MACHINE LEARNING DENOISING SOLUTION FOR ACOUSTOELECTRIC IMAGING

by

Eric Nelson

―――――――――――――

A Thesis Submitted to the Faculty of the

JAMES C. WYANT COLLEGE OF OPTICAL SCIENCES

In Partial Fulfillment of the Requirements

For the Degree of

MASTER OF SCIENCE

In the Graduate College

THE UNIVERSITY OF ARIZONA

2024

THE UNIVERSITY OF ARIZONA
GRADUATE COLLEGE

As members of the Master's Committee, we certify that we have read the thesis prepared by **Eric Nelson**, titled *Machine Learning Denoising Solution for Acoustoelectric Imaging* and recommend that it be accepted as fulfilling the dissertation requirement for the Master's Degree.

_____   Date: 05/06/24
Professor Russell S. Witte

_____   Date: 05/06/24
Professor Florian Willomitzer

_____   Date: 05/06/24
Professor Janet Meiling Roveda

Final approval and acceptance of this thesis is contingent upon the candidate's submission of the final copies of the thesis to the Graduate College.

I hereby certify that I have read this thesis prepared under my direction and recommend that it be accepted as fulfilling the Master's requirement.

_____   Date: 05/06/24

Professor Russell S. Witte
Master's Thesis Committee Chair
Wyant College of Optical Sciences

Acknowledgements

Completion of my thesis, as well as the personal development achieved along the way, could not have been done on my own. There are far too many people that I must thank for helping me on my journey that I would be certain to forget some if I was to name them here. A few integral influences will be addressed. I extend my deepest thanks to my supervisor and mentor, Professor Russell Witte, who has believed in me since I first came to the University of Arizona. Russ first introduced me to Arizona by giving me an internship position at his start up company where I gained some foundations in the topic that would later become my thesis project. Upon completion of my internship, I joined my first lab in the college of optical sciences, the intelligent imaging and sensing laboratory led by Professor Amit Ashok. After a year I realized that the projects I was supporting were not fulfilling my interests. I once again contacted Professor Russell Witte about joining his lab. After meeting with Russ, I began attending the weekly meetings of the Experimental Ultrasound and Neural Imaging Laboratory where I was greeted by my previous colleagues. Russ quickly found me a project which he thought I could contribute to with my new experience. This project became my thesis topic.

As my thesis project developed, collaborating labs began getting involved. I next want to thank our collaborators at the University of Washington led by Professor Matthew O'Donnell. I also want to thank our collaborating lab here at the University of

Arizona led by Professor Janet Roveda. Janet's lab provided guidance to get past the many obstacles faced during my project and Janet went on to be on my thesis committee. I also would like to thank the final member of my thesis committee Professor Florian Willomitzer. Before Florian joined my thesis committee, I was a student in his Computational Imaging and Computer Vision course where I further developed my technical knowledge on computational imaging as well as my technical communication skills. I also want to recognize both Jini Kandyil and Professor Brian Anderson who have been great advisors to me since I've come to Arizona. Both Jini and Brian have had open door policies with me and have provided me with all the guidance I needed to navigate graduate school

Finally, I want to thank my cohort of students at the College of Optical Sciences who came with me from the University of Nevada, Reno. Having these friends as my support system is a large part of why I was successful in this program. This includes my friend and mentor Christopher Salinas who is also in Russ' lab. Chris has been my friend since high school and is the primary influence which brought me here to the College of Optical Sciences. These are some of many influences I've had over the years, thank you for helping me grow into a person I can be proud of.

Dedications

For my family. To my mother and sisters who taught me to love, laugh, and live. To my father who taught me to be curious. And to my grandparents who taught me to always be my best, to always grow, and to always believe in the future.

Table of Contents

List of Figures

Abstract

Acoustoelectric imaging is a novel medical imaging technique that uses ultrasound to image the flow of current in a tissue. Acoustoelectric imaging has a wide range of high impact applications such as being able to study and diagnose diseases of the heart and brain as well as being a potential surgical assistive tool. The primary obstacle in bringing acoustoelectric imaging to the forefront of medical imaging is the inability to effectively perform the technique noninvasively. Performing the technique in-vivo requires cutting edge ultrasound and radiofrequency (RF) sensing technology, and along with the ultra-high sensitivity of the instruments comes a high susceptibility to background noise. The very small acoustoelectric signals end up buried in a high noise floor, thus the need for high performing denoising capabilities. This thesis explores the potential of pursuing machine learning as a denoising solution. An acoustoelectric data simulation pipeline is used to provide realistic acoustoelectric data along with the corresponding ground truth. A machine learning architecture called HINet is explored for its reputation as being high performing in image restoration tasks. Through a series of hyperparameter testing, the HINet model is optimized for the simulated acoustoelectric data and is found to outperform the traditional denoising methods by an average of 10dB by PSNR.

Chapter 1

Introduction

1.1 Background

1.1.1 Acoustoelectric effect

In medical imaging, the physiology that we're interested in may not always be directly observable. Sometimes, that which we wish to see may be measurable, but the image quality may be fundamentally limited by the nature of the signal or the system. In such situations, one may wish to exploit another related physical phenomenon to act as a contrast mechanism in an image. One such physical phenomenon that one may wish to use is the acoustoelectric (AE) effect.[1]

The AE effect is a novel physical mechanism produced from the interaction of ultrasound waves and electric current flowing through a medium. Ultrasound is coupled into a current carrying medium; because the ultrasound is a traveling pressure wave, the medium is contracted and expanded according to that pressure wave. This contraction and expansion modulate the density of the material and thus also modulates the electrical resistivity of the medium. This encodes the underlying electric signal according to the modulated resistivity of the medium. This resultant AE signal produced is proportional to the underlying current source density within the medium.

1.1.2   Acoustoelectric Imaging Theory

Acoustoelectric Imaging (AEI) is a 4-D medical imaging technique in which

ultrasound is coupled into a current carrying medium and multiple remote recording

electrodes (leads) measure the resultant AE voltage perturbations.[The resultant AE

signal on a given lead will be a function of the electrical properties of the medium, the

geometry of the system, the underlying current source density, and the ultrasound

profile. The following figure will describe the AE voltage signal V_i^{AE} for a given lead i is

produced. Please note that the fundamental AE interaction constant K is material

specific, and the US pulse waveform a(t) refers to the slow time propagation of the

ultrasound.[2]

$$V_i^{AE} = \iiint (\bar{J}^L \bullet J^I)(-K_1\rho_0\Delta P)\, dx'dy'dz'$$

The ultrasound modulated electrical signal is proportional to the local current density.

$$\Delta P(x,y,z,t) = P_0 b(x,y,z)a(t - z/c)$$

US beam

Ultrasound modulates resistivity of medium and thus the electrical signal via Ohm's Law.

