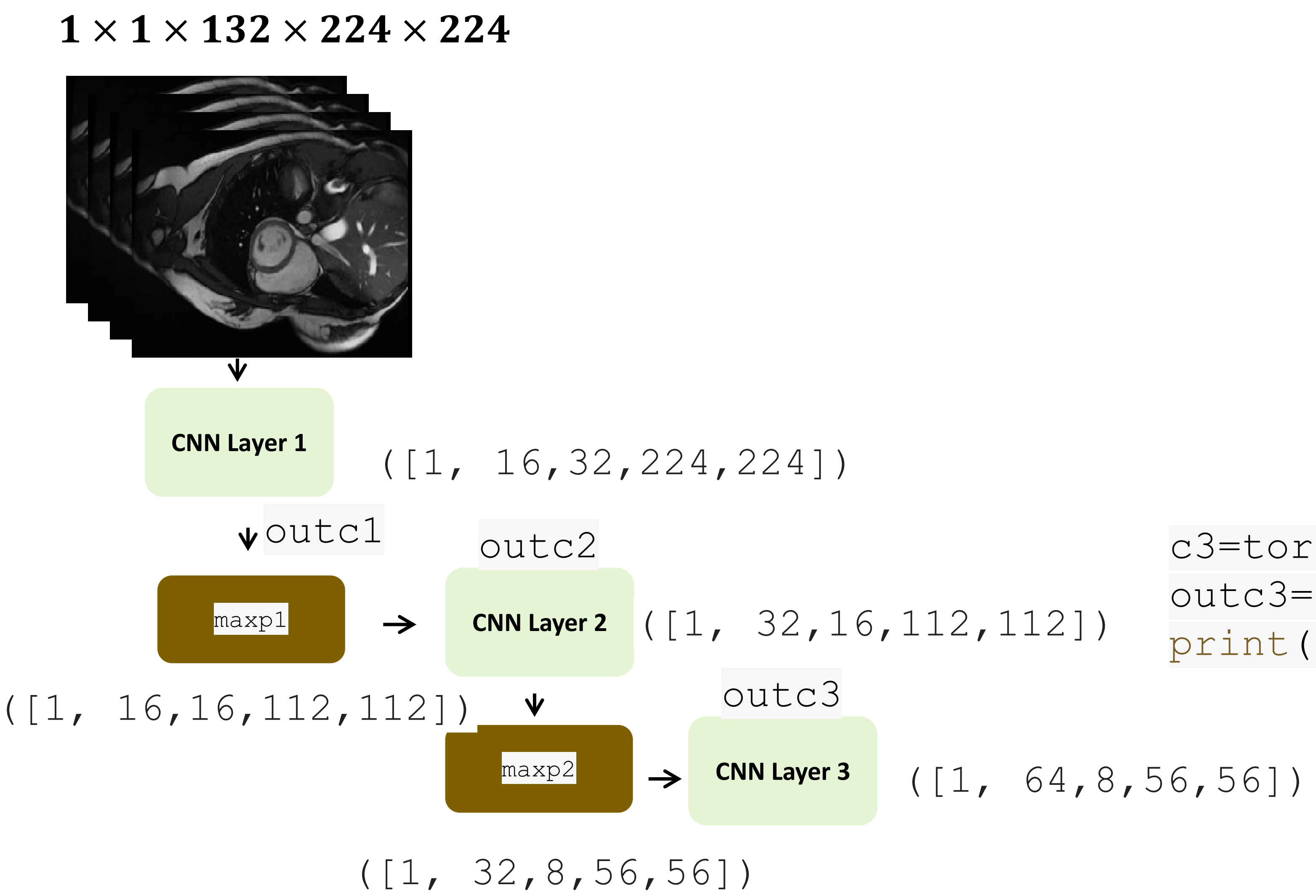
```
maxp1=torch.nn.MaxPool3d(2,2)
p1=maxp1(outc1)
print(p1.shape)
```



