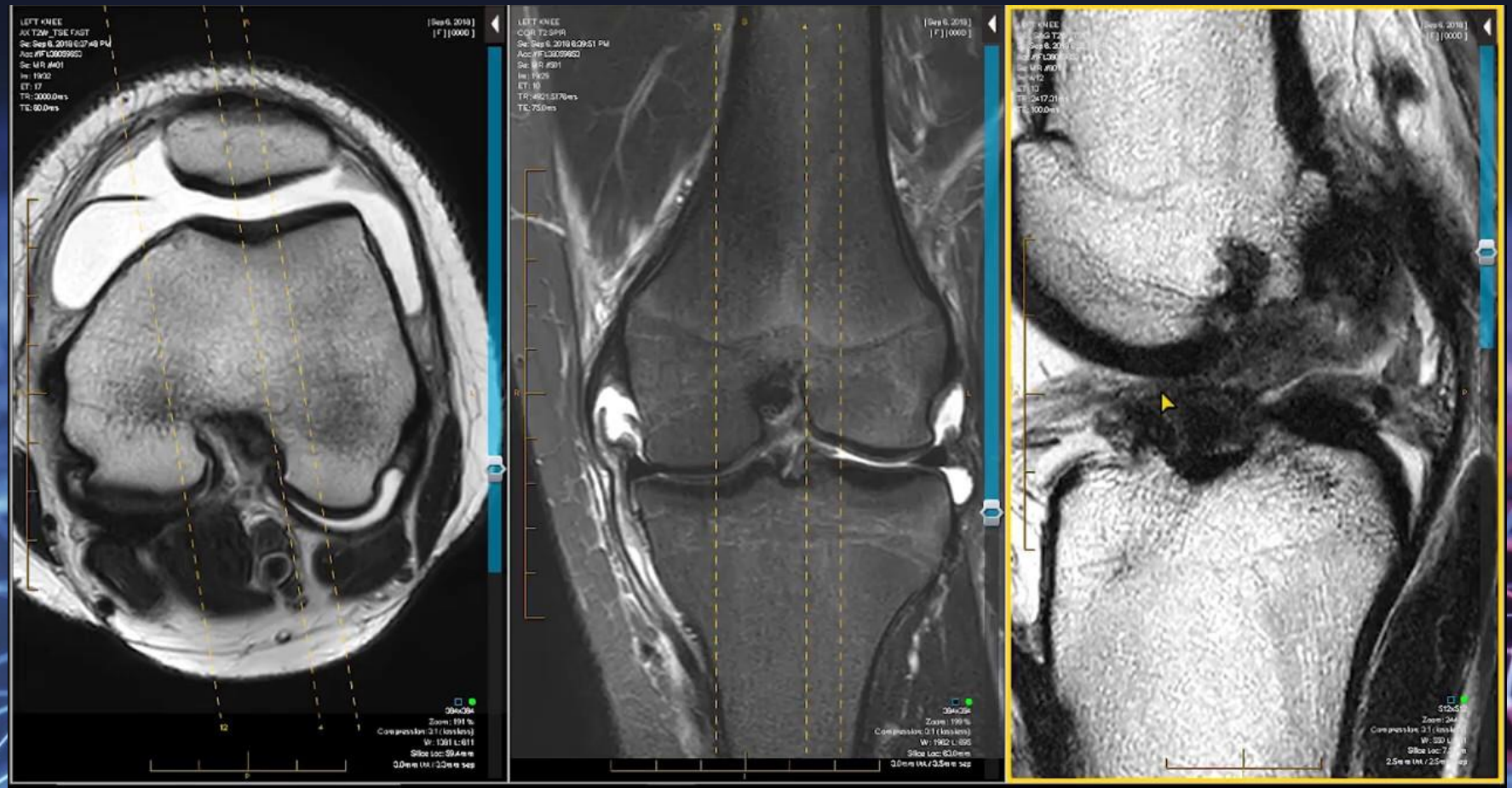
Laplacian pyramid-based complex neural network learning for fast MR imaging

Ketbjano Vocaj – Matteo Russo – Paolo Tarantino

Magnetic Resonance Imaging (MRI)



From wikipedia.org: ”Magnetic resonance imaging (MRI) is a medical imaging technique used in radiology to form pictures of the anatomy and the physiological processes of the body”.