Lead i

AE image

$\bar{J}_i^L$: lead field i (A·m⁻²)
J^I: current field (A·m⁻²)
ρ: electric resistivity (Ω· m)
K_I: interaction constant
ΔP: acoustic pressure (Pa)
$b(x,y,z)$: US beam field pattern
$a(t)$: pulse waveform
c: speed of sound (m/s)

Figure 1: The measured AE image is a function of the local current and ultrasound fields. [3]

There are many ways ultrasound can be sent into the medium which will directly affect the produced image. One common "sampling" strategy for ultrasound is focused ultrasound pulses. In this sampling strategy, a separate ultrasound pulse is sent to each point in the volume for each measurement. Focused ultrasound sampling provides the best resolution but the long sampling time. Although several sampling strategies were considered, for the purposes of this project a focused ultrasound sampling strategy was utilized. This concept will be described later when discussing the data simulation pipeline.

1.1.3 Acoustoelectric Data

Acoustoelectric data is typically recorded as a voltage on a recording electrode or lead. Multiple data channels are recorded on N different leads. Ultrasound is focused on different spatial locations in the measurement volume. In the direction of ultrasound propagation, the depth direction is sampled as the ultrasound propagates through the medium. By scanning the ultrasound beam across the X or Y direction, we can form a 2D image. This is called the slow time dimension and is converted to spatial depth via multiplication by the speed of sound through the medium. We also have the fast time dimension which contains information of how the local time-varying currents evolve. Finally, it is standard to collect many repeated measurements or trials of the

data for post processing purposes. This makes the full acoustoelectric dataset six

dimensional, including data channels, the three spatial dimensions of the volume, fast

time, and trials.
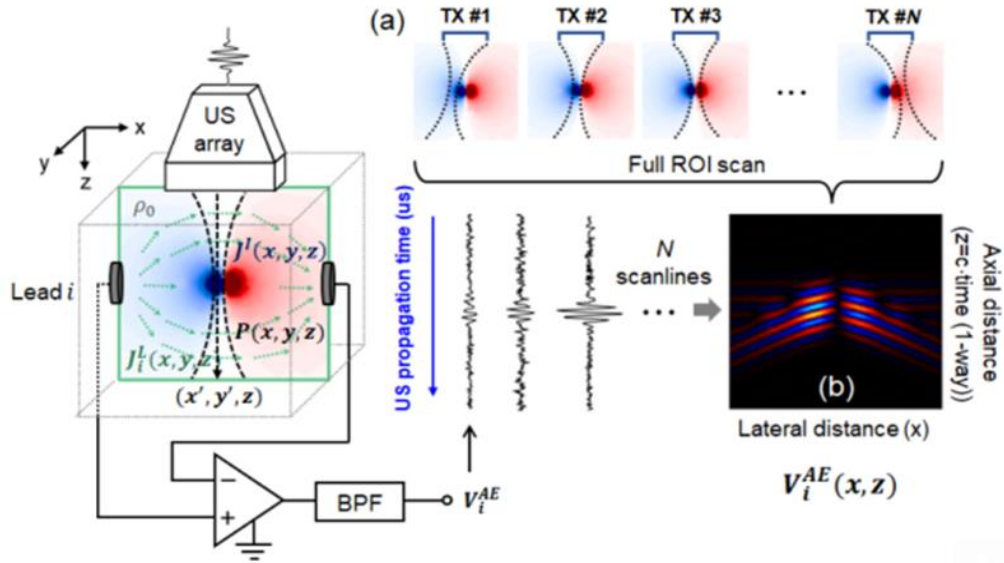
Figure 2: A 1D voltage vs. time plot is captured from a single electrode and a single firing of the ultrasound transducer. A 2D AE image can be captured by scanning the ultrasound in either the X or Y direction.[4]

1.2   AE Data Simulation

1.2.1   Simulation pipeline

The Experimental Ultrasound Neural Imaging Laboratory (EUNIL) here at the

University of Arizona has long collaborated with faculty at the University of

Washington on a wide variety of projects. One such project was on developing an AE

simulation pipeline to aid in software development and analysis tools. The resulting

pipeline involves separating the modeling into several professional modeling tools;

Field II, COMSOL, and MATLAB. Field II is a free professional simulation tool

specializing in modeling ultrasound fields and imaging using linear acoustics theory.

COMSOL is a general-purpose Multiphysics simulation tool that we use to model the

electrical and radio frequency properties of our selected geometry. The simulation files

produced in these tools are then imported into MATLAB where the resultant AE fields

and sensing are simulated.

Figure 3: Ultrasound fields are simulated in Field II. Electrical and RF properties are simulated in COMSOL. These simulated properties are imported into MATLAB to then model the resultant AE fields and RF measurements.[4]

1.2.2   Simulation Parameters

For the purposes of the thesis project, a simplified model system developed by our partners at the University of Washington is considered.[4]   A cube of dimensions 8x8x8 cm filled with saline (0.9% NaCl) is the medium. Ultrasound enters the medium through an acoustic window at the bottom of the cube. The ultrasound transducer is a customized 1.5D, 0.6 MHz concave-linear US array with 126 piezoelectric elements and a focus of 35 mm (Sonic Concepts Inc., Bothell, WA, USA). A single dipole current source, provided by a pair of platinum electrodes, is placed in the center of the cube, and is recorded by three gold cup recording electrodes positioned as seen in the following figure.

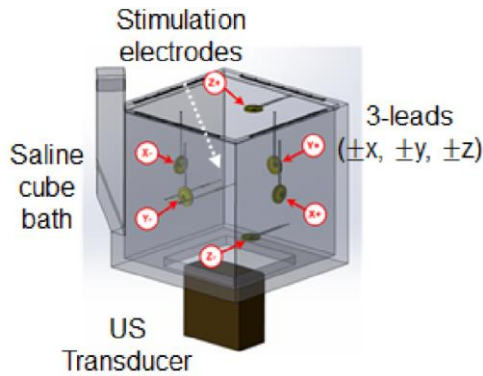Figure 4: Simulated saline cube geometry (Kang et al. 2022[3]).

The 0.9% NaCl saline in the cube is at room temperature which would make the acoustoelectric interaction constant $K_I = 0.034/MPa$ with the conductivity $\sigma = 1.2$ S/m. The electric fields simulated in COMSOL were as follows: a pair of cylindrical (diameter = 0.2 mm) platinum electrodes injected with a 200 Hz current waveform and three

cylindrical gold recording electrodes (diameter = 10mm) injected with unit current 1 A connected to a common ground. Realistic acoustic fields parameters were obtained from a programmable research US system (Vantage 64 LE, Verasonics Inc., Redmond, WA, USA) for our ultrasound transducer which were then fed into Field II to calculate the 3D spatial impulse response over time (4D). The maximum input pressure used is 1 MPa.

