Setup

In [ ]:
```
import h5py
import os
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.ticker import ScalarFormatter
import matplotlib.animation as animation
from mpl_toolkits.mplot3d import Axes3D
from scipy.linalg import svd
import pickle
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
from torch.utils.data import Dataset, DataLoader
from einops import rearrange
```

In [ ]:
```
!pip install torchvision
```

```
Requirement already satisfied: torchvision in /usr/local/lib/python3.11/dist
-packages (0.21.0+cu124)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packa
ges (from torchvision) (2.0.2)
Requirement already satisfied: torch==2.6.0 in /usr/local/lib/python3.11/dis
t-packages (from torchvision) (2.6.0+cu124)
Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in /usr/local/lib/pytho
n3.11/dist-packages (from torchvision) (11.2.1)
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-pa
ckages (from torch==2.6.0->torchvision) (3.18.0)
Requirement already satisfied: typing-extensions>=4.10.0 in /usr/local/lib/p
ython3.11/dist-packages (from torch==2.6.0->torchvision) (4.13.2)
Requirement already satisfied: networkx in /usr/local/lib/python3.11/dist-pa
ckages (from torch==2.6.0->torchvision) (3.4.2)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.11/dist-pack
ages (from torch==2.6.0->torchvision) (3.1.6)
Requirement already satisfied: fsspec in /usr/local/lib/python3.11/dist-pack
ages (from torch==2.6.0->torchvision) (2025.3.2)
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.4.127 in /usr/loca
l/lib/python3.11/dist-packages (from torch==2.6.0->torchvision) (12.4.127)
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.4.127 in /usr/lo
cal/lib/python3.11/dist-packages (from torch==2.6.0->torchvision) (12.4.127)
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.4.127 in /usr/loca
l/lib/python3.11/dist-packages (from torch==2.6.0->torchvision) (12.4.127)
Requirement already satisfied: nvidia-cudnn-cu12==9.1.0.70 in /usr/local/li
b/python3.11/dist-packages (from torch==2.6.0->torchvision) (9.1.0.70)
Requirement already satisfied: nvidia-cublas-cu12==12.4.5.8 in /usr/local/li
b/python3.11/dist-packages (from torch==2.6.0->torchvision) (12.4.5.8)
Requirement already satisfied: nvidia-cufft-cu12==11.2.1.3 in /usr/local/li
b/python3.11/dist-packages (from torch==2.6.0->torchvision) (11.2.1.3)
Requirement already satisfied: nvidia-curand-cu12==10.3.5.147 in /usr/local/
lib/python3.11/dist-packages (from torch==2.6.0->torchvision) (10.3.5.147)
Requirement already satisfied: nvidia-cusolver-cu12==11.6.1.9 in /usr/local/
lib/python3.11/dist-packages (from torch==2.6.0->torchvision) (11.6.1.9)
Requirement already satisfied: nvidia-cusparse-cu12==12.3.1.170 in /usr/loca
l/lib/python3.11/dist-packages (from torch==2.6.0->torchvision) (12.3.1.170)
Requirement already satisfied: nvidia-cusparselt-cu12==0.6.2 in /usr/local/l
ib/python3.11/dist-packages (from torch==2.6.0->torchvision) (0.6.2)
Requirement already satisfied: nvidia-nccl-cu12==2.21.5 in /usr/local/lib/py
thon3.11/dist-packages (from torch==2.6.0->torchvision) (2.21.5)
Requirement already satisfied: nvidia-nvtx-cu12==12.4.127 in /usr/local/lib/
python3.11/dist-packages (from torch==2.6.0->torchvision) (12.4.127)
Requirement already satisfied: nvidia-nvjitlink-cu12==12.4.127 in /usr/loca
l/lib/python3.11/dist-packages (from torch==2.6.0->torchvision) (12.4.127)
Requirement already satisfied: triton==3.2.0 in /usr/local/lib/python3.11/di
st-packages (from torch==2.6.0->torchvision) (3.2.0)
Requirement already satisfied: sympy==1.13.1 in /usr/local/lib/python3.11/di
st-packages (from torch==2.6.0->torchvision) (1.13.1)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.
11/dist-packages (from sympy==1.13.1->torch==2.6.0->torchvision) (1.3.0)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.11/
dist-packages (from jinja2->torch==2.6.0->torchvision) (3.0.2)
```

In [ ]:
```
from google.colab import auth
auth.authenticate_user()
```

In [ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

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

Dataset

Testing (Evaluation)

In [ ]:
```
def get_eval_data(scale=4, patch_size=64):
  filepath = "/content/drive/MyDrive/CAKRes/DatasetRe16k/test/nskt_Re16000.h
  file = h5py.File(filepath, 'r')
  dataset = file['fields']
  n_skip = 2
  n_size = int(2048/n_skip)
  t_size = 100

  display(dataset.shape)

  u = dataset[:, 0, ::n_skip, ::n_skip]
  v = dataset[:, 1, ::n_skip, ::n_skip]
  w = dataset[:, 2, ::n_skip, ::n_skip]

  reshaped_u = u.reshape(t_size, n_size * n_size).T
  reshaped_v = v.reshape(t_size, n_size * n_size).T
  reshaped_w = w.reshape(t_size, n_size * n_size).T

  # Mean and Fluctuations
  mean_u = np.mean(reshaped_u, axis=1).reshape(-1, 1)
  mean_v = np.mean(reshaped_v, axis=1).reshape(-1, 1)
  mean_w = np.mean(reshaped_w, axis=1).reshape(-1, 1)

  eval_dataset = FluidFlowDataset(u, v, w, scale=scale, patch_size=patch_siz

  return eval_dataset

eval_dataset = get_eval_data()
```

In [ ]:
```
eval_loader = DataLoader(eval_dataset, batch_size=64, shuffle=False)
```

Training

Here we just use 1000 sample to test the baseline model (due to limited resources as well)

In [ ]:
```
filepath = "/content/drive/MyDrive/CAKRes/DatasetRe16k/train/nskt_Re16000.h5

# Open the H5 file
file = h5py.File(filepath, 'r')
```

```
#inspect file structure
def print_structure(name, obj):
    if isinstance(obj, h5py.Group):
        print(f"Group: {name}")
    elif isinstance(obj, h5py.Dataset):
        print(f"Dataset: {name}, Shape: {obj.shape}, Type: {obj.dtype}")

file.visititems(print_structure)
```

```
Dataset: fields, Shape: (1000, 3, 2048, 2048), Type: float64
```

In [ ]:
```
dataset = file['fields']

t_skip = 5 #reduce time by this factor
n_skip = 2 #reduce u, v, and w grid by this factor
t_size = int(1000/t_skip)
n_size = int(2048/n_skip)


#load the dataset into a numpy array
u = dataset[::t_skip, 0, ::n_skip, ::n_skip]
v = dataset[::t_skip, 1, ::n_skip, ::n_skip]
w = dataset[::t_skip, 2, ::n_skip, ::n_skip]
```

In [ ]:
```
# Reshape the u,v,w data: (time, grid, grid) -> (grid x  grid, time)
reshaped_u = u.reshape(t_size, n_size * n_size).T
reshaped_v = v.reshape(t_size, n_size * n_size).T
reshaped_w = w.reshape(t_size, n_size * n_size).T

# Mean and Fluctuations
mean_u = np.mean(reshaped_u, axis=1).reshape(-1, 1)
mean_v = np.mean(reshaped_v, axis=1).reshape(-1, 1)
mean_w = np.mean(reshaped_w, axis=1).reshape(-1, 1)
```