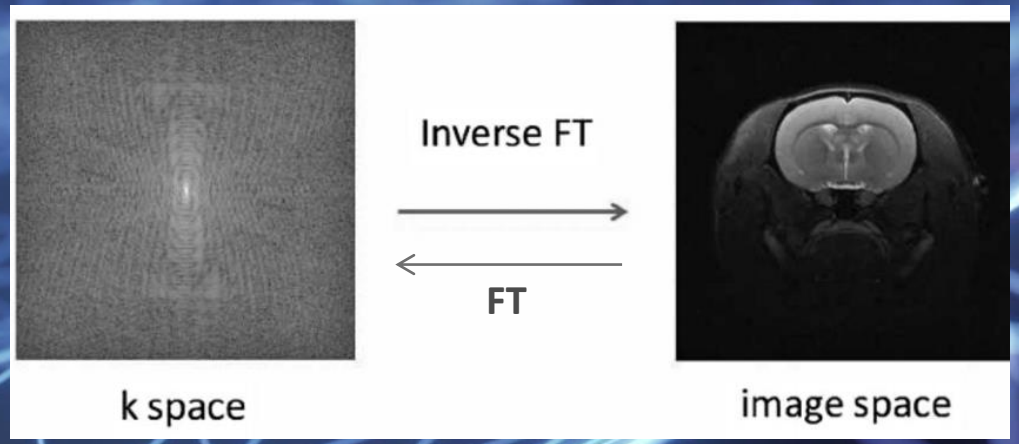
Purposes and issues

Issue: Scan time is the limiting factor on performance.

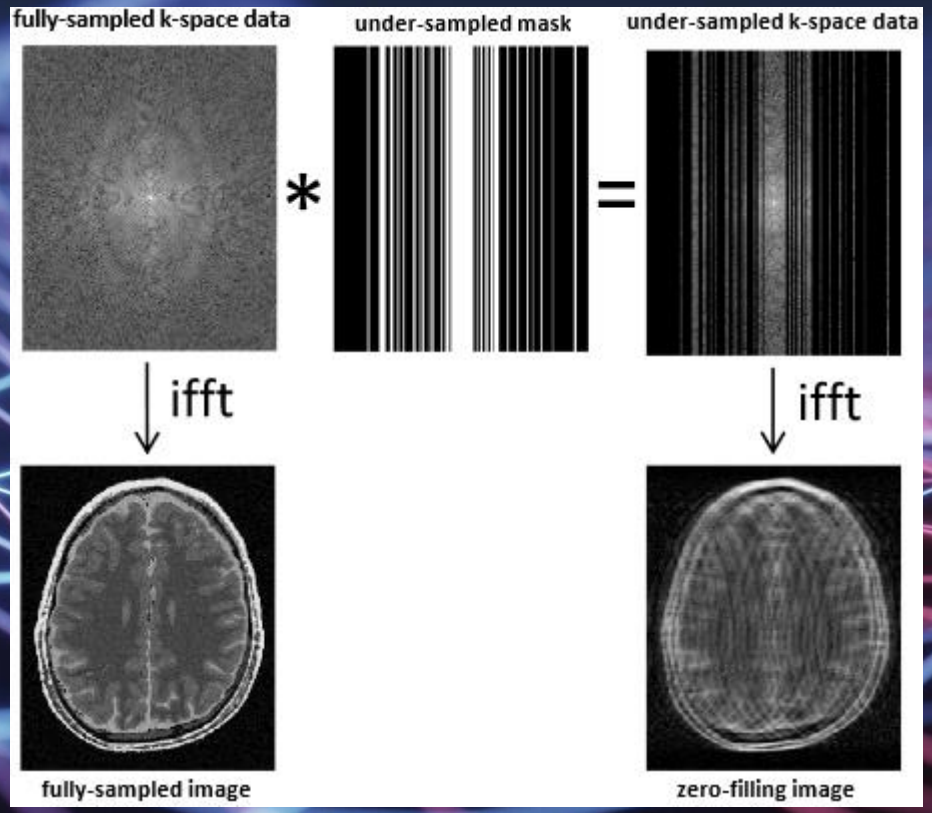
Goal: Find methodologies for reconstructing high-resolution images in a fast way.

Solution: undersample the k-space data.