1.2.3 Dimensionality

For the sake of keeping our problem as simple as possible for a proof of concept, a limited dataset was modeled and simulated. The modeled physics of the generated dataset reflects that of a saline cube with a single, non-time-varying dipole in the center being measured by three electrode leads. The ultrasound is then steered only in the x-direction. For these assumptions, the 6D acoustoelectric dataset is reduced to a 4D dataset. This is due to the Y spatial dimension and the fast time dimension being invariant and thus being ignored for the purposes of simplicity. The generated dataset thus has the dimensions (channels, X-direction, Slow time or depth, Trials). To generate data that we would observe in an experiment, we add gaussian noise to a noiseless acoustoelectric image. Note: In an AE experiment, the ultrasound transducer acts as a passband filter to the system.

Single firing of ultrasound transducer

Scanning along X and Y results in a measurement of the full volume since ultrasound propagation gives us the depth measurement.
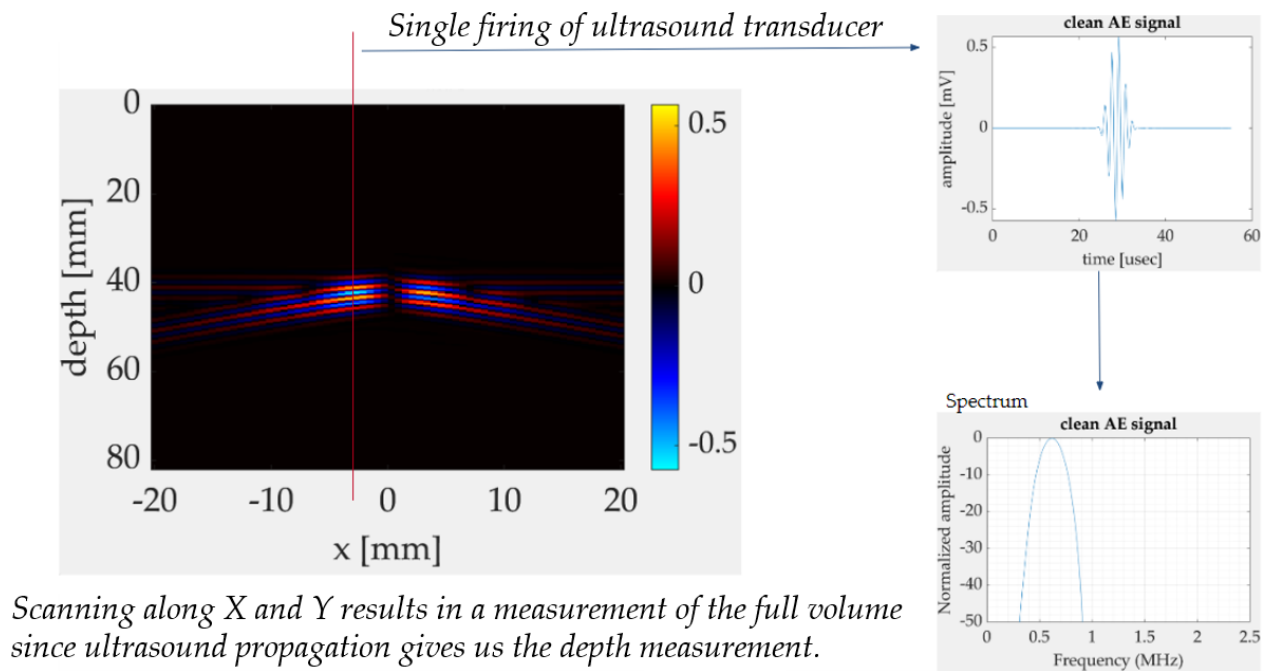
Figure 5: In the data simulation, first a noiseless acoustoelectric image is generated where the spectrum of the AE signal corresponds to the passband of the ultrasound transducer.[4]

1.3   Motivation

1.3.1   Potential advantages of AEI over other medical imaging

AEI allows the mapping of biological currents in a volume in real time at unprecedented spatial and temporal resolution.[7] Providing noninvasive 4D mapping of localized electrical activity at sub-millimeter spatial resolution and millisecond temporal resolution makes AEI competitive with the standard medical imaging modalities such as MRI, PET/CT, EEG, and ECG for imaging the brain and heart

18

respectively. However promising, AEI is not yet commonly deployed due to a few

obstacles preventing practical use. These obstacles will be discussed in depth in this

thesis.

1.3.2   Clinical AE Imaging for brain and heart disease diagnosis

Imaging the flow of electricity offers much potential utility for clinical

applications. Having an observable value which is directly proportional to the current

density of the tissue opens many doors. We can image current carrying tissues such as

those of the heart and brain. Many such biological signals are small and highly

localized, making other medical diagnostic tools with poor spatial resolution

capabilities ineffective. AEI provides noninvasive 4D mapping of localized electrical

activity which has the potential to greatly enhance our ability to both study and

diagnose diseases of the heart and brain.[6]
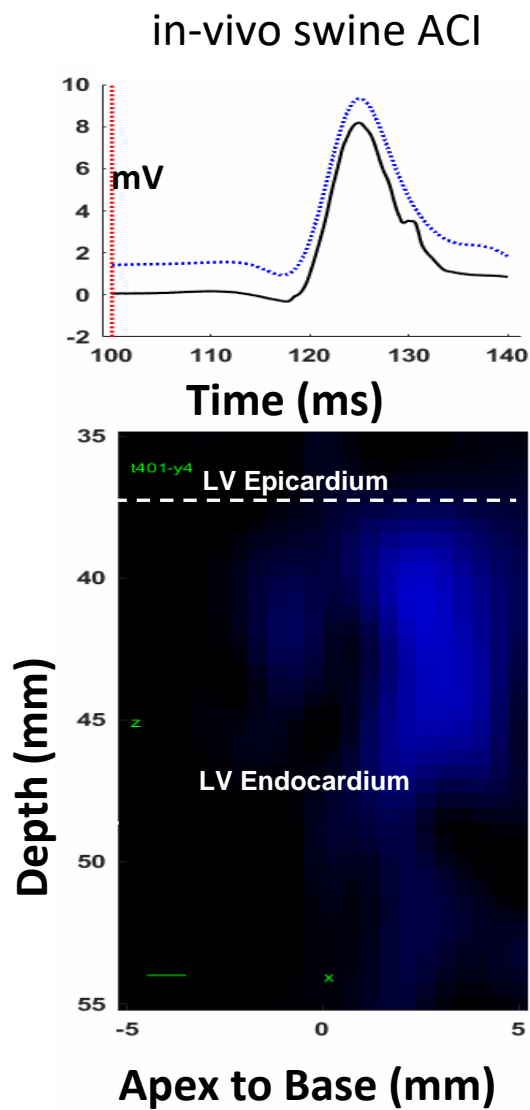
in-vivo swine ACI