In [ ]:
```
display(u.shape)
```

```
(200, 1024, 1024)
```

In [ ]:
```
!pip install basicsr
```

```
Collecting basicsr
  Downloading basicsr-1.4.2.tar.gz (172 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0.0/172.5 kB ? eta -:--:--
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╸ 163.8/172.5 kB 7.8 MB/s eta 0:0
0:01
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 172.5/172.5 kB 4.5 MB/s eta 0:0
0:00
  Preparing metadata (setup.py) ... done
Collecting addict (from basicsr)
  Downloading addict-2.4.0-py3-none-any.whl.metadata (1.0 kB)
Requirement already satisfied: future in /usr/local/lib/python3.11/dist-pack
ages (from basicsr) (1.0.0)
Collecting lmdb (from basicsr)
  Downloading lmdb-1.6.2-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86
_64.whl.metadata (1.1 kB)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.11/dist
-packages (from basicsr) (2.0.2)
Requirement already satisfied: opencv-python in /usr/local/lib/python3.11/di
st-packages (from basicsr) (4.11.0.86)
Requirement already satisfied: Pillow in /usr/local/lib/python3.11/dist-pack
ages (from basicsr) (11.2.1)
Requirement already satisfied: pyyaml in /usr/local/lib/python3.11/dist-pack
ages (from basicsr) (6.0.2)
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-pa
ckages (from basicsr) (2.32.3)
Requirement already satisfied: scikit-image in /usr/local/lib/python3.11/dis
t-packages (from basicsr) (0.25.2)
Requirement already satisfied: scipy in /usr/local/lib/python3.11/dist-packa
ges (from basicsr) (1.15.2)
Collecting tb-nightly (from basicsr)
  Downloading tb_nightly-2.20.0a20250428-py3-none-any.whl.metadata (1.9 kB)
Requirement already satisfied: torch>=1.7 in /usr/local/lib/python3.11/dist-
packages (from basicsr) (2.6.0+cu124)
Requirement already satisfied: torchvision in /usr/local/lib/python3.11/dist
-packages (from basicsr) (0.21.0+cu124)
Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-packag
es (from basicsr) (4.67.1)
Collecting yapf (from basicsr)
  Downloading yapf-0.43.0-py3-none-any.whl.metadata (46 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 46.8/46.8 kB 3.3 MB/s eta 0:0
0:00
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-pa
ckages (from torch>=1.7->basicsr) (3.18.0)
Requirement already satisfied: typing-extensions>=4.10.0 in /usr/local/lib/p
ython3.11/dist-packages (from torch>=1.7->basicsr) (4.13.2)
Requirement already satisfied: networkx in /usr/local/lib/python3.11/dist-pa
ckages (from torch>=1.7->basicsr) (3.4.2)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.11/dist-pack
ages (from torch>=1.7->basicsr) (3.1.6)
Requirement already satisfied: fsspec in /usr/local/lib/python3.11/dist-pack
ages (from torch>=1.7->basicsr) (2025.3.2)
Collecting nvidia-cuda-nvrtc-cu12==12.4.127 (from torch>=1.7->basicsr)
  Using cached nvidia_cuda_nvrtc_cu12-12.4.127-py3-none-manylinux2014_x86_6
4.whl.metadata (1.5 kB)
Collecting nvidia-cuda-runtime-cu12==12.4.127 (from torch>=1.7->basicsr)
  Using cached nvidia_cuda_runtime_cu12-12.4.127-py3-none-manylinux2014_x86_
```

```
64.whl.metadata (1.5 kB)
Collecting nvidia-cuda-cupti-cu12==12.4.127 (from torch>=1.7->basicsr)
  Using cached nvidia_cuda_cupti_cu12-12.4.127-py3-none-manylinux2014_x86_6
4.whl.metadata (1.6 kB)
Collecting nvidia-cudnn-cu12==9.1.0.70 (from torch>=1.7->basicsr)
  Using cached nvidia_cudnn_cu12-9.1.0.70-py3-none-manylinux2014_x86_64.whl.
metadata (1.6 kB)
Collecting nvidia-cublas-cu12==12.4.5.8 (from torch>=1.7->basicsr)
  Using cached nvidia_cublas_cu12-12.4.5.8-py3-none-manylinux2014_x86_64.wh
l.metadata (1.5 kB)
Collecting nvidia-cufft-cu12==11.2.1.3 (from torch>=1.7->basicsr)
  Using cached nvidia_cufft_cu12-11.2.1.3-py3-none-manylinux2014_x86_64.whl.
metadata (1.5 kB)
Collecting nvidia-curand-cu12==10.3.5.147 (from torch>=1.7->basicsr)
  Using cached nvidia_curand_cu12-10.3.5.147-py3-none-manylinux2014_x86_64.w
hl.metadata (1.5 kB)
Collecting nvidia-cusolver-cu12==11.6.1.9 (from torch>=1.7->basicsr)
  Using cached nvidia_cusolver_cu12-11.6.1.9-py3-none-manylinux2014_x86_64.w
hl.metadata (1.6 kB)
Collecting nvidia-cusparse-cu12==12.3.1.170 (from torch>=1.7->basicsr)
  Using cached nvidia_cusparse_cu12-12.3.1.170-py3-none-manylinux2014_x86_6
4.whl.metadata (1.6 kB)
Requirement already satisfied: nvidia-cusparselt-cu12==0.6.2 in /usr/local/l
ib/python3.11/dist-packages (from torch>=1.7->basicsr) (0.6.2)
Requirement already satisfied: nvidia-nccl-cu12==2.21.5 in /usr/local/lib/py
thon3.11/dist-packages (from torch>=1.7->basicsr) (2.21.5)
Requirement already satisfied: nvidia-nvtx-cu12==12.4.127 in /usr/local/lib/
python3.11/dist-packages (from torch>=1.7->basicsr) (12.4.127)
Collecting nvidia-nvjitlink-cu12==12.4.127 (from torch>=1.7->basicsr)
  Using cached nvidia_nvjitlink_cu12-12.4.127-py3-none-manylinux2014_x86_64.
whl.metadata (1.5 kB)
Requirement already satisfied: triton==3.2.0 in /usr/local/lib/python3.11/di
st-packages (from torch>=1.7->basicsr) (3.2.0)
Requirement already satisfied: sympy==1.13.1 in /usr/local/lib/python3.11/di
st-packages (from torch>=1.7->basicsr) (1.13.1)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.
11/dist-packages (from sympy==1.13.1->torch>=1.7->basicsr) (1.3.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/py
thon3.11/dist-packages (from requests->basicsr) (3.4.1)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dis
t-packages (from requests->basicsr) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.
11/dist-packages (from requests->basicsr) (2.4.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.
11/dist-packages (from requests->basicsr) (2025.1.31)
Requirement already satisfied: imageio!=2.35.0,>=2.33 in /usr/local/lib/pyth
on3.11/dist-packages (from scikit-image->basicsr) (2.37.0)
Requirement already satisfied: tifffile>=2022.8.12 in /usr/local/lib/python
3.11/dist-packages (from scikit-image->basicsr) (2025.3.30)
Requirement already satisfied: packaging>=21 in /usr/local/lib/python3.11/di
st-packages (from scikit-image->basicsr) (24.2)
Requirement already satisfied: lazy-loader>=0.4 in /usr/local/lib/python3.1
1/dist-packages (from scikit-image->basicsr) (0.4)
Requirement already satisfied: absl-py>=0.4 in /usr/local/lib/python3.11/dis
t-packages (from tb-nightly->basicsr) (1.4.0)
Requirement already satisfied: grpcio>=1.48.2 in /usr/local/lib/python3.11/d
```

```
ist-packages (from tb-nightly->basicsr) (1.71.0)
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.11/
dist-packages (from tb-nightly->basicsr) (3.8)
Requirement already satisfied: protobuf!=4.24.0,>=3.19.6 in /usr/local/lib/p
ython3.11/dist-packages (from tb-nightly->basicsr) (5.29.4)
Requirement already satisfied: setuptools>=41.0.0 in /usr/local/lib/python3.
11/dist-packages (from tb-nightly->basicsr) (75.2.0)
Requirement already satisfied: six>1.9 in /usr/local/lib/python3.11/dist-pac
kages (from tb-nightly->basicsr) (1.17.0)
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /us
r/local/lib/python3.11/dist-packages (from tb-nightly->basicsr) (0.7.2)
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.11/
dist-packages (from tb-nightly->basicsr) (3.1.3)
Requirement already satisfied: platformdirs>=3.5.1 in /usr/local/lib/python
3.11/dist-packages (from yapf->basicsr) (4.3.7)
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.1
1/dist-packages (from werkzeug>=1.0.1->tb-nightly->basicsr) (3.0.2)
Using cached nvidia_cublas_cu12-12.4.5.8-py3-none-manylinux2014_x86_64.whl
(363.4 MB)
Using cached nvidia_cuda_cupti_cu12-12.4.127-py3-none-manylinux2014_x86_64.w
hl (13.8 MB)
Using cached nvidia_cuda_nvrtc_cu12-12.4.127-py3-none-manylinux2014_x86_64.w
hl (24.6 MB)
Using cached nvidia_cuda_runtime_cu12-12.4.127-py3-none-manylinux2014_x86_6
4.whl (883 kB)
Using cached nvidia_cudnn_cu12-9.1.0.70-py3-none-manylinux2014_x86_64.whl (6
64.8 MB)
Using cached nvidia_cufft_cu12-11.2.1.3-py3-none-manylinux2014_x86_64.whl (2
11.5 MB)
Using cached nvidia_curand_cu12-10.3.5.147-py3-none-manylinux2014_x86_64.whl
(56.3 MB)
Using cached nvidia_cusolver_cu12-11.6.1.9-py3-none-manylinux2014_x86_64.whl
(127.9 MB)
Using cached nvidia_cusparse_cu12-12.3.1.170-py3-none-manylinux2014_x86_64.w
hl (207.5 MB)
Using cached nvidia_nvjitlink_cu12-12.4.127-py3-none-manylinux2014_x86_64.wh
l (21.1 MB)
Downloading addict-2.4.0-py3-none-any.whl (3.8 kB)
Downloading lmdb-1.6.2-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_6
4.whl (297 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 297.8/297.8 kB 16.3 MB/s eta 0:0
0:00
Downloading tb_nightly-2.20.0a20250428-py3-none-any.whl (5.5 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 5.5/5.5 MB 56.9 MB/s eta 0:00:00
Downloading yapf-0.43.0-py3-none-any.whl (256 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 256.2/256.2 kB 14.6 MB/s eta 0:0
0:00
Building wheels for collected packages: basicsr
  Building wheel for basicsr (setup.py) ... done
  Created wheel for basicsr: filename=basicsr-1.4.2-py3-none-any.whl size=21
4817 sha256=b32ad54c3fa8690b5b054076c71e91fc9bf25346843963a3c1f22341bd16096d
  Stored in directory: /root/.cache/pip/wheels/6d/a4/b3/9f888ba88efcae6dd4bb
ce69832363de9c4051142674f779fa
Successfully built basicsr
Installing collected packages: lmdb, addict, yapf, nvidia-nvjitlink-cu12, nv
idia-curand-cu12, nvidia-cufft-cu12, nvidia-cuda-runtime-cu12, nvidia-cuda-n
```

```
vrtc-cu12, nvidia-cuda-cupti-cu12, nvidia-cublas-cu12, tb-nightly, nvidia-cu
sparse-cu12, nvidia-cudnn-cu12, nvidia-cusolver-cu12, basicsr
  Attempting uninstall: nvidia-nvjitlink-cu12
    Found existing installation: nvidia-nvjitlink-cu12 12.5.82
    Uninstalling nvidia-nvjitlink-cu12-12.5.82:
      Successfully uninstalled nvidia-nvjitlink-cu12-12.5.82
  Attempting uninstall: nvidia-curand-cu12
    Found existing installation: nvidia-curand-cu12 10.3.6.82
    Uninstalling nvidia-curand-cu12-10.3.6.82:
      Successfully uninstalled nvidia-curand-cu12-10.3.6.82
  Attempting uninstall: nvidia-cufft-cu12
    Found existing installation: nvidia-cufft-cu12 11.2.3.61
    Uninstalling nvidia-cufft-cu12-11.2.3.61:
      Successfully uninstalled nvidia-cufft-cu12-11.2.3.61
  Attempting uninstall: nvidia-cuda-runtime-cu12
    Found existing installation: nvidia-cuda-runtime-cu12 12.5.82
    Uninstalling nvidia-cuda-runtime-cu12-12.5.82:
      Successfully uninstalled nvidia-cuda-runtime-cu12-12.5.82
  Attempting uninstall: nvidia-cuda-nvrtc-cu12
    Found existing installation: nvidia-cuda-nvrtc-cu12 12.5.82
    Uninstalling nvidia-cuda-nvrtc-cu12-12.5.82:
      Successfully uninstalled nvidia-cuda-nvrtc-cu12-12.5.82
  Attempting uninstall: nvidia-cuda-cupti-cu12
    Found existing installation: nvidia-cuda-cupti-cu12 12.5.82
    Uninstalling nvidia-cuda-cupti-cu12-12.5.82:
      Successfully uninstalled nvidia-cuda-cupti-cu12-12.5.82
  Attempting uninstall: nvidia-cublas-cu12
    Found existing installation: nvidia-cublas-cu12 12.5.3.2
    Uninstalling nvidia-cublas-cu12-12.5.3.2:
      Successfully uninstalled nvidia-cublas-cu12-12.5.3.2
  Attempting uninstall: nvidia-cusparse-cu12
    Found existing installation: nvidia-cusparse-cu12 12.5.1.3
    Uninstalling nvidia-cusparse-cu12-12.5.1.3:
      Successfully uninstalled nvidia-cusparse-cu12-12.5.1.3
  Attempting uninstall: nvidia-cudnn-cu12
    Found existing installation: nvidia-cudnn-cu12 9.3.0.75
    Uninstalling nvidia-cudnn-cu12-9.3.0.75:
      Successfully uninstalled nvidia-cudnn-cu12-9.3.0.75
  Attempting uninstall: nvidia-cusolver-cu12
    Found existing installation: nvidia-cusolver-cu12 11.6.3.83
    Uninstalling nvidia-cusolver-cu12-11.6.3.83:
      Successfully uninstalled nvidia-cusolver-cu12-11.6.3.83
Successfully installed addict-2.4.0 basicsr-1.4.2 lmdb-1.6.2 nvidia-cublas-c
u12-12.4.5.8 nvidia-cuda-cupti-cu12-12.4.127 nvidia-cuda-nvrtc-cu12-12.4.127
nvidia-cuda-runtime-cu12-12.4.127 nvidia-cudnn-cu12-9.1.0.70 nvidia-cufft-cu
12-11.2.1.3 nvidia-curand-cu12-10.3.5.147 nvidia-cusolver-cu12-11.6.1.9 nvid
ia-cusparse-cu12-12.3.1.170 nvidia-nvjitlink-cu12-12.4.127 tb-nightly-2.20.0
a20250428 yapf-0.43.0
```