K-space and Undersampling



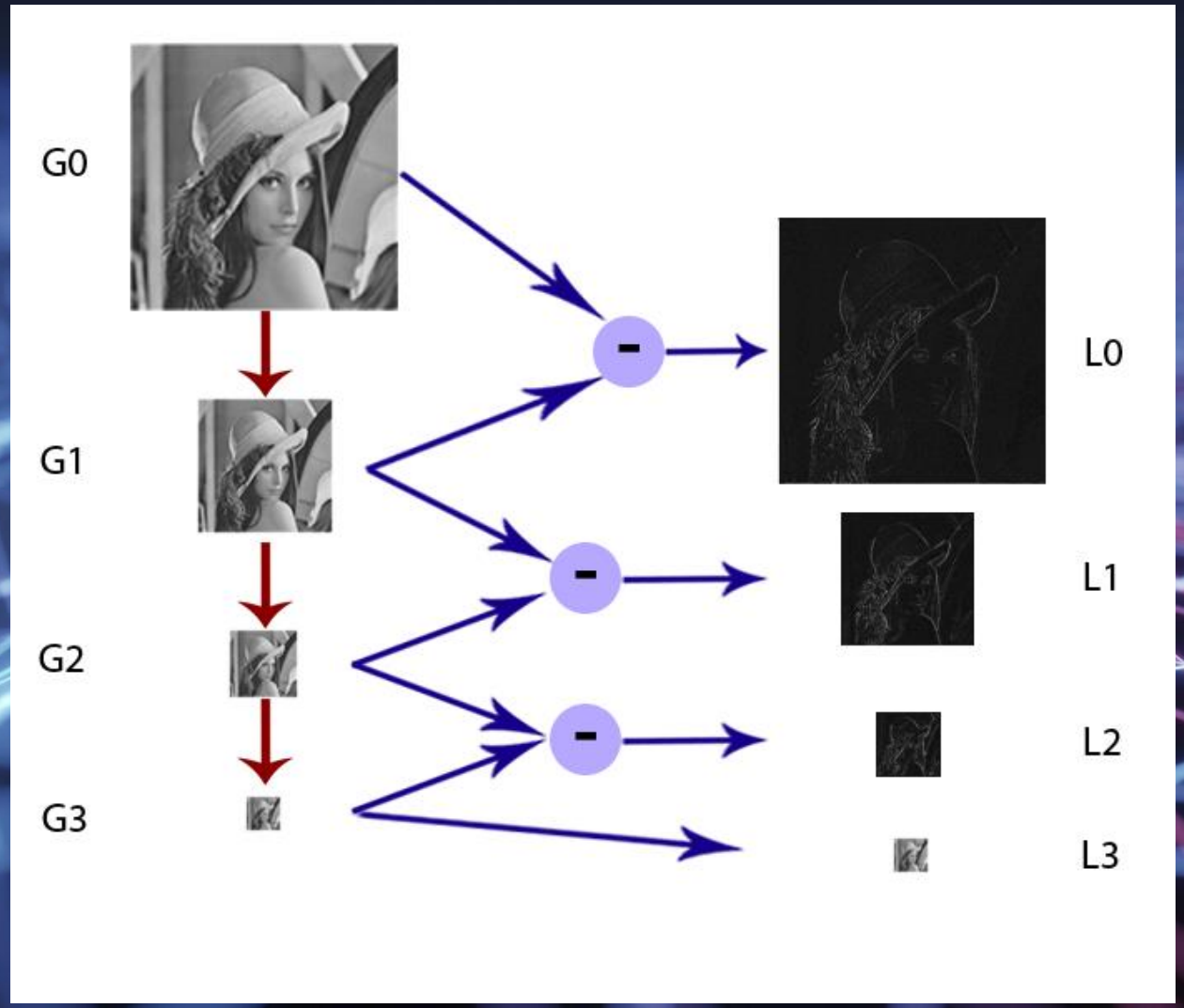
K-space and Fourier Transform