Alvarez et al. 2020
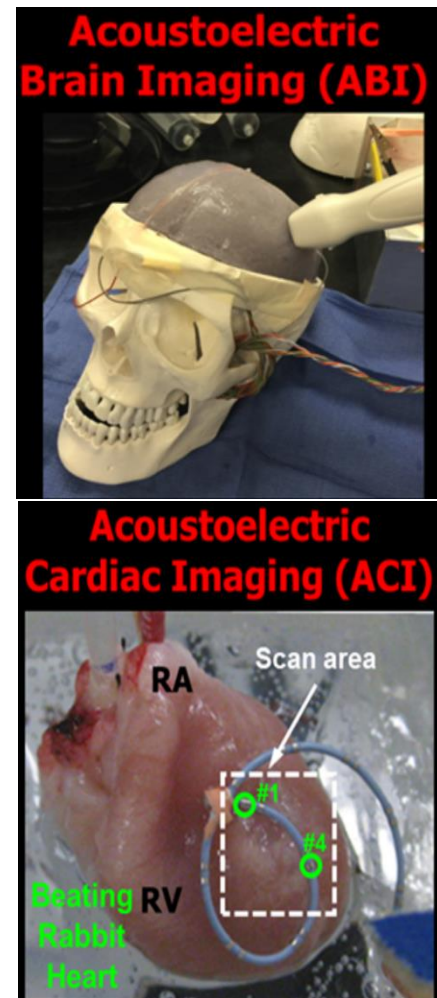
Qin et al. 2015

Figure 6: To the left is a current density reconstruction from AEI of an in-vivo swine heart.[12] The right shows images of AEI being performed on a human brain and in-vivo rabbit heart.[13]

1.3.3   Surgical assistive tool

In medicine, the prerogative is to do no harm and of course this extends to surgical procedures. However, the intricacies often involved for certain individuals can make this highly difficult. Particularly in surgeries involving highly sensitive tissues such as the heart and brain. It is often challenging to know where the exact boundaries of problem tissues are located. The high localization offered by AEI can assist with this problem. For example, with removing a section of the brain believed to be causing seizures in a patient with epilepsy, the healthy and troubled tissues may be impossible to distinguish to the human eye. But AEI can precisely indicate which sections of the tissue are firing during seizure episodes. Minimizing the amount of healthy brain tissue removed in this example would drastically improve that patient's outcome.

1.4   Problem

1.4.1   Noninvasive AE imaging

As previously mentioned, there are a few obstacles that need to be overcome before AEI becomes more practical for field use. The most significant will be discussed throughout this thesis. Currently, AEI can be performed with excellent results when the recording electrode is in proximity with the current source.[7] However, being that we are discussing a medical imaging modality, the ideal case would be performing this imaging noninvasively. Because the signal strength drops off with the distance

squared, this makes the task of performing AEI noninvasively quite challenging. The AE signal gets to a level low enough that the background noise begins to dominate.

1.4.2 *Weak signal detection with high noise floor*

Being that the original biological signals are already very small, and the produced AE signal will be even smaller, we suddenly have a big problem distinguishing our interested signal with the background noise. For example, the local currents in the brain can be less than 0.1 mA/cm^2.[5] The noise floor can be larger than this due to radiofrequency interference and aberrations related to ex-vivo ultrasound propagation. The state-of-the-art AE systems utilize cutting edge ultrasound and radiofrequency sensing technology to perform the imaging technique. With the ultra-sensitive sensing technology comes a very high susceptibility to background noise sources. Before imaging, there is an involved process of grounding the system to get the system noise floor as low as possible. The lowest noise floor we can achieve is still not low enough to perform AEI noninvasively.

1.4.3 *Need for a robust denoising method*

Even with every effort to minimize the noise floor, we need to put our raw measurements through a series of slow-time, fast-time, and bandpass filtering as well as averaging many samples. Although the increase in SNR from the raw images to the

resultant images is significant, the signal fall-off from noninvasive AEI is too great. To increase the viability of AEI in a clinical setting, we still need another 10-20 dB improvement to our SNR. We have increased our capabilities to sense the tiny biological signals we're interested in, but the issue of noise needs to be addressed. A robust denoising algorithm is necessary to bring AEI into the spotlight. We believe machine learning (ML) may be able to provide the denoising capabilities we are searching for.

1.5   Proposed Solution

1.5.1   Why ML?

For the last decade, ML has demonstrated unprecedented versatility in tackling image processing tasks, such as denoising, in countless applications. We believe our standard methods of filtering and averaging may be outperformed by a robust ML strategy. Through discussion with collaborators, we've decided to try a Half Instance Regularization Network (HINet) ML framework that would be an excellent starting point. HINet has developed recent acclaim for its performance in image restoration tasks such as denoising, deblurring, and deraining. HINet provides almost state-of-the-art performance in a relatively simple end to end package. A robust deep learning method may just provide the 10-20 dB SNR improvement we need to make AEI a standard modality in clinical settings.

1.5.2 Goal of thesis

Since this project began, it has been my goal to explore ML as a potential denoising method and begin the framework for continued development of ML for AEI. Many ML architectures have been investigated and several have been identified as potential future endeavors. Due to my limited time in the group, a simple but high performing model has been chosen for the purposes of my project. It is the goal of this thesis to investigate ML as a strong candidate for denoising AE images. The methods and results hereafter will act as both a proof of concept and framework for future ML development in the group.

Chapter 2

Methodology and Results

2.1   Machine Learning Methods

2.1.1 ML Theory

Machine Learning has continually and increasingly demonstrated great utility and flexibility in a wide range of problems for the past couple decades. This includes regression problems which the AE denoising problem would be classified. The

framework for a machine learning architecture can be described as a neural network. A neural network is an artificial mathematical model made up of many groups of interconnected units called neurons and is used to approximate a mathematical function through a series of non-linear transformations. Each transformation, or layer, is a matrix function with weights that govern how the input is continually changed as it is fed through the model. It is only through the algorithmic process of training the neural network that the model gains the ability to transform a given input to a desired output.