In [ ]:
```python
from basicsr.archs.swinir_arch import SwinIR
from basicsr.data.transforms import paired_random_crop
from basicsr.utils.img_util import img2tensor, tensor2img
```

In [ ]:
```python
class FluidFlowDataset(Dataset):
    def __init__(self, u, v, w, scale=4, patch_size=64):
        self.u = u
```

```
        self.v = v
        self.w = w
        self.scale = scale
        self.patch_size = patch_size
        self.num_samples = u.shape[0]

    def __len__(self):
        return self.num_samples

    def __getitem__(self, idx):
        # Get HR image (combine all three channels)
        hr = np.stack([self.u[idx], self.v[idx], self.w[idx]], axis=-1)

        # Create LR by downsampling
        lr = hr[::self.scale, ::self.scale, :]

        # Random crop
        hr, lr = paired_random_crop(hr, lr, self.patch_size, self.scale)

        # Convert to tensor
        hr = img2tensor(hr, bgr2rgb=False, float32=True)
        lr = img2tensor(lr, bgr2rgb=False, float32=True)

        return {'lq': lr, 'gt': hr}
```

Loss function

In [ ]:
```
import math
```

In [ ]:
```
def _calculate_derivatives(tensor, kernel, padding_val, boundary_mode, spaci
    """Helper function to calculate derivatives using convolution."""
    if boundary_mode != 'zeros':
        pad = (padding_val, padding_val, padding_val, padding_val)
        tensor_padded = F.pad(tensor, pad, mode=boundary_mode)
        conv_padding = 0
    else:
        tensor_padded = tensor
        conv_padding = padding_val

    derivative = F.conv2d(tensor_padded, kernel, padding=conv_padding) / spa
    return derivative
```

In [ ]:
```
def divergence_loss(predicted_output, dx=2.0*math.pi/2048.0, dy=2.0*math.pi/
    """Calculates the divergence loss (||du/dx + dv/dy||)."""
    if predicted_output.shape[1] < 2:
        raise ValueError(f"Divergence loss requires at least 2 channels (u,
    u_pred = predicted_output[:, 0:1, :, :]
    v_pred = predicted_output[:, 1:2, :, :]

    kernel_dx = torch.tensor([[[[0., 0., 0.], [-0.5, 0., 0.5], [0., 0., 0.]]
    kernel_dy = torch.tensor([[[[0., -0.5, 0.], [0., 0., 0.], [0., 0.5, 0.]]
    padding_val = 1
```

```
    du_dx = _calculate_derivatives(u_pred, kernel_dx, padding_val, boundary_
    dv_dy = _calculate_derivatives(v_pred, kernel_dy, padding_val, boundary_

    divergence = du_dx + dv_dy

    if loss_type == 'l1':
        loss = torch.mean(torch.abs(divergence))
    elif loss_type == 'l2':
        loss = torch.mean(divergence**2)
    else:
        raise ValueError("divergence loss_type must be 'l1' or 'l2'")
    return loss
```

In [ ]:

```
def vorticity_consistency_loss(predicted_output, dx=2.0*math.pi/2048.0, dy=2
    """Calculates vorticity consistency loss (||w_pred - (dv/dx - du/dy)||).
    if predicted_output.shape[1] != 3:
        raise ValueError(f"Vorticity consistency loss requires 3 channels (u
    u_pred = predicted_output[:, 0:1, :, :]
    v_pred = predicted_output[:, 1:2, :, :]
    w_pred = predicted_output[:, 2:3, :, :]

    kernel_dx = torch.tensor([[[[0., 0., 0.], [-0.5, 0., 0.5], [0., 0., 0.]]
    kernel_dy = torch.tensor([[[[0., -0.5, 0.], [0., 0., 0.], [0., 0.5, 0.]]
    padding_val = 1

    dv_dx = _calculate_derivatives(v_pred, kernel_dx, padding_val, boundary_
    du_dy = _calculate_derivatives(u_pred, kernel_dy, padding_val, boundary_

    w_calculated = dv_dx - du_dy
    diff = w_pred - w_calculated

    if loss_type == 'l1':
        loss = torch.mean(torch.abs(diff))
    elif loss_type == 'l2':
        loss = torch.mean(diff**2)
    else:
        raise ValueError("vorticity loss_type must be 'l1' or 'l2'")
    return loss
```

In [ ]:

```
class L1DivergenceLoss(nn.Module):
    """
    Calculates a combined loss: standard L1 pixel-wise loss plus a
    physics-informed divergence loss.

    Loss = L1(predicted, target) + lambda_div * DivergenceLoss(predicted)
    """
    def __init__(self, lambda_div=0.01, dx=2.0*math.pi/2048.0, dy=2.0*math.p
        """
        Args:
            lambda_div (float): Weighting factor for the divergence loss ter
            dx (float): Grid spacing in the x-direction for divergence calcu
            dy (float): Grid spacing in the y-direction for divergence calcu
            div_loss_type (str): Type of norm ('l1' or 'l2') for divergence
            boundary_mode (str): Boundary mode ('zeros', 'reflect', 'replica
                                 for derivative calculation in divergence lo
```

```
        """
        super().__init__()
        self.lambda_div = lambda_div
        self.dx = dx
        self.dy = dy
        self.div_loss_type = div_loss_type
        self.boundary_mode = boundary_mode
        self.l1_loss = nn.L1Loss() # Standard pixel-wise L1 loss

    def forward(self, predicted_output, target_output):
        """
        Calculates the combined loss.

        Args:
            predicted_output (torch.Tensor): The output tensor from the netw
                                             Shape: (N, C, H, W). Assumes C=3
                                             or at least C=2 (u, v).
            target_output (torch.Tensor): The ground truth high-resolution t
                                          Shape: (N, C, H, W).

        Returns:
            torch.Tensor: A scalar tensor representing the combined loss.
        """
        # 1. Calculate standard L1 pixel-wise loss
        pixel_loss = self.l1_loss(predicted_output, target_output)

        # 2. Calculate the physics-informed divergence loss
        #    This only depends on the predicted velocities (u, v)
        div_loss = divergence_loss(predicted_output,
                                   dx=self.dx,
                                   dy=self.dy,
                                   loss_type=self.div_loss_type,
                                   boundary_mode=self.boundary_mode)

        # 3. Combine the losses
        total_loss = pixel_loss + self.lambda_div * div_loss

        return total_loss
```