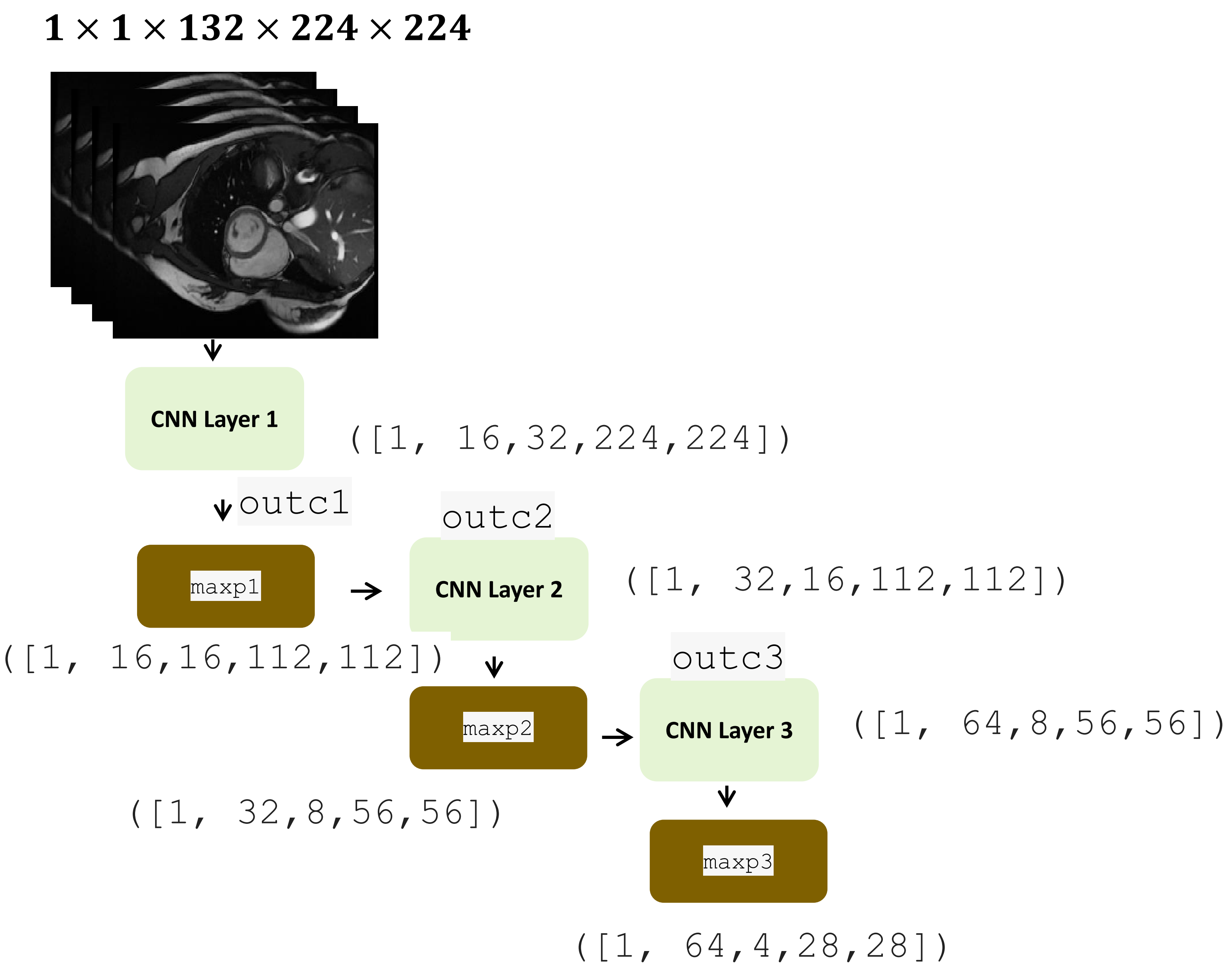

```
c2=torch.nn.Conv3d(16,32,3,padding=1)
outc2=c2(p1)
print(outc2.shape)
```



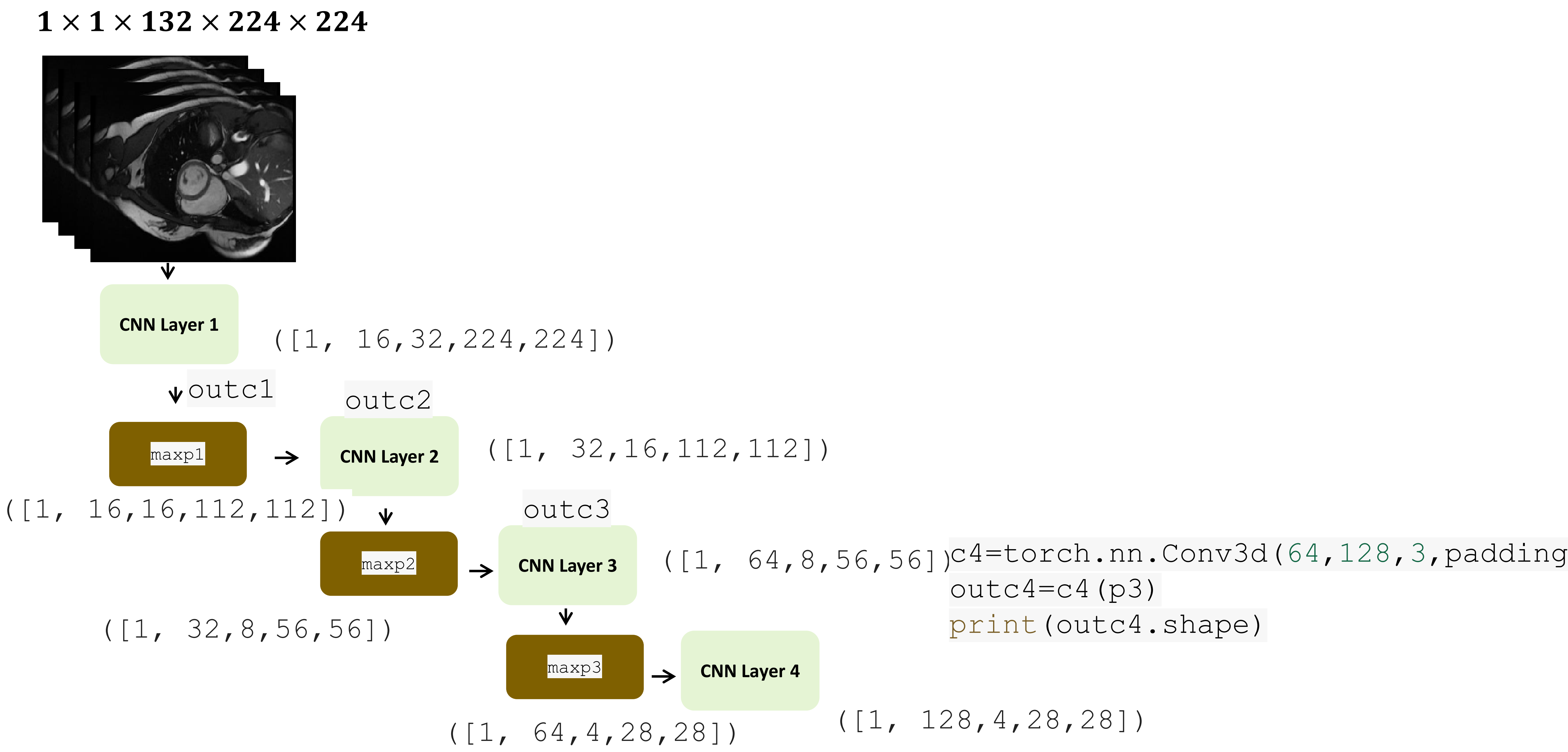

```
maxp2=torch.nn.MaxPool3d(2,2)
p2=maxp2(outc2)
print(p2.shape)
```