How network training is implemented largely depends on the type of learning that is being done. In our problem, we're using simulated data with corresponding ground truth data to iteratively calculate loss and update model parameters. This is considered supervised learning. In this case, as well as with other learning types, an optimizer, loss function, and activation function need to be chosen for the initializing the training process. The training process involves fetching a batch of data from a data loader, a forward pass of that batch through the model, evaluating the loss, and back propagating that loss along the gradient of the loss function to update the network weights.
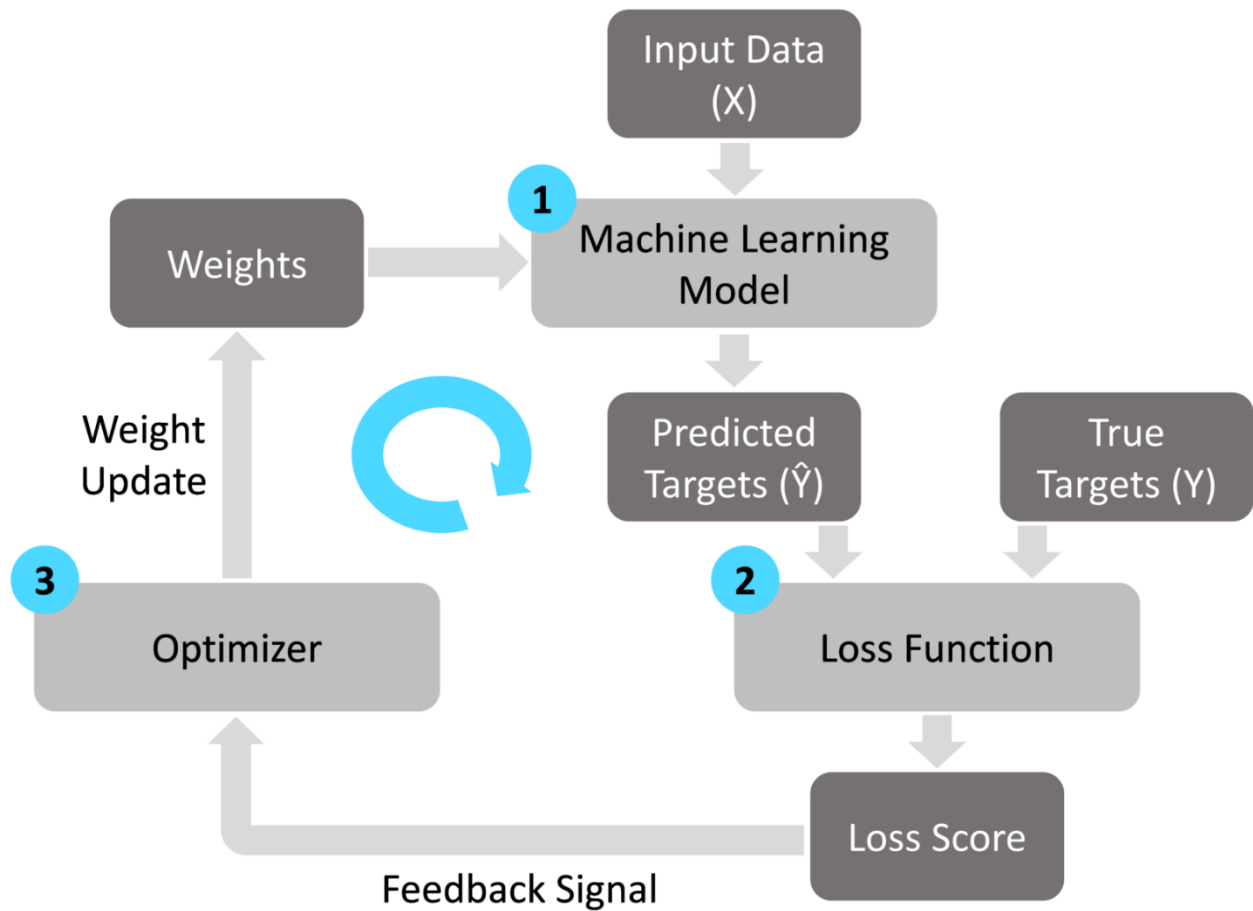
Figure 7: Training loop involves a forward pass through the model, calculating the loss, and backpropagating that loss to update the model weights.

2.1.2   ML solution

Choosing a network architecture is a nontrivial task. Different architectures have pros and cons and will accel in different problems since features are extracted in different ways. EUNIL and collaborators here at the University of Arizona deliberated for quite some time and considered several different architectures. An elegant end-to-end network called Half Instance Normalization Network (HINet) was selected. [3]HINet is an architecture that came out of an image restoration machine learning

competition and has recently been gaining attention for its performance in various

image restoration tasks. HINet is a multi-stage network with two U-Net subnetworks.

The creators proposed a novel Half Instance Normalization (HIN) block. The HIN

block is included in each subnetwork's encoder and improves the robustness of the

extracted features. To further enrich multi-scale features and improve network cross

connections, Cross-Stage Feature Fusion (CSFF) block and Supervised Attention

Module (SAM) are also throughout the network.
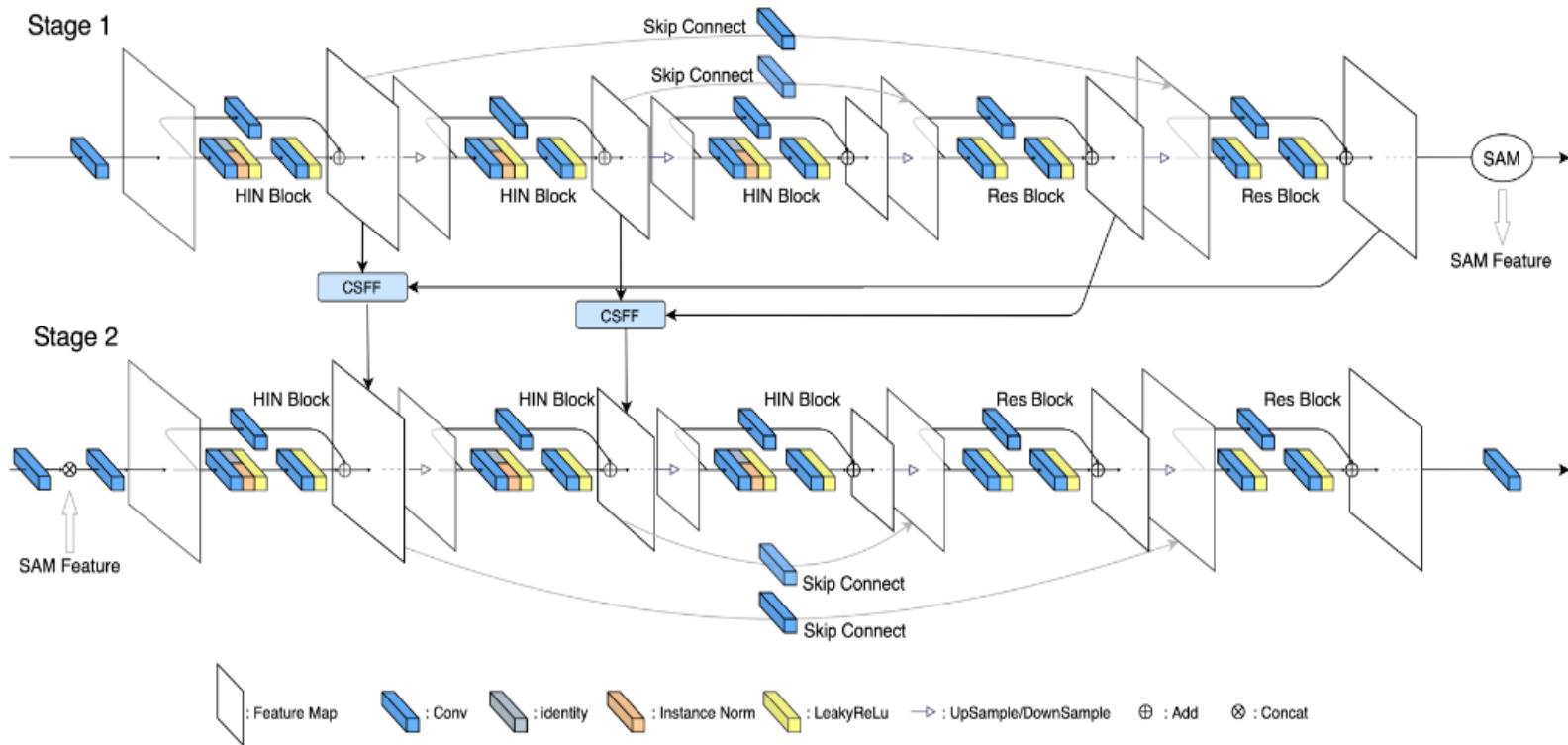
Figure 8: HINet architecture[14]