Undersampling procedure

Complex Laplacian Decomposition

Gaussian Pyramid

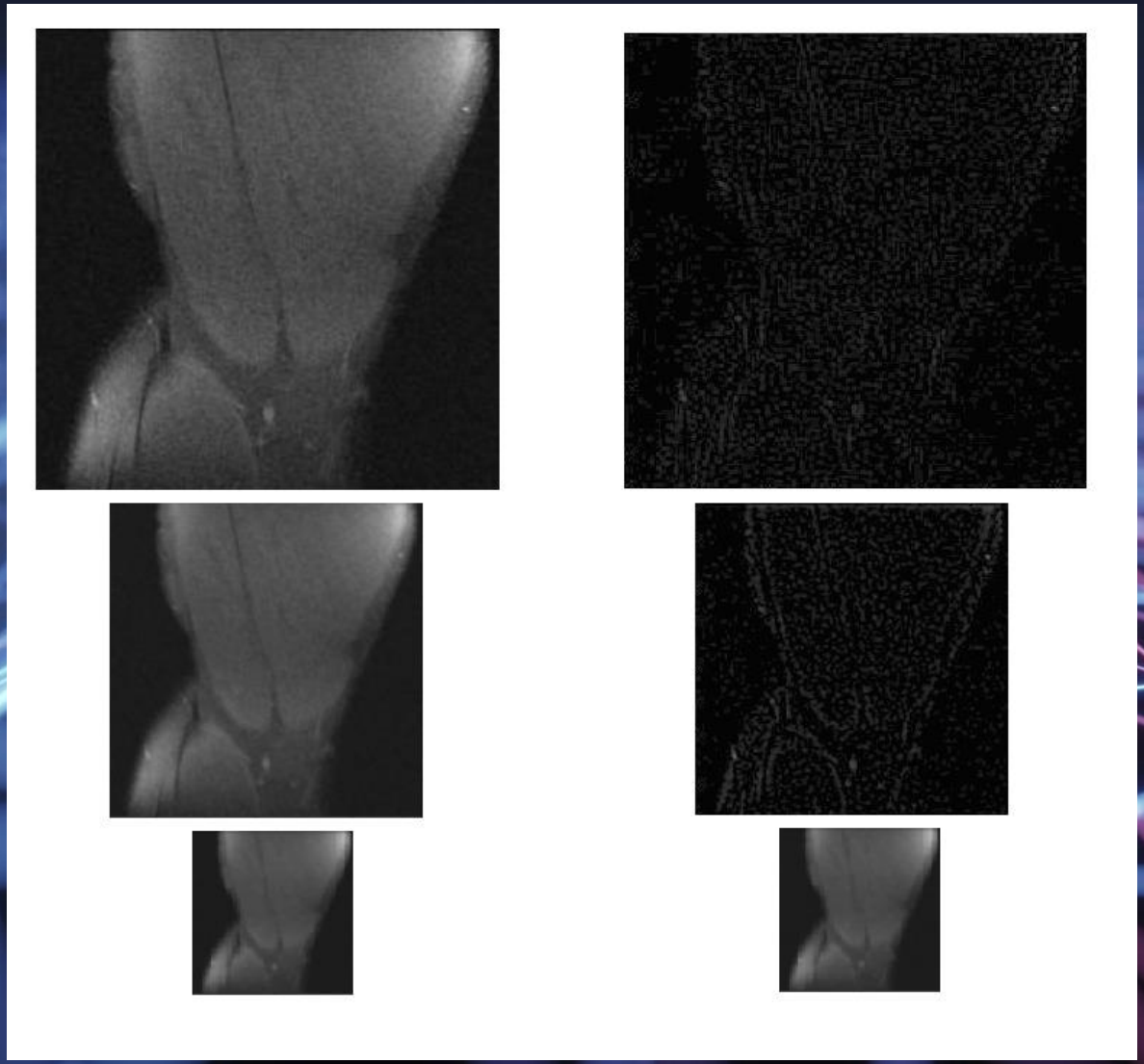