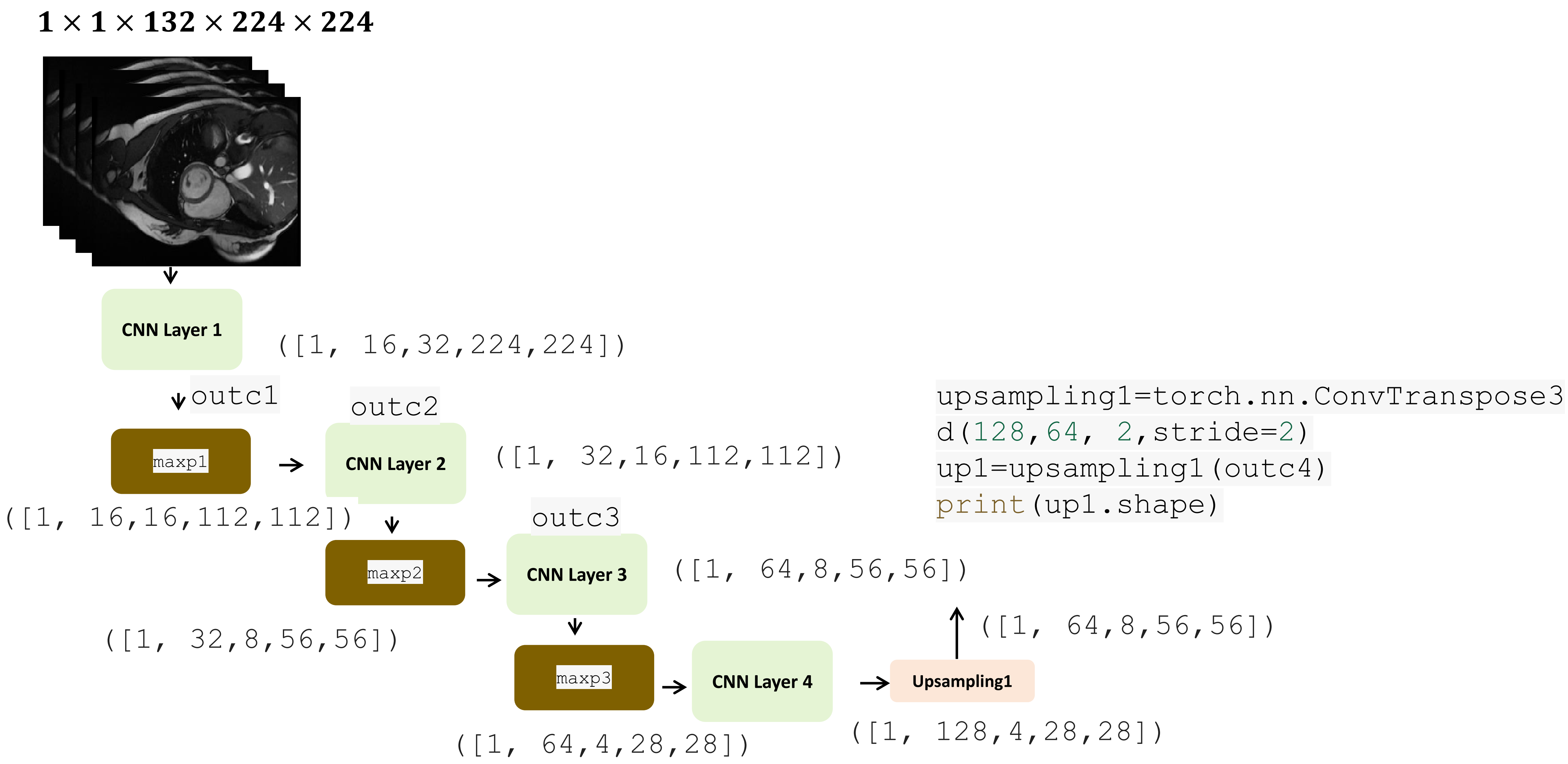

```
c3=torch.nn.Conv3d(32,64,3,padding=1)
outc3=c3(p2)
print(outc3.shape)
```