L1 + Div Loss + Vort Loss

In [ ]:

```
class L1DivVortLoss(nn.Module):
    """
    Calculates a combined loss:
    L1 Loss + lambda_div * Divergence Loss + lambda_vort * Vorticity Consist
    """
    def __init__(self, lambda_div=0.01, lambda_vort=0.01,
                 dx=2.0*math.pi/2048.0, dy=2.0*math.pi/2048.0,
                 div_loss_type='l2', vort_loss_type='l2',
                 boundary_mode='zeros'):
        """
        Args:
            lambda_div (float): Weighting factor for the divergence loss ter
            lambda_vort (float): Weighting factor for the vorticity consiste
            dx (float): Grid spacing in the x-direction for physics losses.
```

```
            dy (float): Grid spacing in the y-direction for physics losses.
            div_loss_type (str): Type of norm ('l1' or 'l2') for divergence
            vort_loss_type (str): Type of norm ('l1' or 'l2') for vorticity
            boundary_mode (str): Boundary mode ('zeros', 'reflect', 'replica
                                 for derivative calculations.
        """
        super().__init__()
        self.lambda_div = lambda_div
        self.lambda_vort = lambda_vort
        self.dx = dx
        self.dy = dy
        self.div_loss_type = div_loss_type
        self.vort_loss_type = vort_loss_type
        self.boundary_mode = boundary_mode

        # Standard pixel-wise L1 loss component
        self.l1_loss = nn.L1Loss()

    def forward(self, predicted_output, target_output):
        """
        Calculates the combined loss.

        Args:
            predicted_output (torch.Tensor): The output tensor from the netw
                                             Shape: (N, 3, H, W) (u, v, w).
            target_output (torch.Tensor): The ground truth high-resolution t
                                          Shape: (N, 3, H, W).

        Returns:
            torch.Tensor: A scalar tensor representing the combined loss.
            dict: A dictionary containing the individual loss components (op
        """
        if predicted_output.shape[1] != 3:
             raise ValueError(f"Expected 3 channels (u, v, w) for predicted_
        if target_output.shape[1] != 3:
             raise ValueError(f"Expected 3 channels (u, v, w) for target_out
        if predicted_output.shape != target_output.shape:
             raise ValueError(f"Predicted shape {predicted_output.shape} mus


        # 1. Calculate standard L1 pixel-wise loss
        pixel_loss = self.l1_loss(predicted_output, target_output)

        # 2. Calculate the divergence loss (uses predicted u, v)
        div_loss = divergence_loss(predicted_output,
                                   dx=self.dx,
                                   dy=self.dy,
                                   loss_type=self.div_loss_type,
                                   boundary_mode=self.boundary_mode)

        # 3. Calculate the vorticity consistency loss (uses predicted u, v,
        vort_cons_loss = vorticity_consistency_loss(predicted_output,
                                                    dx=self.dx,
                                                    dy=self.dy,
                                                    loss_type=self.vort_loss_t
                                                    boundary_mode=self.boundar
```

```
        # 4. Combine the losses with weighting factors
        total_loss = pixel_loss + self.lambda_div * div_loss + self.lambda_v

        # Optional: Return individual components for logging/monitoring
        loss_components = {
            'total_loss': total_loss.item(),
            'l1_loss': pixel_loss.item(),
            'divergence_loss': div_loss.item(),
            'vorticity_consistency_loss': vort_cons_loss.item()
        }

        return total_loss
```

Evaluation Function

PSNR

In [ ]:

```
def psnr(target: torch.Tensor, prediction: torch.Tensor, data_range: float =
    """
    Calculates the Peak Signal-to-Noise Ratio (PSNR) between two tensors.

    Args:
        target (torch.Tensor): The ground truth tensor (e.g., high-resolutio
                               Shape: (N, C, H, W) or (C, H, W) or (H, W).
        prediction (torch.Tensor): The predicted tensor (e.g., super-resolve
                                   Must have the same shape as target.
        data_range (float, optional): The range of the input data (maximum v
                                      If None, it will be estimated from the
                                      as max(target) - min(target).
    Returns:
        float: The PSNR value in dB. Returns +inf if the prediction and targ
               Returns NaN if inputs are invalid.
    """
    if not isinstance(target, torch.Tensor) or not isinstance(prediction, to
        raise TypeError(f"Inputs must be torch.Tensor, got {type(target)} ar

    if target.shape != prediction.shape:
        raise ValueError(f"Input shapes must match, got {target.shape} and {

    # Calculate Mean Squared Error (MSE)
    # Reduce over all dimensions except batch (if present)
    reduce_dims = tuple(range(1, target.dim())) # Dims C, H, W if N, C, H, W
    if not reduce_dims: # Handle case of single value tensors (or just H, W)
        reduce_dims = None

    mse = torch.mean((target - prediction) ** 2, dim=reduce_dims)

    # Handle case where MSE is zero (perfect reconstruction)
    if torch.all(mse == 0):
        return float('inf')
```

```
    # Handle potential batch dimension (average MSE across batch if needed)
    if mse.dim() > 0 and mse.shape[0] > 1: # Check if it's a batch MSE tensc
        mse = torch.mean(mse)
    elif mse.dim() > 0: # Single element tensor
        mse = mse.item()
    else: # Scalar already
        mse = mse.item()

    if mse == 0: # Double check after potential averaging
         return float('inf')

    # Determine data range (MAX_I value)
    if data_range is None:
        # Estimate from target data. Be cautious: this might vary batch to b
        # It's often better to specify a fixed data_range based on dataset k
        max_val = torch.max(target)
        min_val = torch.min(target)
        _data_range = (max_val - min_val).item()
        if _data_range == 0: # Handle constant image case
             # Avoid division by zero if data range is zero and mse > 0
             # This scenario is unlikely with real data but possible
             return -10.0 * math.log10(mse) if mse > 0 else float('inf')
    else:
        _data_range = data_range

    # Calculate PSNR
    try:
        psnr_val = 10.0 * math.log10(_data_range**2 / mse)
    except ValueError:
        # Can happen if mse is negative (shouldn't be) or zero (already hanc
        return float('nan') # Indicate an issue

    return psnr_val
```

In [ ]:
```
def evaluate_psnr(model, dataloader, device):
    model.eval()
    total_psnr = 0.0
    with torch.no_grad():
        for batch in dataloader:
            lr = batch['lq'].to(device)
            hr = batch['gt'].to(device)

            outputs = model(lr)
            current_psnr = psnr(hr, outputs)
            total_psnr += current_psnr

    return total_psnr / len(dataloader)
```

SSIM

In [ ]:
```
import torch
import numpy as np
from skimage.metrics import structural_similarity
```

```python
# Evaluation function using SSIM
def evaluate_ssim(model, dataloader, device):
    model.eval()
    total_ssim = 0.0
    total_samples = 0

    with torch.no_grad():
        for batch in dataloader:
            # Move data to the specified device
            lr = batch['lq'].to(device)
            hr = batch['gt'].to(device) # Ground truth HR

            # Get model output
            outputs = model(lr) # Predicted HR

            # Move tensors to CPU to use with numpy
            hr_np = hr.cpu().numpy()
            outputs_np = outputs.cpu().numpy()

            if hr_np.ndim == 4 and hr_np.shape[1] == 3:
                hr_np = np.transpose(hr_np, (0, 2, 3, 1)) # -> NHWC
                outputs_np = np.transpose(outputs_np, (0, 2, 3, 1)) # -> NHW
                channel_axis_param = -1 # Channels are now the last axis
            else:
                 # Handle other cases if necessary (e.g., grayscale, differe
                 # If grayscale (N, 1, H, W), you might squeeze the channel
                 # and set multichannel=False, channel_axis=None
                 print("Warning: Unexpected data shape. Assuming non-standar
                 # Default behavior might be needed here based on your exact
                 channel_axis_param = None # Or adjust as needed


            # Iterate through samples in the batch
            batch_size = hr_np.shape[0]
            for i in range(batch_size):
                hr_sample = hr_np[i]
                output_sample = outputs_np[i]

                # --- SSIM Calculation ---
                # Determine the data range. If your data is normalized (e.g.
                # set data_range=1.0. Otherwise, calculate from the ground t
                # Using the actual range of the HR sample is often robust.
                data_range = hr_sample.max() - hr_sample.min()

                # Handle potential edge case where data_range is zero
                if data_range == 0:
                   # If HR is constant, SSIM is 1 if output matches, 0 other
                   if np.all(hr_sample == output_sample):
                       ssim_val = 1.0
                   else:
                       # Or handle as a special case, e.g., assign 0 or skip
                       ssim_val = 0.0 # Or handle as needed
                       # print(f"Warning: Zero data range encountered for sa
                else:
                    ssim_val = structural_similarity(
                        hr_sample,
```

```
                        output_sample,
                        multichannel=(channel_axis_param is not None), # Tru
                        data_range=data_range,
                        channel_axis=channel_axis_param # Specify channel ax
                    )

                total_ssim += ssim_val

            total_samples += batch_size

    # Calculate average SSIM over all samples
    average_ssim = total_ssim / total_samples if total_samples > 0 else 0
    return average_ssim
```

Scale = 4

Model Setup

!! Note: Patch size above 64 makes the Google Collab Runtime restart

In [ ]:
```
lambda_div = 0.00001
lambda_vort = 0.00001
patch_size = 64
```

In [ ]:
```
scale = 4 # Here we try with scaling of 4, next we try with 8 as well to obs
batch_size = 16
num_epochs = 50
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

# Create datasets
split_idx = int(0.8 * len(u))
train_dataset = FluidFlowDataset(u[:split_idx], v[:split_idx], w[:split_idx]
val_dataset = FluidFlowDataset(u[split_idx:], v[split_idx:], w[split_idx:],

# Create dataloaders
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True
val_loader = DataLoader(val_dataset, batch_size=batch_size, shuffle=False)

# The swinIR model initialization
model = SwinIR(upscale=scale,
               in_chans=3,
               img_size=patch_size,
               window_size=8,
               img_range=1.,
               depths=[6, 6, 6, 6],
               embed_dim=60,
               num_heads=[6, 6, 6, 6],
               mlp_ratio=2,
               upsampler='pixelshuffle',
               resi_connection='1conv').to(device)

# loss and optimizer setup
```

```
criterion = L1DivVortLoss(lambda_div=lambda_div, lambda_vort=lambda_vort, di
optimizer = optim.Adam(model.parameters(), lr=1e-4)
```