2.1.3   Activation Function, Loss Function, and Optimizer

Activation functions decide to what degree should an abstract (hidden) layer be

activated. Activation functions act as hidden layer weights and introduce non-linearity

to a neuron. The HINet architecture uses the LeakyReLU activation function in the HIN

block and Res block.
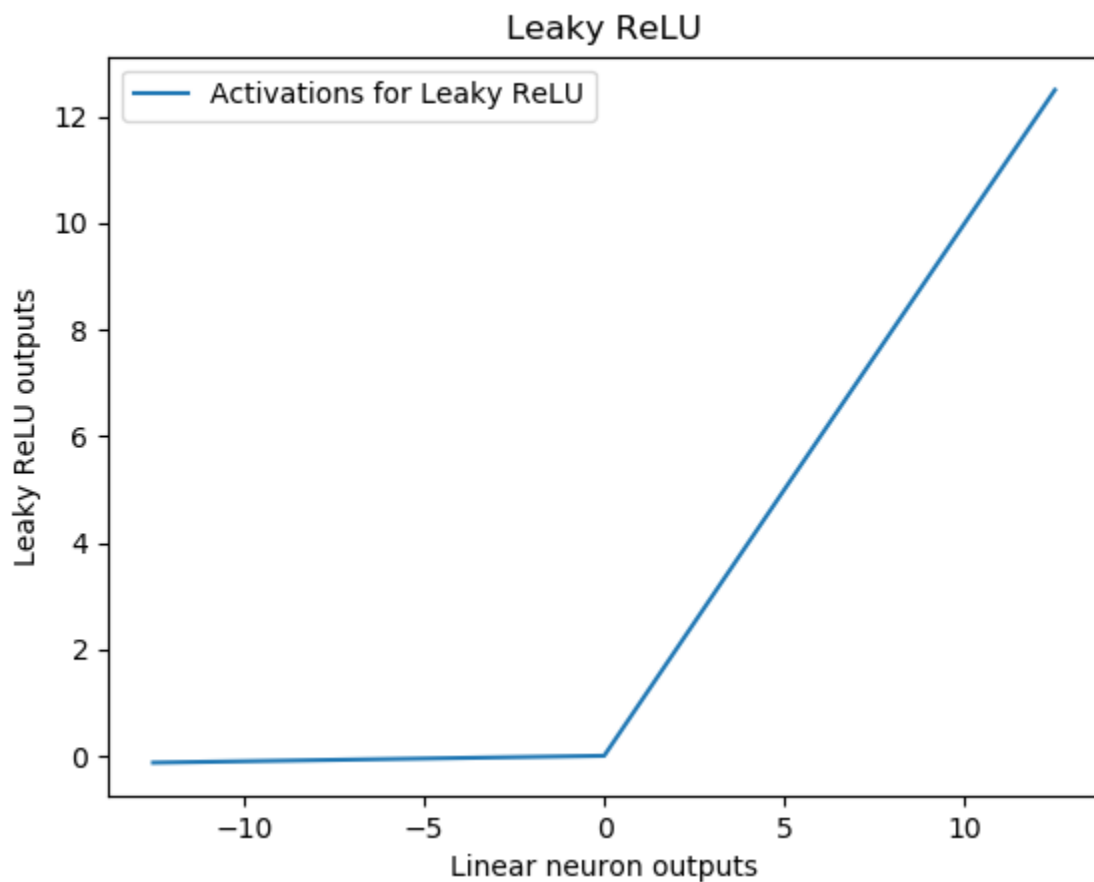
Figure 9: Leaky ReLU has a nonzero slope when less than 0 to reduce the likelihood of hidden layer inactivation.
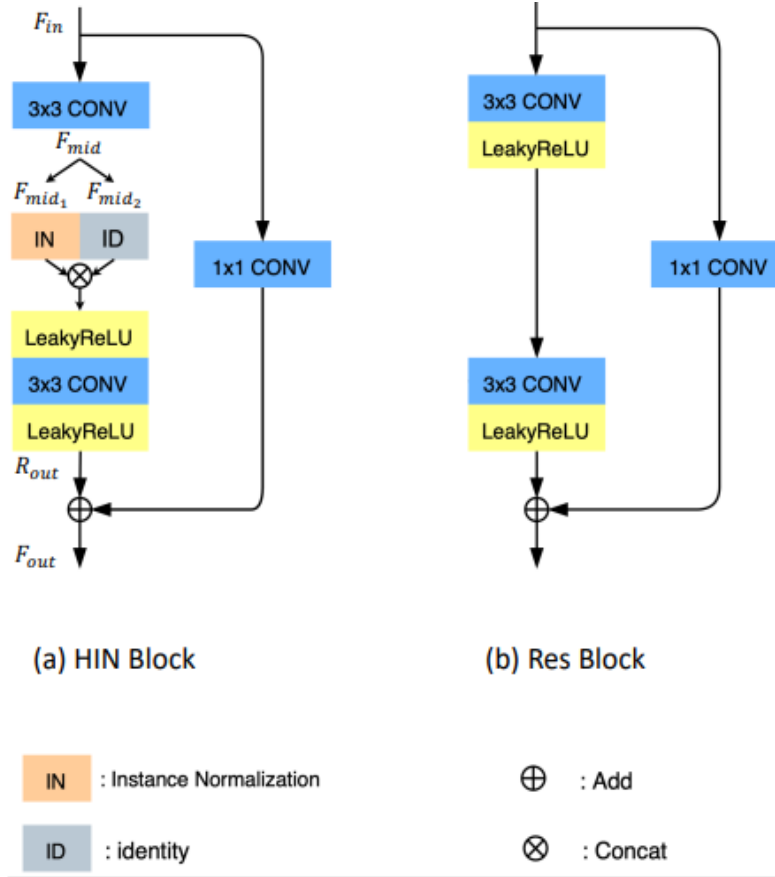
Figure 10: HIN block and Res Block with leaky ReLU activation function[14]

The loss function is a very important choice when designing a machine learning model. Loss functions guide the optimization algorithm during training and directly impact the resultant predictive ability of the model. One of the most used loss functions in regression tasks is Mean Squared Error (MSE), also known as the L2 Loss. MSE calculates the average of the squared difference between the predicted and ground truth values. A crucial aspect to MSE as a loss function is the fact that the value is being squared. Because of this, larger errors are penalized severely in the optimization process.