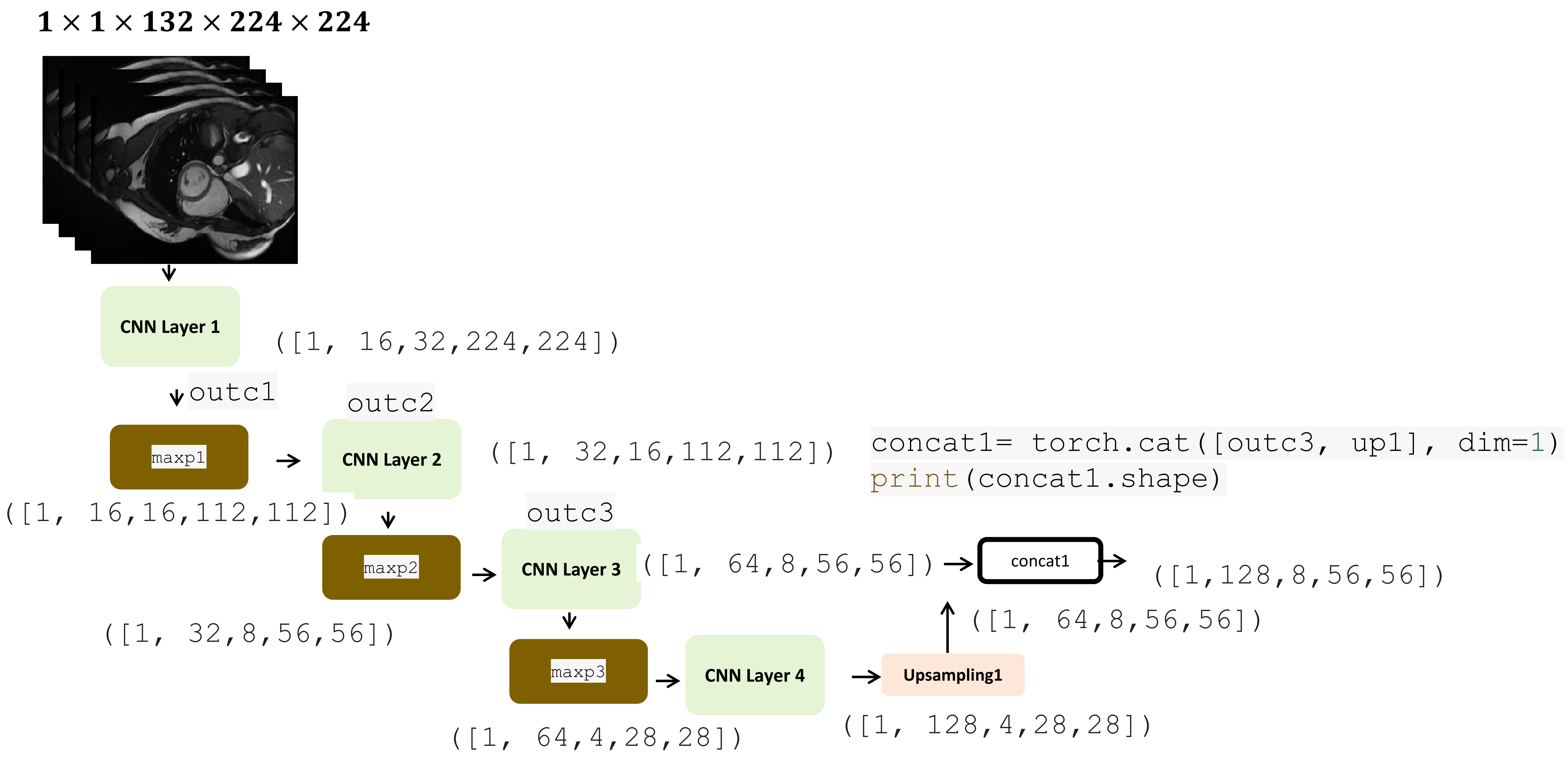

```
maxp3=torch.nn.MaxPool3d(2,2)
p3=maxp3(outc3)
print(p3.shape)
```



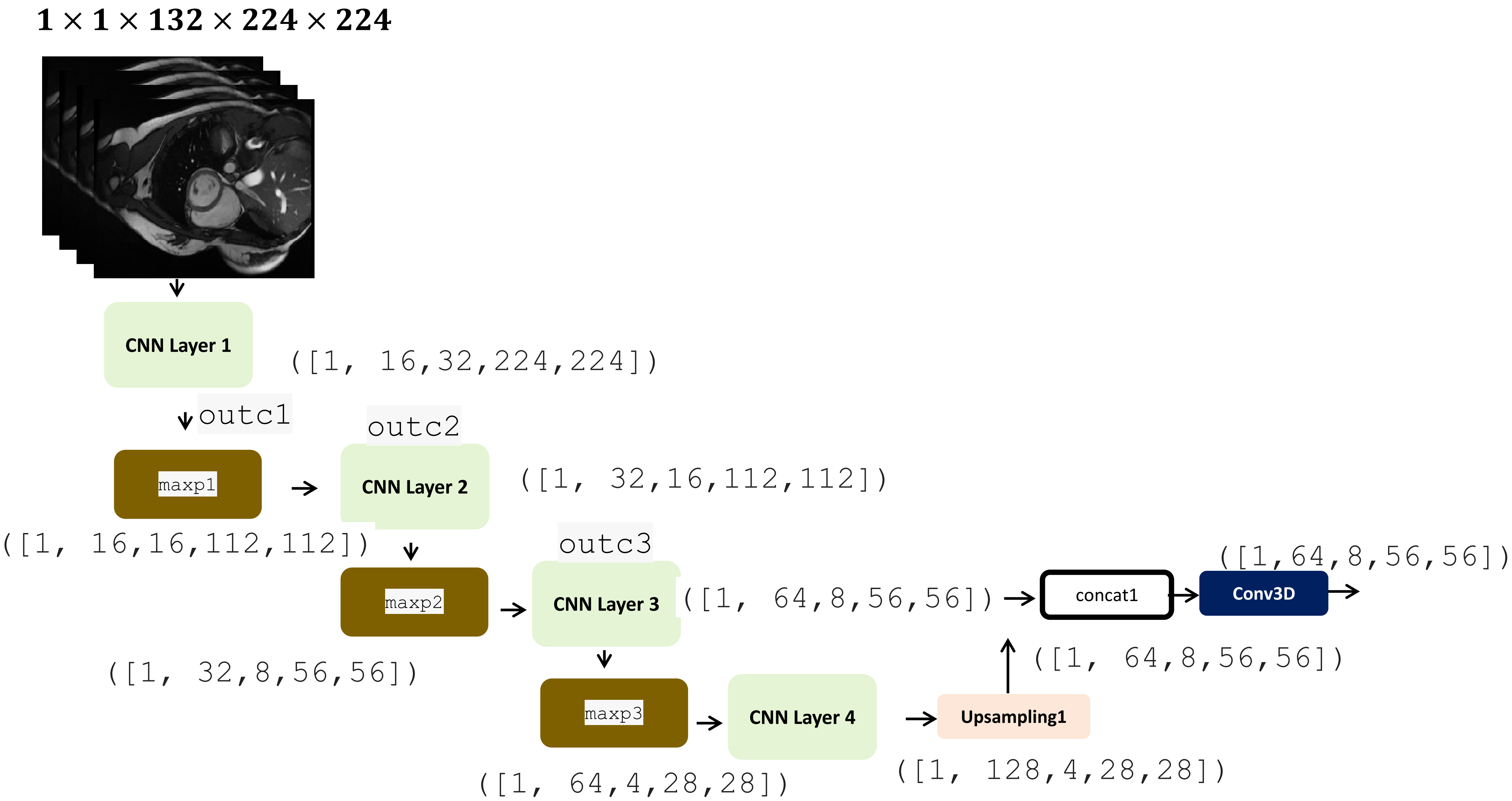
1 × 1 × 132 × 224 × 224
CNN Layer 1
([1, 16,32,224,224])
outc1
maxp1
([1, 16,16,112,112])
outc2
CNN Layer 2
([1, 32,16,112,112])
maxp2
([1, 32,8,56,56])
outc3
CNN Layer 3
([1, 64,8,56,56])
c4=torch.nn.Conv3d(64,128,3,padding
outc4=c4(p3)
print(outc4.shape)
maxp3
([1, 64,4,28,28])
CNN Layer 4
([1, 128,4,28,28])