Laplacian Pyramid

$$L_0 = G_0 - G_1$$

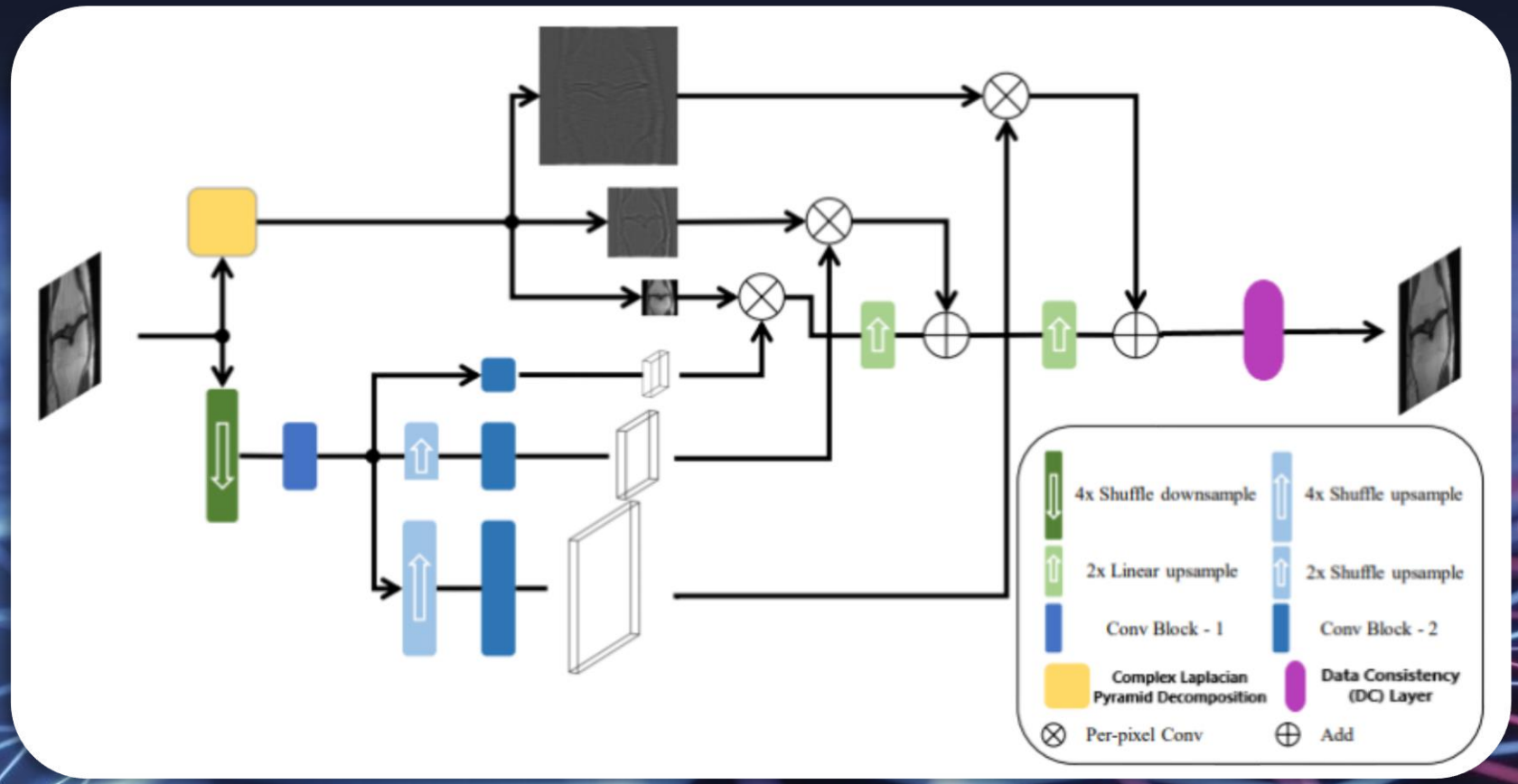
Complex Laplacian Decomposition (2)