Another important choice to make when designing a machine learning algorithm

is of course the optimization function. For this problem, an optimization function called

Adaptive Moment Estimation (Adam) was selected.[4] Adam is an extension to

traditional stochastic gradient descent. However, Adam is commonly hailed for

combining advantages from two other stochastic gradient descent methods, Adaptive

Gradient Algorithm (AdaGrad) and Root Mean Square Propagation (RMSProp).

AdaGrad adaptively chooses a learning rate to improve the performance on problems

with sparse gradients. RMSProp adjusts learning rate based on the magnitude of the

most recent gradient calculation making it accel on non-stationary problems. Having

both qualities, Adam is a highly effective and highly efficient optimization algorithm

which has championed its way as a go to in the ML community.

2.2   Data Analysis

2.2.1   *Hyperparameter testing*

There were three hyperparameters in my model that were tested. Epochs, batch

size, and learning rate. Epochs indicate how many times the entire input dataset will be

passed through the model for training. Batch size is a hyperparameter determined by

how the data loader fetches data. The data loader breaks the total dataset up into

batches to be fed into the network one at a time. Training the model in this mini

batched manner helps stability and training performance. The learning rate is a

parameter that helps the optimizer determine how much to tweak a neuron's weight during training. A learning rate that is too large will result in training instability where a local minimum of the loss function is not approached, and the loss will bounce around. Decreasing the learning rate too much will result in extraordinarily large training times. A balance between these three hyperparameters needs to be made to monitor model fitting.

2.2.2 *Monitoring model fitting (training loss vs test loss)*

The neural network model, like all trained models, is susceptible to underfitting and overfitting.[5] The model fitting can be loosely determined by how the training and test loss behave during training. Underfitting would be due to the loss function's minima not being reached before training is complete. This would look like the training and test loss to continue trending down at the end of training. Overfitting would be due to the training dataset being "memorized" by the network resulting in the model not generalizing to the test set which was not used to adjust the model weights.

2.2.3 *Image Analysis Metrics*

The primary image analysis metrics which are used to determine the performance of the model are Peak Signal-to-Noise Ratio (PSNR) and our loss function metric MSE. These metrics are defined as follows:

$$PSNR = 10\log_{10}\left(\frac{MAX_I^2}{MSE}\right)$$

$$MSE = \frac{1}{mn}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}[I(i,j) - K(i,j)]^2$$

PSNR is a widely adopted metric for comparing reconstruction quality. We will be using PSNR to compare our network denoising capabilities to our traditional methods of averaging.

2.3 Results

2.3.1 Training Loss Vs. Test Loss

Many combinations of hyperparameters were tested when training the network. Too many epochs or too high of a learning rate often resulted in training instability where the loss and validation loss would suddenly explode. The best training results I was able to obtain were with epochs=6, learning rate=0.0005, and batch size of 25. The training losses were collected and plotted as follows:

```
Epoch [1/6], Loss: 0.0003603249788284302, Val Loss: 0.0005005234270356596
Epoch [2/6], Loss: 0.00014489305613096803, Val Loss: 0.000203664632286336511
Epoch [3/6], Loss: 8.349707059096545e-05, Val Loss: 0.00011520738189574331
Epoch [4/6], Loss: 5.3292384109222699e-05, Val Loss: 7.020289194770157e-05
Epoch [5/6], Loss: 3.74910923710555856e-05, Val Loss: 5.1808045100018788e-05
Epoch [6/6], Loss: 2.766924080788158e-05, Val Loss: 3.6705601814901456e-05
```

Figure 11: The Loss and Validation Loss were printed in the command window.
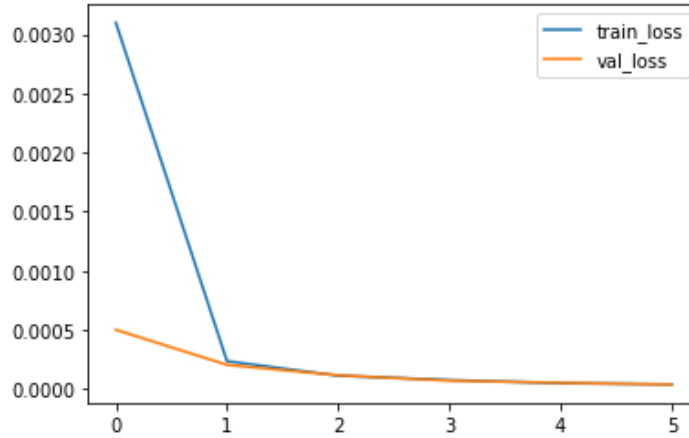
Figure 12: The collected Loss and Validation loss were collected and plotted to demonstrate that overfitting is not taking place.

When any more epochs were added, both losses began to grow.

2.3.2   Results with Optimized Hyperparameters Vs. Traditional Methods

Finally, tested the network results against our traditional method of averaging multiple trials. The left column shows examples of three different levels of gaussian noise added to the clean AE image. The center column shows our traditional method of averaging 100 trials. The averaged trials in this case had the same underlying signal, same noise level, but different realizations of the same noise. The right column is the network output when the noisy images in the left column are input into the HINet model we just trained.