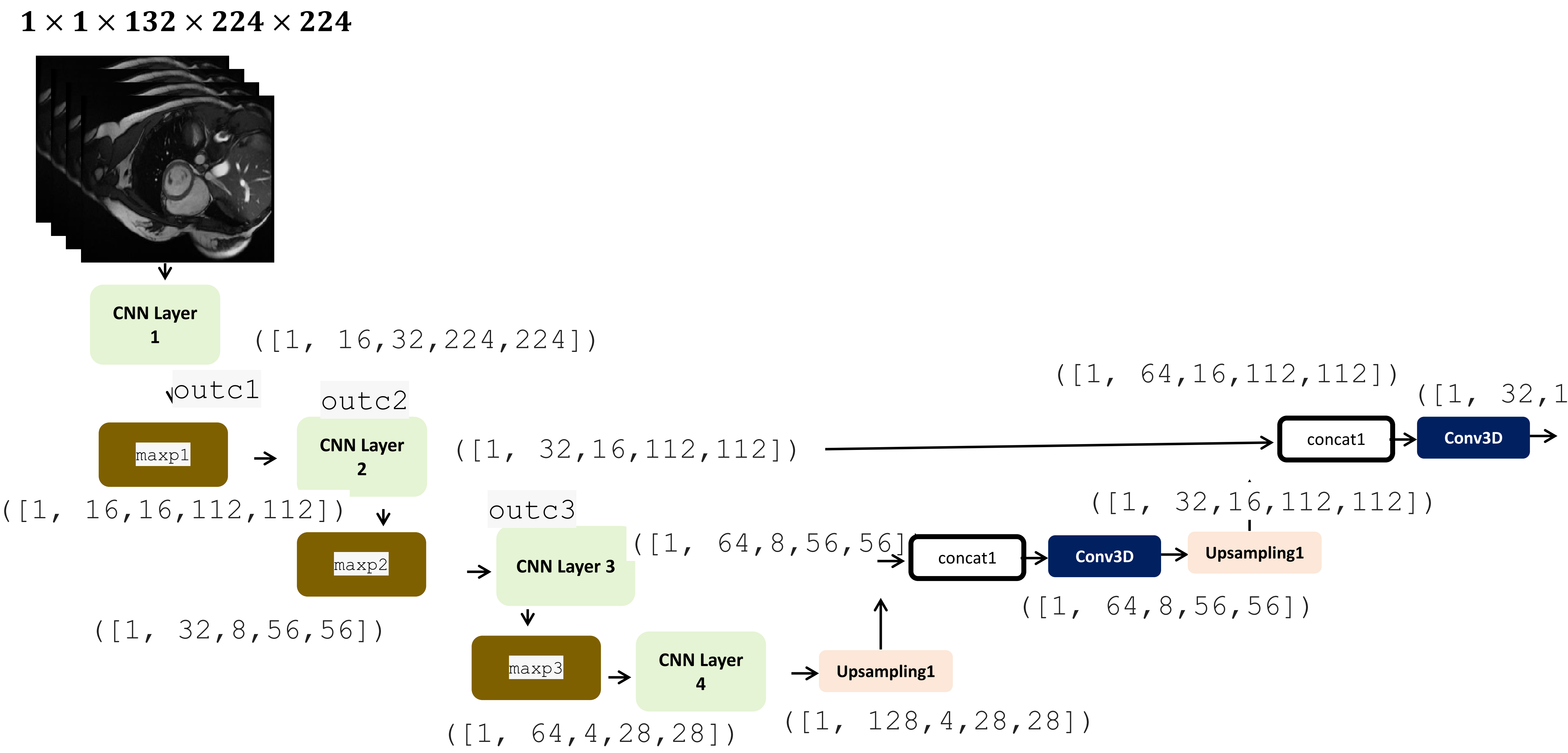
1 × 1 × 132 × 224 × 224
CNN Layer 1
([1, 16,32,224,224])
outc1
maxp1
([1, 16,16,112,112])
outc2
CNN Layer 2
([1, 32,16,112,112])
maxp2
([1, 32,8,56,56])
outc3
CNN Layer 3
([1, 64,8,56,56])
maxp3
([1, 64,4,28,28])
CNN Layer 4
([1, 128,4,28,28])
Upsampling1
([1, 64,8,56,56])
upsampling1=torch.nn.ConvTranspose3d(128,64, 2,stride=2)
up1=upsampling1(outc4)
print(up1.shape)