Train

In [ ]:

```
best_val_loss = float('inf')

train_losses = []
val_losses = []

model_save_name = f'best_swinir_model_{scale}_div{lambda_div}_vort{lambda_vo

for epoch in range(num_epochs):
    model.train()
    train_loss = 0.0

    for batch in train_loader:
        lr = batch['lq'].to(device)
        hr = batch['gt'].to(device)

        optimizer.zero_grad()

        # forward pass
        outputs = model(lr)
        loss = criterion(outputs, hr)

        # backward pass and optimize
        loss.backward()
        optimizer.step()

        train_loss += loss.item()

    # validation
    model.eval()
    val_loss = 0.0
    with torch.no_grad():
        for batch in val_loader:
            lr = batch['lq'].to(device)
            hr = batch['gt'].to(device)

            outputs = model(lr)
            loss = criterion(outputs, hr)
            val_loss += loss.item()

            outputs_np = outputs.detach().cpu().numpy()
            hr_np = hr.detach().cpu().numpy()



    # Print statistics
    train_loss /= len(train_loader)
    val_loss /= len(val_loader)
```

```
    train_losses.append(train_loss)
    val_losses.append(val_loss)

    print(f'Epoch [{epoch+1}/{num_epochs}], Train Loss: {train_loss:.4f}, Va

    # Save best model
    if val_loss < best_val_loss:
        best_val_loss = val_loss
        torch.save(model.state_dict(), model_save_name)

        # Store the model to Google Drive
        !cp {model_save_name} /content/drive/MyDrive/CAKRes/Kapildev-Working


# Load best model
model.load_state_dict(torch.load(model_save_name))
```

```
Epoch [1/50], Train Loss: 2.6025, Val Loss: 2.5769
Epoch [2/50], Train Loss: 2.3228, Val Loss: 2.4208
Epoch [3/50], Train Loss: 2.1617, Val Loss: 1.8134
Epoch [4/50], Train Loss: 1.5935, Val Loss: 1.3608
Epoch [5/50], Train Loss: 1.3329, Val Loss: 1.2230
Epoch [6/50], Train Loss: 1.2122, Val Loss: 1.0991
Epoch [7/50], Train Loss: 1.1144, Val Loss: 1.0778
Epoch [8/50], Train Loss: 1.0168, Val Loss: 1.0314
Epoch [9/50], Train Loss: 0.9603, Val Loss: 0.9553
Epoch [10/50], Train Loss: 0.9343, Val Loss: 0.8848
Epoch [11/50], Train Loss: 0.8708, Val Loss: 0.8446
Epoch [12/50], Train Loss: 0.8346, Val Loss: 0.8267
Epoch [13/50], Train Loss: 0.8376, Val Loss: 0.8518
Epoch [14/50], Train Loss: 0.8148, Val Loss: 0.7875
Epoch [15/50], Train Loss: 0.7785, Val Loss: 0.7392
Epoch [16/50], Train Loss: 0.7485, Val Loss: 0.7062
Epoch [17/50], Train Loss: 0.7240, Val Loss: 0.7212
Epoch [18/50], Train Loss: 0.7177, Val Loss: 0.7543
Epoch [19/50], Train Loss: 0.7151, Val Loss: 0.7116
Epoch [20/50], Train Loss: 0.7054, Val Loss: 0.6648
Epoch [21/50], Train Loss: 0.6782, Val Loss: 0.6480
Epoch [22/50], Train Loss: 0.6186, Val Loss: 0.5985
Epoch [23/50], Train Loss: 0.6461, Val Loss: 0.5862
Epoch [24/50], Train Loss: 0.6526, Val Loss: 0.6391
Epoch [25/50], Train Loss: 0.6382, Val Loss: 0.6235
Epoch [26/50], Train Loss: 0.6265, Val Loss: 0.6035
Epoch [27/50], Train Loss: 0.6155, Val Loss: 0.6190
Epoch [28/50], Train Loss: 0.6057, Val Loss: 0.5864
Epoch [29/50], Train Loss: 0.6230, Val Loss: 0.5642
Epoch [30/50], Train Loss: 0.6189, Val Loss: 0.5994
Epoch [31/50], Train Loss: 0.6058, Val Loss: 0.6068
Epoch [32/50], Train Loss: 0.5862, Val Loss: 0.5855
Epoch [33/50], Train Loss: 0.6167, Val Loss: 0.5803
Epoch [34/50], Train Loss: 0.6030, Val Loss: 0.5708
Epoch [35/50], Train Loss: 0.6136, Val Loss: 0.5823
Epoch [36/50], Train Loss: 0.6150, Val Loss: 0.6444
Epoch [37/50], Train Loss: 0.5550, Val Loss: 0.5782
Epoch [38/50], Train Loss: 0.5735, Val Loss: 0.5456
Epoch [39/50], Train Loss: 0.5783, Val Loss: 0.6031
Epoch [40/50], Train Loss: 0.5674, Val Loss: 0.5730
Epoch [41/50], Train Loss: 0.5493, Val Loss: 0.5551
Epoch [42/50], Train Loss: 0.5579, Val Loss: 0.5541
Epoch [43/50], Train Loss: 0.5852, Val Loss: 0.5121
Epoch [44/50], Train Loss: 0.5726, Val Loss: 0.5564
Epoch [45/50], Train Loss: 0.5402, Val Loss: 0.5497
Epoch [46/50], Train Loss: 0.5754, Val Loss: 0.5403
Epoch [47/50], Train Loss: 0.5457, Val Loss: 0.5337
Epoch [48/50], Train Loss: 0.5545, Val Loss: 0.5528
Epoch [49/50], Train Loss: 0.5566, Val Loss: 0.5643
Epoch [50/50], Train Loss: 0.5623, Val Loss: 0.5587
```

Out[ ]: <All keys matched successfully>

Convergence Curve

In [ ]:
```
plt.figure(figsize=(15, 5))

# Loss plot
plt.plot(train_losses, label='Train Loss')
plt.plot(val_losses, label='Val Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
```

Out[ ]:
```
<matplotlib.legend.Legend at 0x782922983410>
```

Evaluation

In [ ]:
```
model.load_state_dict(torch.load(f'best_swinir_model_{scale}_div{lambda_div}

# Evaluation function
def evaluate(model, dataloader):
    model.eval()
    total_loss = 0.0
    with torch.no_grad():
        for batch in dataloader:
            lr = batch['lq'].to(device)
            hr = batch['gt'].to(device)

            outputs = model(lr)
            loss = criterion(outputs, hr)
            total_loss += loss.item()

    return total_loss / len(dataloader)

# Evaluate on validation set
val_loss = evaluate(model, eval_loader)
print(f'Final Test Loss: {val_loss:.4f}')

# With no div loss: Final Validation Loss: 1.2181
# With div loss and coeff = 1: Final Validation Loss: 410.3817
# Now with coeff = 0.1: 44.1709
# Now with coeff = 0.01: 7.2212
# Use L1 criterion: 2.3576, MSE Loss: 22.6876
```

```
# Using div loss = 0, Final Validation Loss: 1.1393
# With div coeff = 0.01 and vort coeff = 0.01: 11.4714
# With div coeff = 0.00001 and vort loss coeff = 0.00001: 1.2598
# With scale = 2, div and loss coeff = 0.00001 : 0.8166
# With scale = 2, coeffs = 0: ___
# With all dataset, scale = 4, coeff = 0.00001 : 0.5137
# With train on 1000 dataset; eval on 100 test: 0.5247
```

```
Final Test Loss: 0.5471
```

Use PSNR

In [ ]:
```
model.load_state_dict(torch.load(f'best_swinir_model_{scale}_div{lambda_div}

val_psnr = evaluate_psnr(model, eval_loader, device)
print(f'Average Test PSNR: {val_psnr:.4f}')

# With div loss and coeff = 0.01, 21.7755
# With no div loss: 27.3784 (so better with no div loss)
# With div loss coeff = 0.01 and vort loss coeff = 0.01: 22.2877
# With div coeff = 0.00001 and vort loss coeff = 0.00001: 26.4322
# With scale = 2, div and loss coeff = 0.00001 : 30.1367
# With scale = 2, coeffs = 0: ___
# With all dataset, scale = 4, coeff = 0.00001 : 35.8653
# With train on 1000 dataset; eval on 100 test: 36.2829
```

```
Average Test PSNR: 36.5077
```

Use SSIM

In [ ]:
```
model.load_state_dict(torch.load(f'best_swinir_model_{scale}_div{lambda_div}

# Evaluate on validation set
# Ensure model is on the correct device before evaluation
model.to(device)
val_ssim = evaluate_ssim(model, eval_loader, device)
print(f'Average Test SSIM: {val_ssim:.4f}')

# Previous output: Average Validation SSIM: 0.7489 (no divergence loss used)
# Now: Average Validation SSIM: 0.1618 (divergence loss used, with divergenc
# Now with div loss and coeff=0.1: 0.1747
# Now with coeff=0.01: 0.2772
# Using div loss coeff = 0: 0.6623
# With div loss coeff = 0.01 and vort loss coeff = 0.01: 0.2131
# With div coeff = 0.00001 and vort loss coeff = 0.00001: 0.5488
# With scale = 2, div and loss coeff = 0.00001 : 0.8226
# With scale = 2, coeffs = 0: 0.7665
# With all dataset, scale = 4, coeff = 0.00001 : 0.9165
# With train on 1000 dataset; eval on 100 test: 0.9189
```