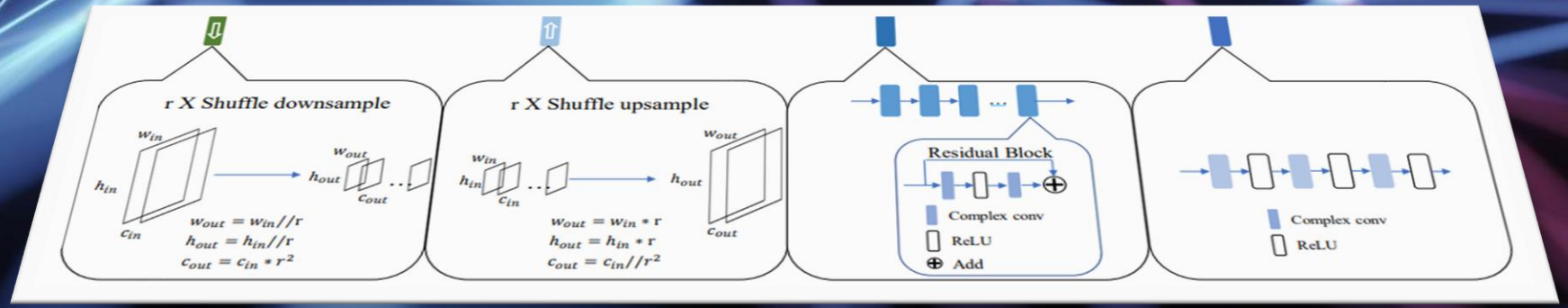
Gaussian Pyramid



Laplacian Pyramid

Network's Structure

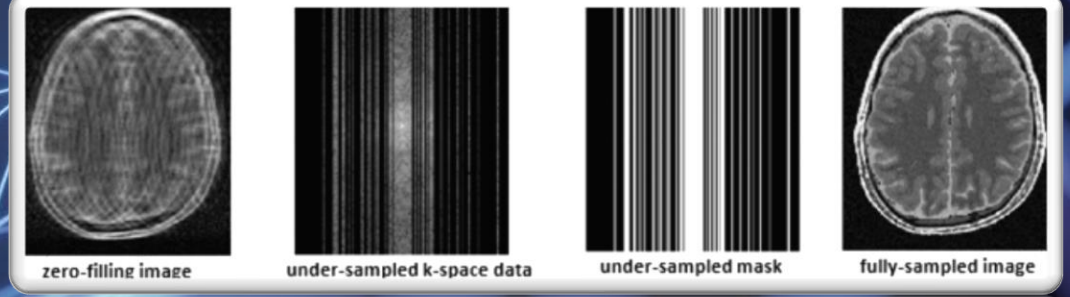




Training The Network

```python
mask_func = RandomMaskFunc(center_fractions=[0.04], accelerations=[8])

def data_transform(kspace, mask, target, data_attributes, filename, slice_num):
    # Transform the data into appropriate format

    ifft_kspace = fastmri.ifft2c(T.to_tensor(kspace))
    crop_kspace = T.complex_center_crop(ifft_kspace, (320,320))
    orig_kspace = fastmri.fft2c(crop_kspace)
    masked_kspace, mask = T.apply_mask(orig_kspace, mask_func)

    mr_img =  fastmri.ifft2c(masked_kspace)

    return mr_img, masked_kspace, mask, target

dataset = mri_data.SliceDataset(
    root=pathlib.Path('./trainset'),
    transform=data_transform,
    challenge='singlecoil'
)
```

Pre-training



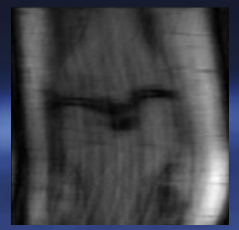
Data employed

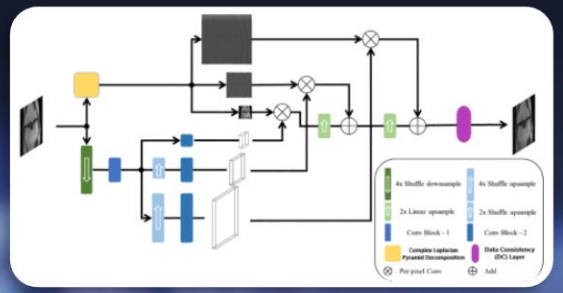
Training code

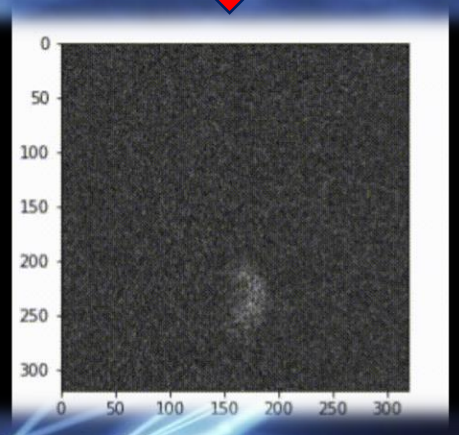
```python
criterion= nn.L1Loss()
optimizer= optim.Adam(net.parameters(), lr=0.0001, betas=(0.9, 0.999),
                      eps=1e-08, weight_decay=0.95, amsgrad=False)

for epoch in range(20):  # loop over the dataset multiple times

    for mr_img, masked_kspace, mask, target in dataset:

        outputs = net(mr_img, masked_kspace, mask)

        loss = criterion(outputs, target)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        # print statistics
        avg_psnr += PSNR(outputs, target).item()
        avg_ssim += SSIM(outputs, target).item()

        pass

    avg_psnr_f += avg_psnr/count_slice
    avg_ssim_f += avg_ssim/count_slice

print('Finished Training')
```