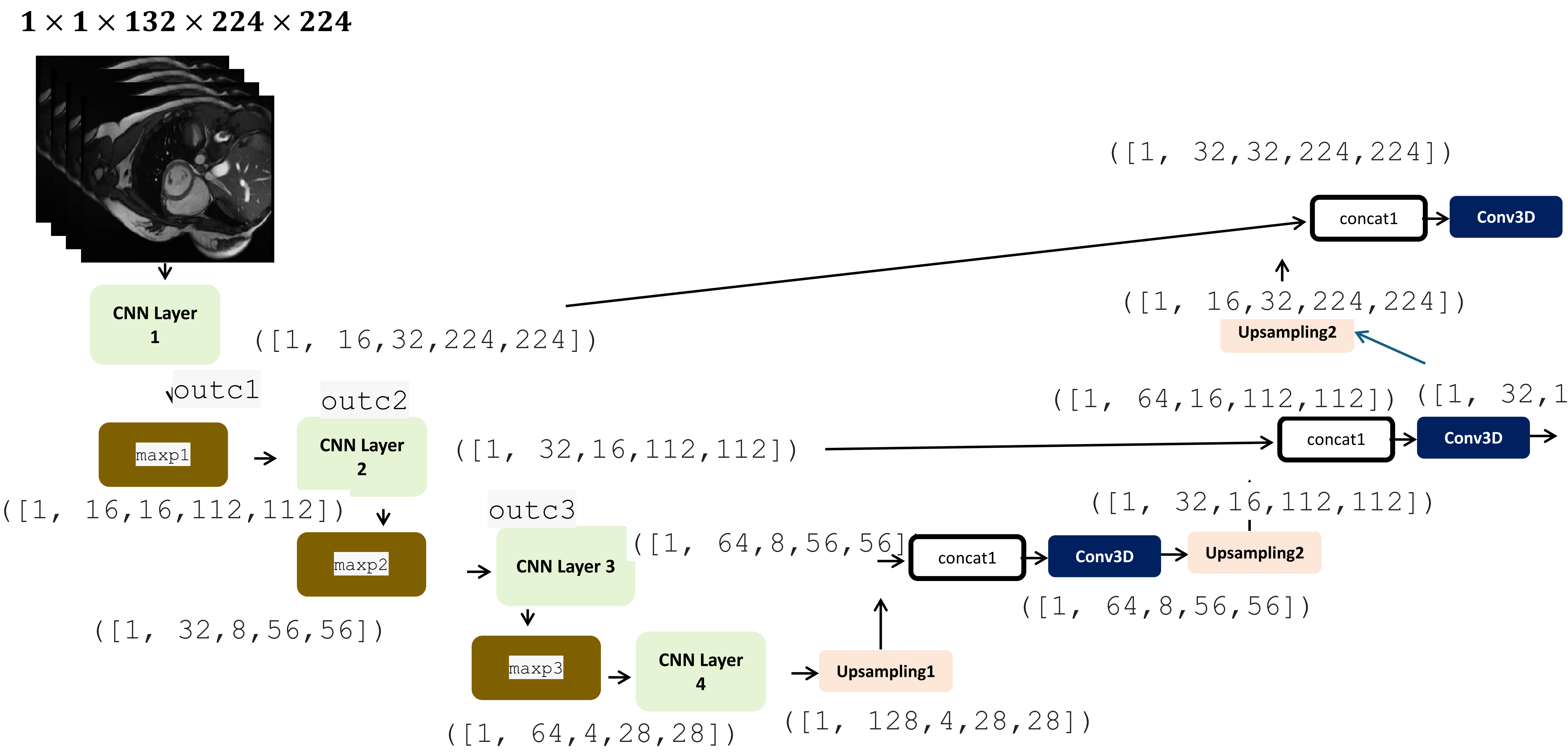


1 × 1 × 132 × 224 × 224
CNN Layer 1
([1, 16,32,224,224])
outc1
maxp1
outc2
CNN Layer 2
([1, 32,16,112,112])
concat1= torch.cat([outc3, up1], dim=1)
print(concat1.shape)
([1, 16,16,112,112])
outc3
maxp2
CNN Layer 3
([1, 64,8,56,56])
concat1
([1,128,8,56,56])
([1, 64,8,56,56])
([1, 32,8,56,56])
maxp3
CNN Layer 4
Upsampling1
([1, 64,4,28,28])
([1, 128,4,28,28])