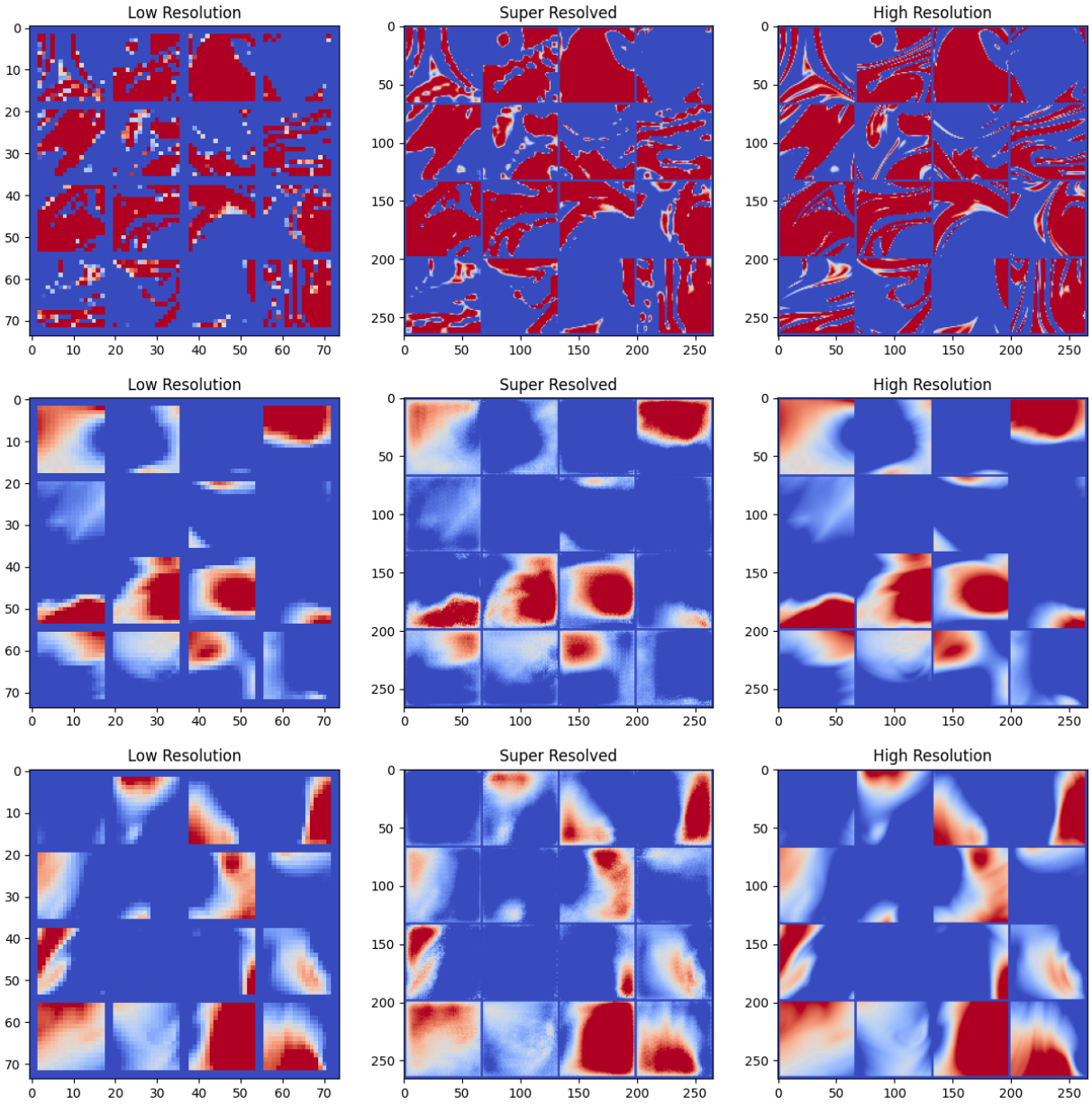
```
Average Test SSIM: 0.9214
```

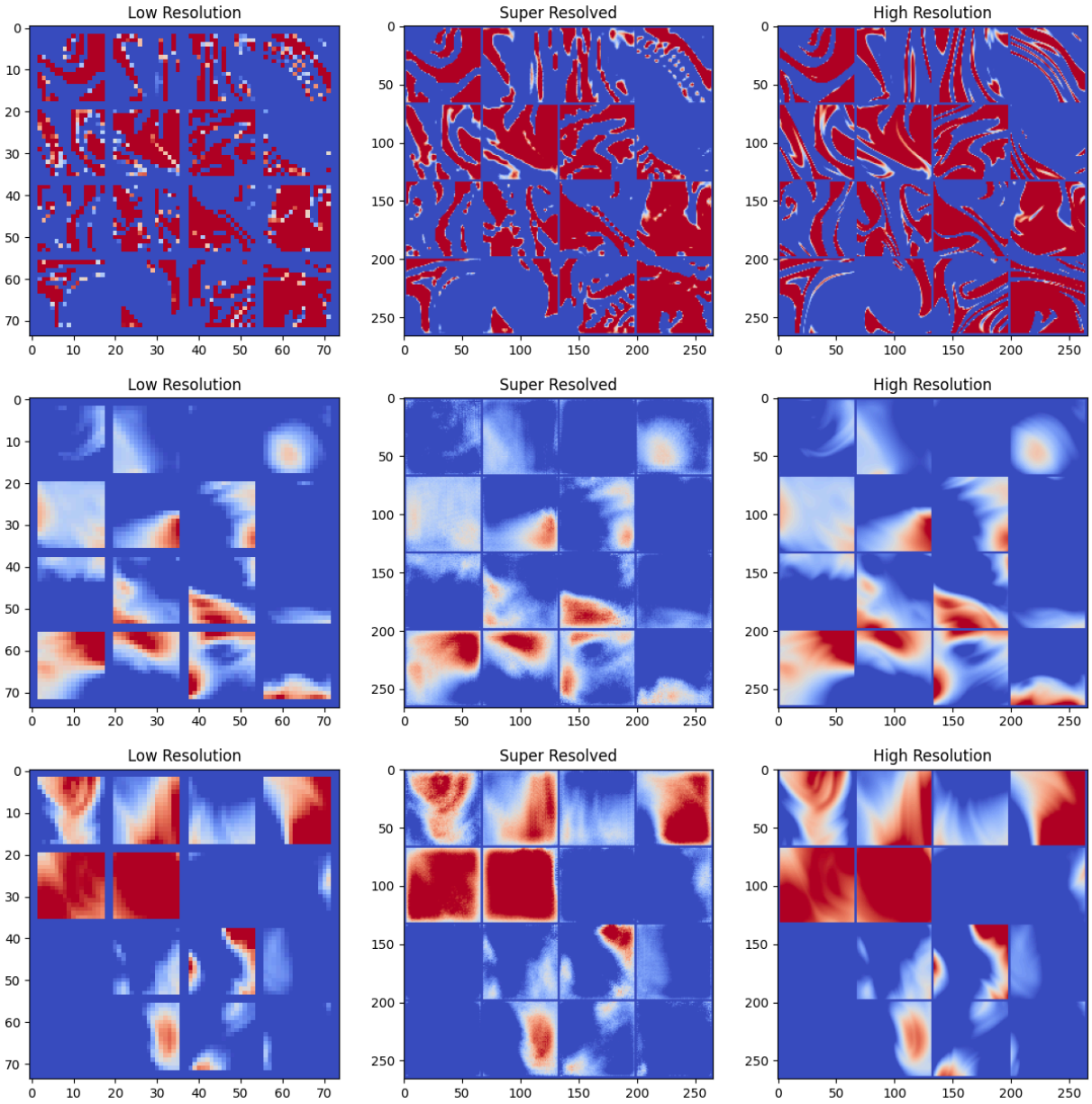
Visualize

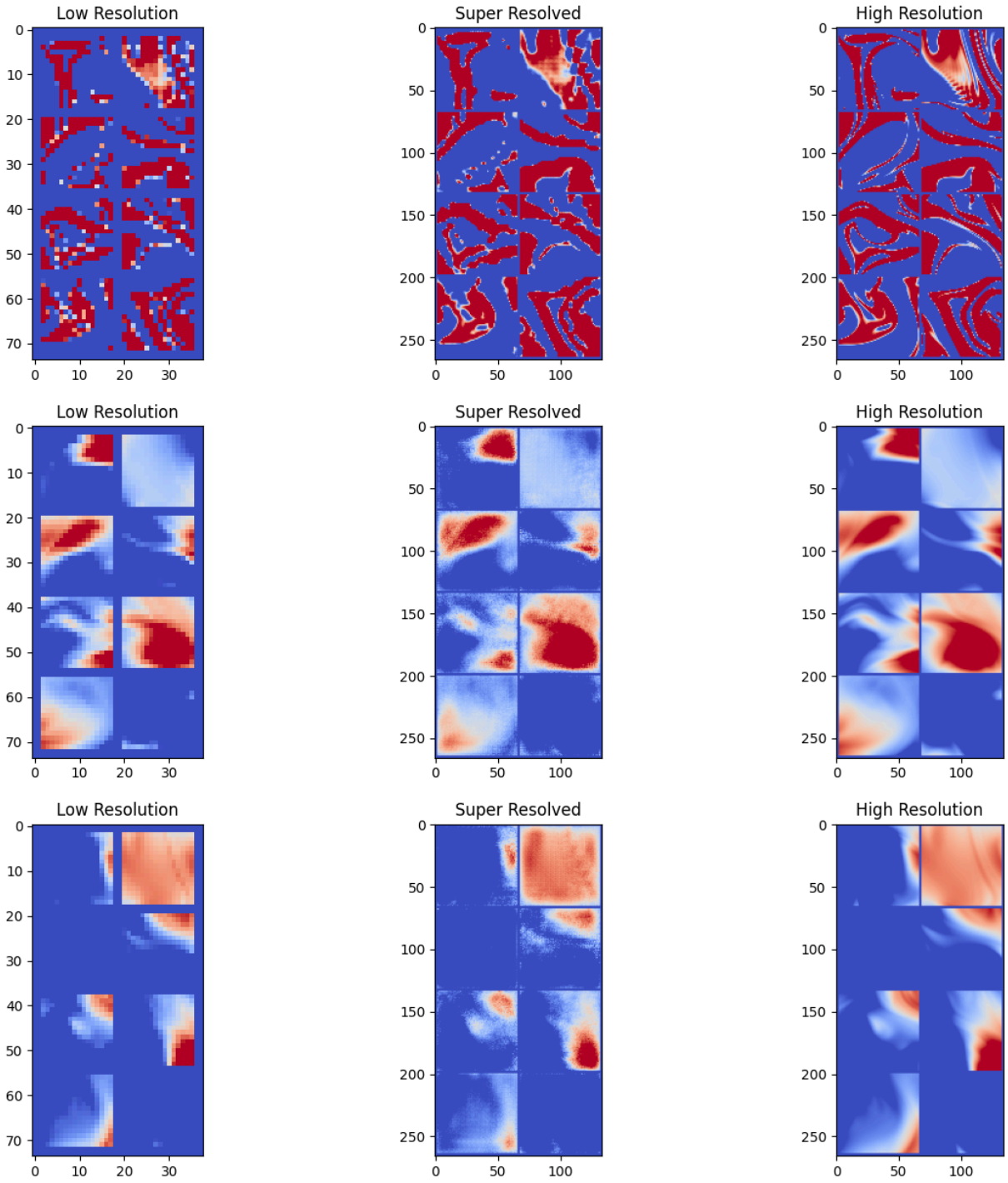
In [ ]:
```python
def visualize_results(model, dataloader, num_samples=3):
    model.eval()
    with torch.no_grad():
        for i, batch in enumerate(dataloader):
            if i >= num_samples:
                break

            lr = batch['lq'].to(device)
            hr = batch['gt'].to(device)

            # Get model prediction
            sr = model(lr)

            # Convert to numpy
            lr_np = tensor2img(lr)
            hr_np = tensor2img(hr)
            sr_np = tensor2img(sr)

            # Plot
            fig, axes = plt.subplots(3, 3, figsize=(15, 15))

            for j in range(3):
              axes[j][0].imshow(lr_np[..., j], cmap='coolwarm')  # Show u co
              axes[j][0].set_title('Low Resolution')
              axes[j][1].imshow(sr_np[..., j], cmap='coolwarm')
              axes[j][1].set_title('Super Resolved')
              axes[j][2].imshow(hr_np[..., j], cmap='coolwarm')
              axes[j][2].set_title('High Resolution')
            plt.show()
```