Testing and Results

Undersampling

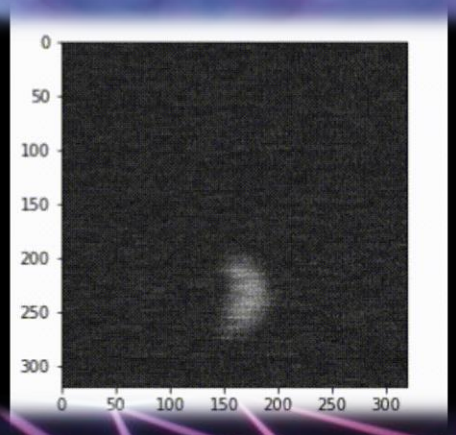






Fully Sampled (target)

Cascade loop

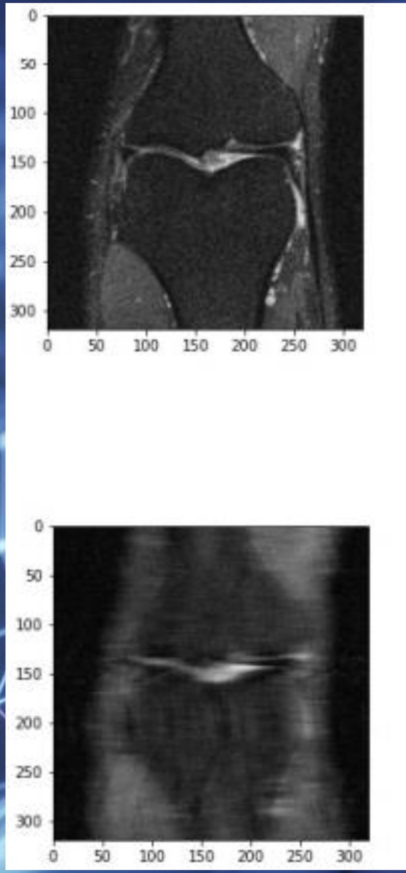


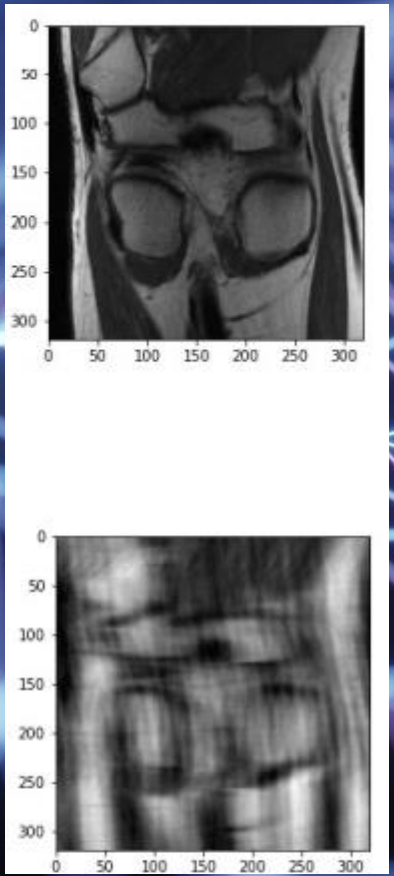
Network outputs

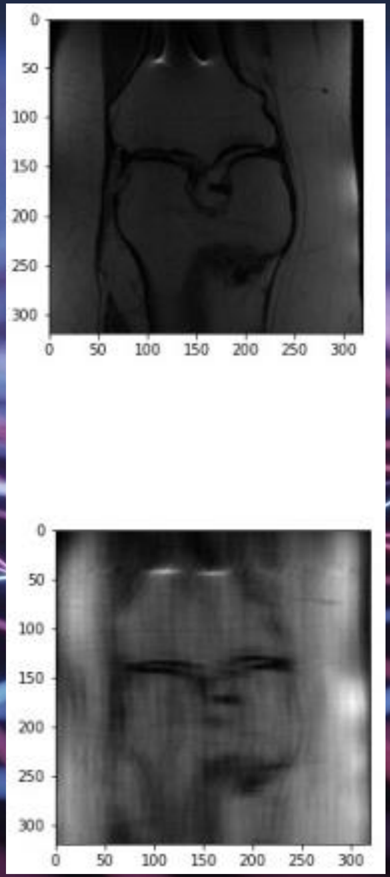
Cascade	SSIM	PSNR
2	0.32	19.63

Results obtained

Testing and Results (2)







SAPIENZA
UNIVERSITÀ DI ROMA

THE END ☺