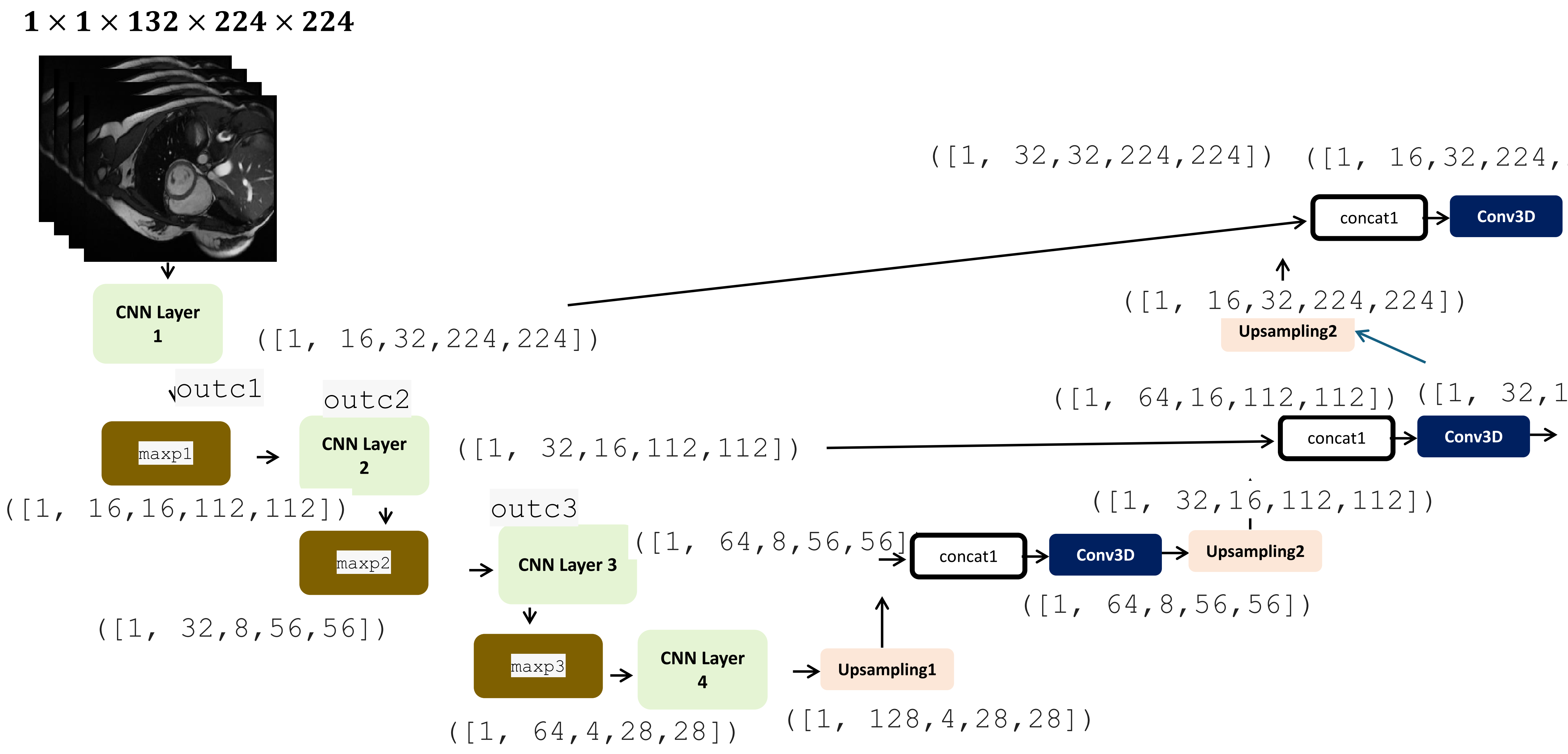


$1 \times 1 \times 132 \times 224 \times 224$
CNN Layer 1
([1, 16,32,224,224])
outc1
maxp1
outc2
CNN Layer 2
([1, 32,16,112,112])
([1, 16,16,112,112])
outc3
maxp2
CNN Layer 3
([1, 64,8,56,56])
concat1
([1,64,8,56,56])
Conv3D
([1, 32,8,56,56])
([1, 64,8,56,56])
maxp3
CNN Layer 4
Upsampling1
([1, 64,4,28,28])
([1, 128,4,28,28])