On validation dataset

In [ ]:
```python
# Ran with: all dataset; scale = 4; div,vort = 0.00001

model.load_state_dict(torch.load(f'best_swinir_model_4_div1e-05_vort1e-05_pa

# Visualize some results
visualize_results(model, val_loader)

# Save the model for later use
torch.save(model.state_dict(), f'swinir_fluid_flow_4_div0.00001_vort0.00001.
```

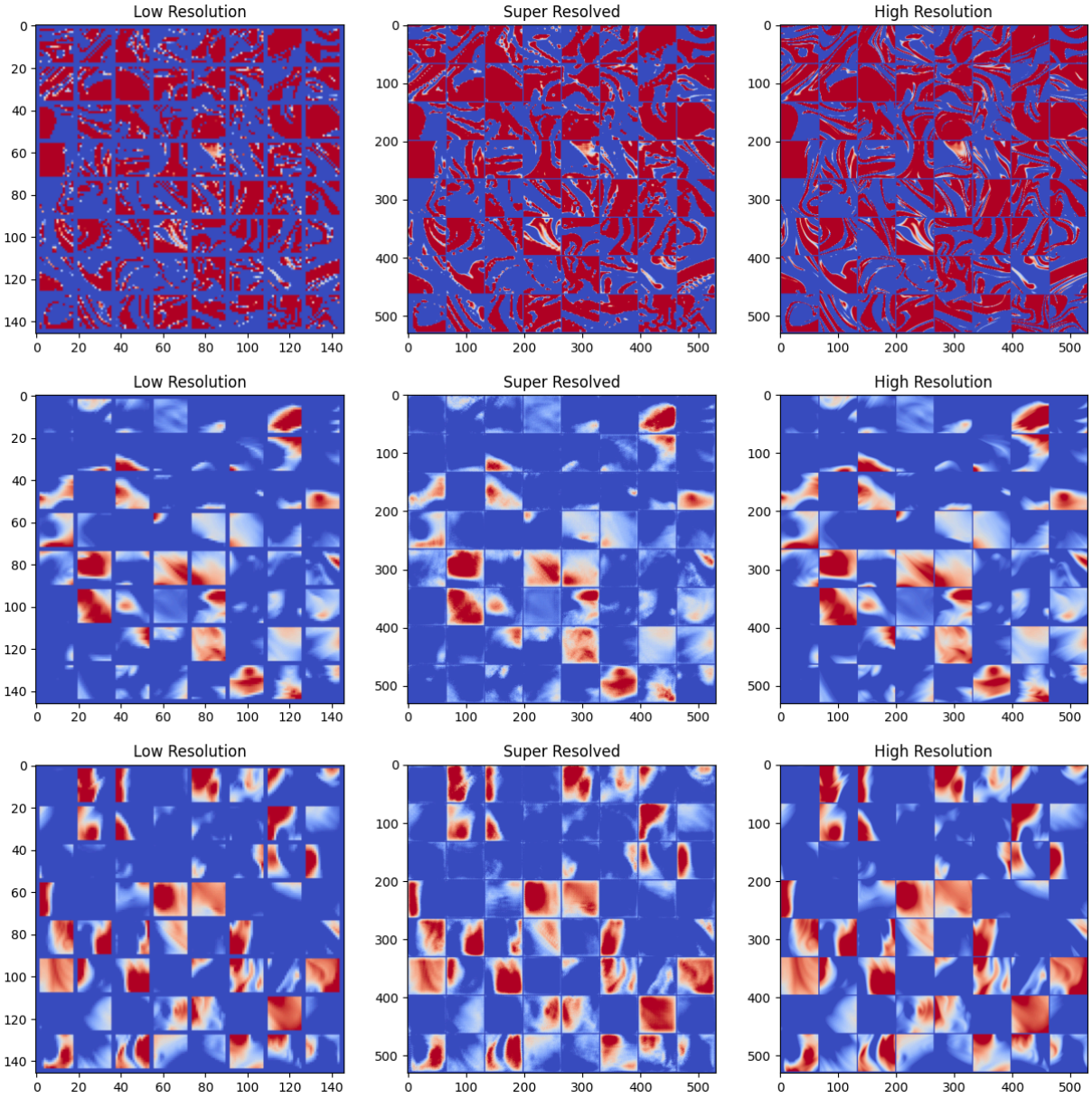

Low Resolution
Super Resolved
High Resolution
Low Resolution
Super Resolved
High Resolution
Low Resolution
Super Resolved
High Resolution

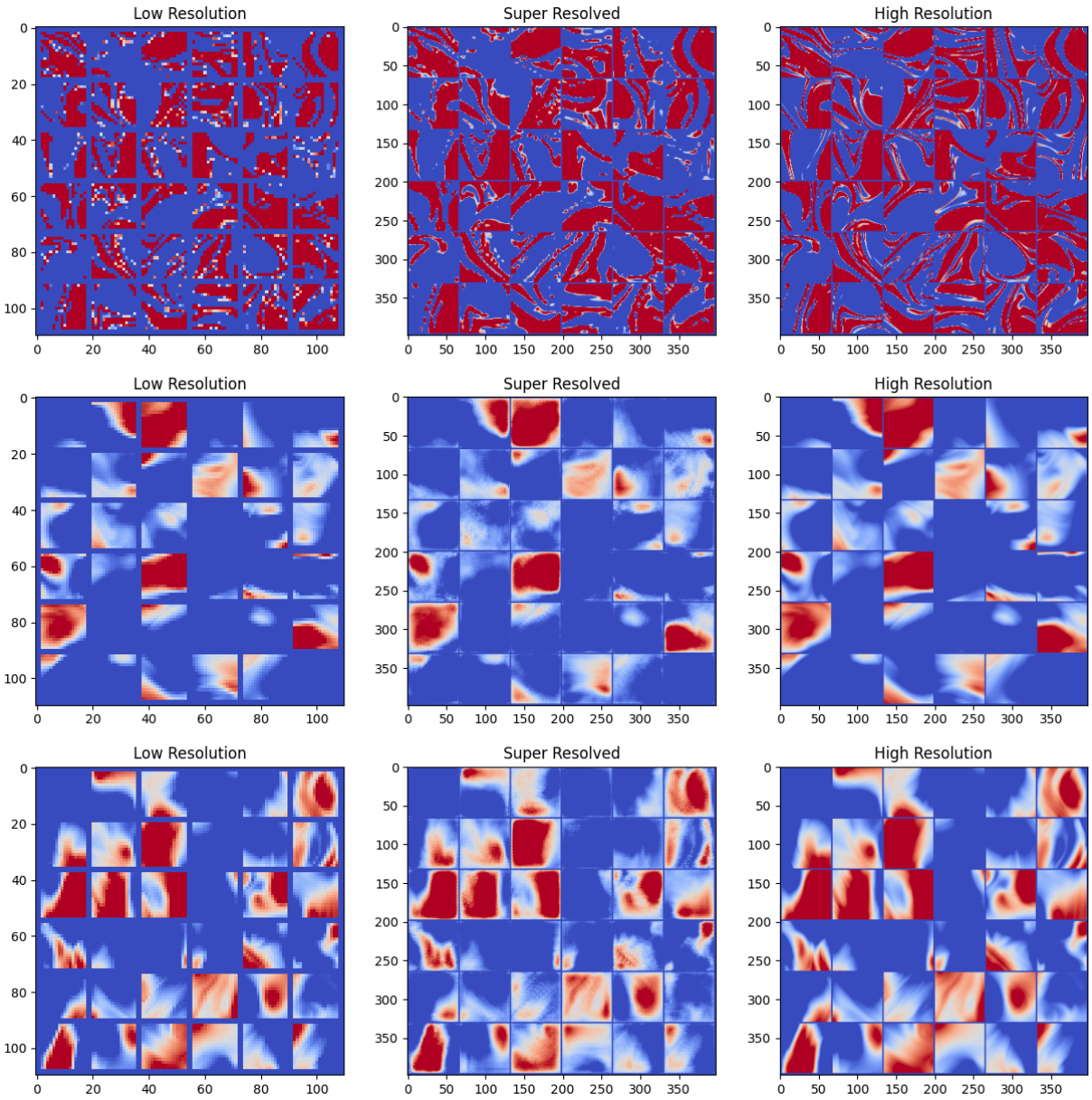

Low Resolution
Super Resolved
High Resolution
Low Resolution
Super Resolved
High Resolution
Low Resolution
Super Resolved
High Resolution



On evaluation dataset

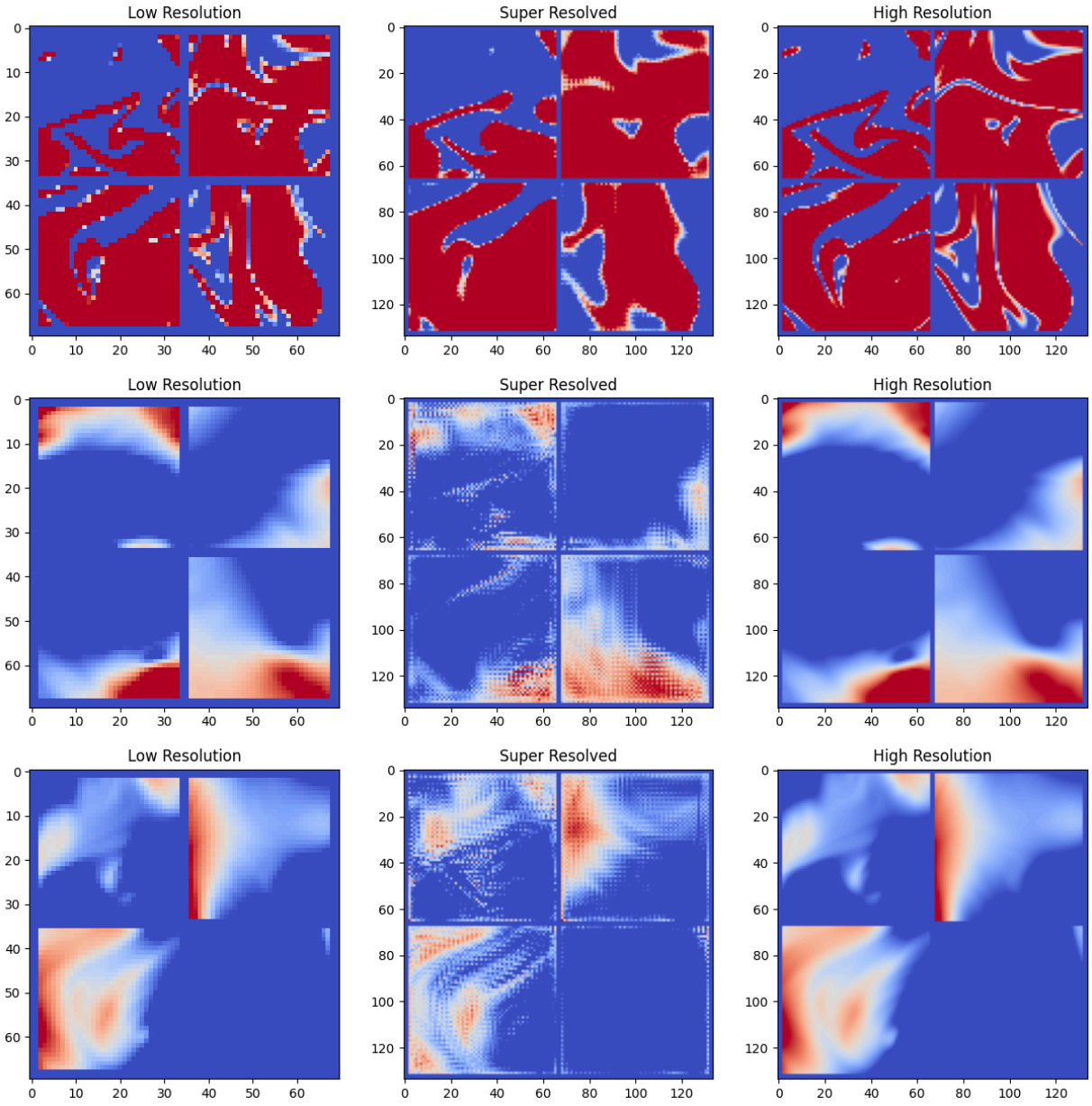
In [ ]:
```
# Ran with: all dataset; scale = 4; div,vort = 0.00001

model.load_state_dict(torch.load(f'best_swinir_model_4_div1e-05_vort1e-05_pa

# Visualize some results
visualize_results(model, eval_loader)
```


Low Resolution
Super Resolved
High Resolution
Low Resolution
Super Resolved
High Resolution
Low Resolution
Super Resolved
High Resolution

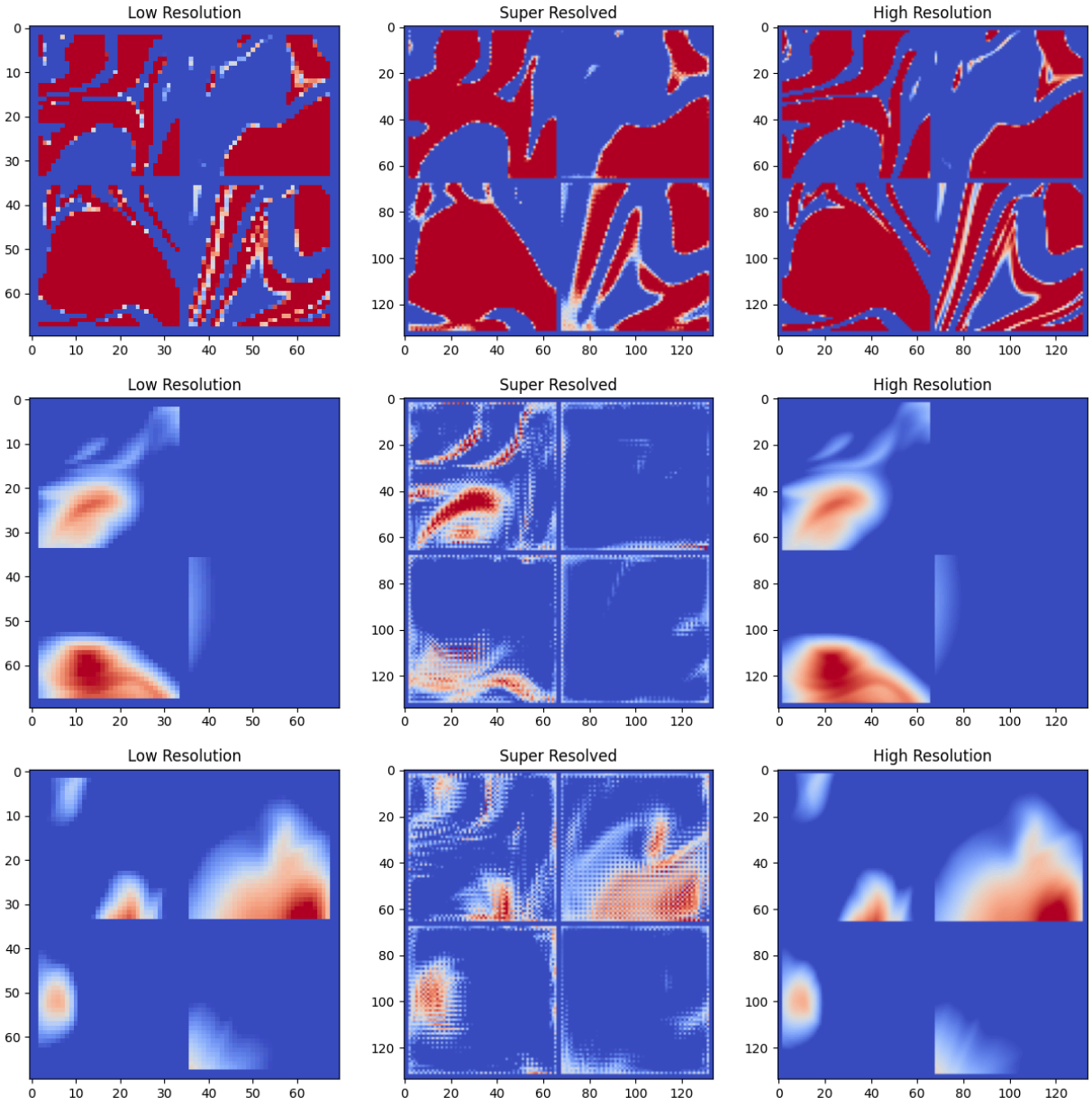


Previous

In [ ]:
```
# Ran with scale = 2; div,vort = 0.00001

model.load_state_dict(torch.load(f'best_swinir_model_2_div0.00001_vort0.0000

# Visualize some results
visualize_results(model, val_loader)
```



In [ ]:
```
# Ran with scale = 2; div,vort coeff = 0

model.load_state_dict(torch.load(f'best_swinir_model_2_div0_vort0.pth'))


# Visualize some results
visualize_results(model, val_loader)

# Save the model for later use
torch.save(model.state_dict(), f'swinir_fluid_flow_{scale}.pth')
```


Low Resolution
Super Resolved
High Resolution
Low Resolution
Super Resolved
High Resolution
Low Resolution
Super Resolved
High Resolution