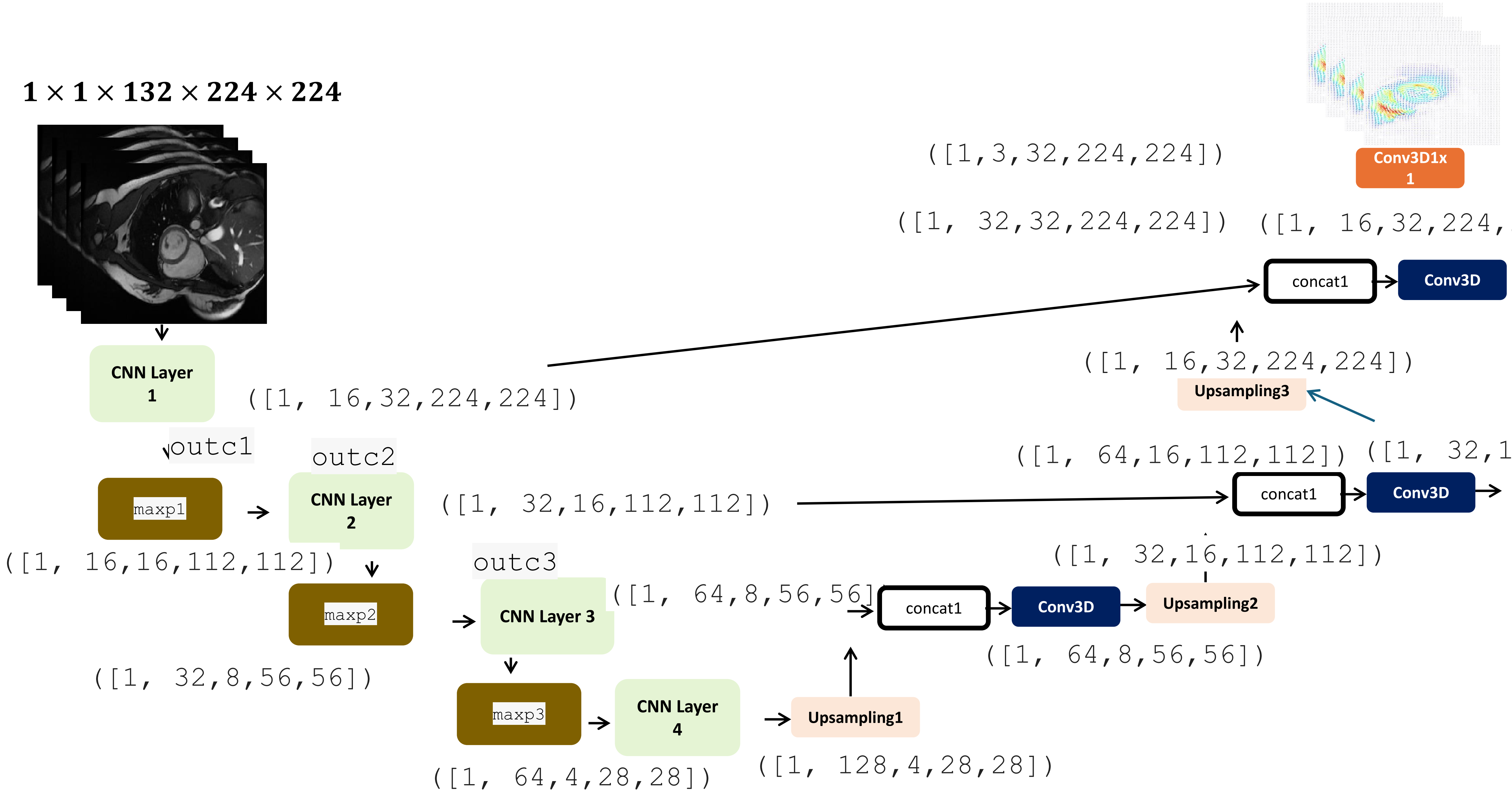

1 × 1 × 132 × 224 × 224
CNN Layer 1
([1, 16,32,224,224])
outc1
maxp1
([1, 16,16,112,112])
outc2
CNN Layer 2
([1, 32,16,112,112])
maxp2
([1, 32,8,56,56])
outc3
CNN Layer 3
([1, 64,8,56,56])
maxp3
([1, 64,4,28,28])
CNN Layer 4
Upsampling1
([1, 128,4,28,28])
concat1
Conv3D
Upsampling1
([1, 64,8,56,56])
([1, 32,16,112,112])
([1, 64,16,112,112])
concat1
Conv3D
([1, 32,1



1 × 1 × 132 × 224 × 224
CNN Layer 1
([1, 16,32,224,224])
outc1
maxp1
([1, 16,16,112,112])
outc2
CNN Layer 2
([1, 32,16,112,112])
maxp2
([1, 32,8,56,56])
outc3
CNN Layer 3
([1, 64,8,56,56])
maxp3
([1, 64,4,28,28])
CNN Layer 4
Upsampling1
([1, 128,4,28,28])
concat1
Conv3D
Upsampling2
([1, 64,8,56,56])
([1, 32,16,112,112])
([1, 64,16,112,112])
concat1
Conv3D
([1, 32,1
Upsampling2
([1, 16,32,224,224])
concat1
Conv3D
([1, 32,32,224,224])



$1 \times 1 \times 132 \times 224 \times 224$
CNN Layer 1
([1, 16,32,224,224])
outc1
maxp1
([1, 16,16,112,112])
outc2
CNN Layer 2
([1, 32,16,112,112])
maxp2
([1, 32,8,56,56])
outc3
CNN Layer 3
([1, 64,8,56,56])
maxp3
([1, 64,4,28,28])
CNN Layer 4
Upsampling1
([1, 128,4,28,28])
concat1
Conv3D
Upsampling2
([1, 64,8,56,56])
([1, 32,16,112,112])
([1, 64,16,112,112])
concat1
Conv3D
([1, 32,1
Upsampling2
([1, 16,32,224,224])
([1, 32,32,224,224])
concat1
Conv3D
([1, 16,32,224,



1 × 1 × 132 × 224 × 224
CNN Layer 1
outc1
maxp1
([1, 16,16,112,112])
CNN Layer 2
outc2
maxp2
([1, 32,8,56,56])
outc3
CNN Layer 3
maxp3
([1, 64,4,28,28])
CNN Layer 4
Upsampling1
([1, 128,4,28,28])
([1, 16,32,224,224])
([1, 32,16,112,112])
([1, 64,8,56,56])
concat1
Conv3D
Upsampling2
([1, 64,8,56,56])
([1, 32,16,112,112])
([1, 64,16,112,112])
concat1
Conv3D
Upsampling3
([1, 16,32,224,224])
concat1
Conv3D
([1, 32,32,224,224])
([1,3,32,224,224])
Conv3D1x1