33
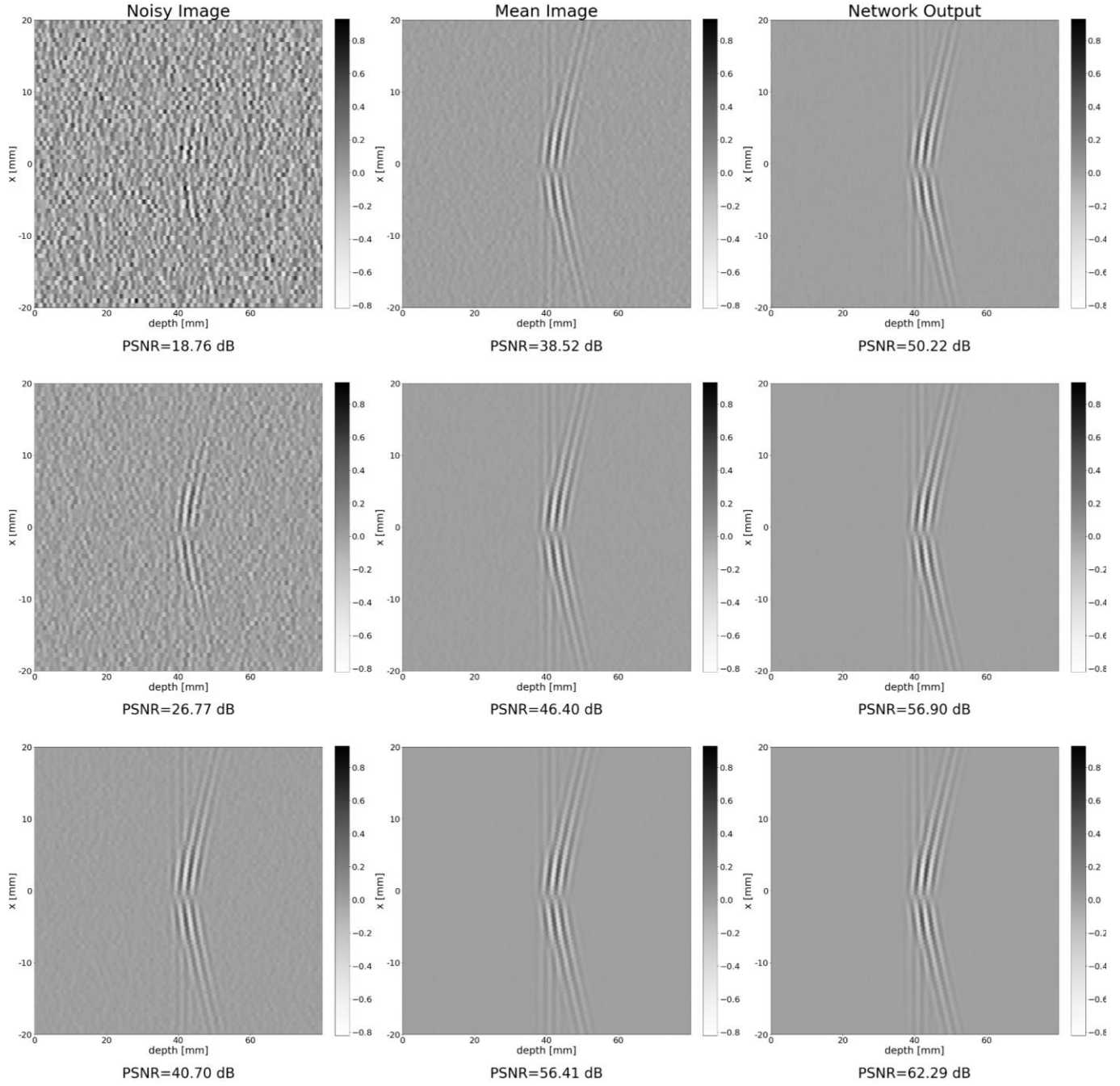
Figure 13: Left column are the noisy images with three different levels of gaussian noise as well as the input to HINet. Center column is the result of averaging 100 trials of the images with different noise realizations. The right column is the output to the network.

Chapter 3

Conclusions and Future Work

3.1   Discussion

3.1.1   *What observations can be drawn from results?*

The HINet outperformed the traditional methods. This is especially surprising since the mean image required 100 trials to achieve that denoising level, and the neural network only took in the single noisy image to produce a far greater denoising level for all the tested noise levels.

3.1.2   *Does the work provide reason to focus ML pursuit?*

The results do provide solid reasoning for focusing EUNIL's ML pursuits for AE denoising. The ML result greatly surpassed expectations. The network was only trained on gaussian noise images. We believe that ML will also be able to handle other noise sources that we experience in a laboratory setting such as spike noise. These other noise sources are also where the traditional methods especially break down, we would expect an even greater improvement in performance with an ML model over the traditional denoising methods in such cases.

3.2   Future Work

3.2.1   *More expansive data simulations*

The ML model that was trained will not generalize well. The simulated dataset was representative of a single physical experiment. This makes the simulated dataset limited and introduces bias in the training process. This is perhaps why the model performed so extraordinarily. To remove this bias, many different physics models need to be represented in the dataset. This would include different AE signals, different recording lead geometries, different ultrasound fields, and different noise sources.

3.2.2   *Feature engineering*

Feature engineering is the process of transforming raw data with specific features into a form that the model can better learn from. The goal is to drive model performance by passing in particularly meaningful and relevant information. In other words, feature engineering is the process of selecting, extracting, and transforming the most critical features from the available data to streamline training efficiency and accuracy. For the AE denoising problem, this could be particularly fruitful since the traditional denoising methods are believed to destroy some low-level information in the raw AE data.

3.2.3   Unsupervised methods (statistical learning)

We defined supervised learning as using ground truth data to guide the training process. Unsupervised learning finds patterns in the data without the ground truth data. Unsupervised learning is often used to solve various clustering problems. So, while this wouldn't be a denoising network, it may help with our small signal detection problem that denoising is attempting to tackle. Unsupervised learning may be used to locate and extract a small AE signal in a large noise floor as they accel in providing essential features in medical imaging and computer vision tasks. Unsupervised learning methods typically rely on iterative *apriori* and dimensionality reduction algorithms along with an autoencoder. An unsupervised learning method may be ideal for EUNIL because we have a vast archive of AE data without ground truth.

3.2.4   Physics-informed ML

In general, supervised learning methods do not generalize well. This is because these methods often overfit the hardware being modeled. Generalizability is critical to deploy these models in the real world. Physics-informed ML (PIML) may be able to provide this desirable trait. PIML is like unsupervised learning in that it requires applying *apriori* knowledge to solve the task. However, PIML utilizes knowledge of physics being modeled to drive performance. PIML enforces physical consistency and

therefore does not hallucinate when presented with never-before-seen data. PIML is

also a single inference process and therefore can be deployed in real time.

Appendix A – Instructions and Source Code

 The distributions for the python packages are managed using Anaconda. A conda

environment with following requirements.txt can be installed from the anaconda

powershell prompt command window with:

```
# using pip
pip install -r requirements.txt

# using Conda
conda create --name <env_name> --file requirements.txt
```

Requirements.txt

```
absl-py==1.4.0
addict @ file:///home/conda/feedstock_root/build_artifacts/addict_1636818143388/work
alabaster==0.7.12
appdirs==1.4.4
argh==0.26.2
astroid @ file:///C:/ci/astroid_1628063237844/work
atomicwrites==1.4.0
attrs @ file:///opt/conda/conda-bld/attrs_1642510447205/work
autopep8 @ file:///tmp/build/80754af9/autopep8_1615918855173/work
Babel @ file:///tmp/build/80754af9/babel_1620871417480/work
backcall @ file:///home/ktietz/src/ci/backcall_1611930011877/work
# Editable install with no version control (basicsr==1.2.0+unknown)
-e c:\users\emnel\documents\hinet-main
bcrypt @ file:///C:/ci/bcrypt_1597936262193/work
black==19.3b0
bleach @ file:///opt/conda/conda-bld/bleach_1641577558959/work
brotlipy==0.7.0
cachetools==4.2.4
certifi==2021.5.30
cffi @ file:///C:/ci/cffi_1625831763871/work
chardet @ file:///C:/ci/chardet_1607706912142/work
charset-normalizer @ file:///tmp/build/80754af9/charset-normalizer_1630003229654/work
click==8.0.3
```

cloudpickle @ file:///tmp/build/80754af9/cloudpickle_1632508026186/work
colorama @ file:///tmp/build/80754af9/colorama_1607707115595/work
cryptography @ file:///C:/ci/cryptography_1635366724772/work
cycler @ file:///tmp/build/80754af9/cycler_1637851556182/work
cytoolz==0.11.0
dask @ file:///tmp/build/80754af9/dask-core_1615055117017/work
dataclasses==0.8
decorator @ file:///opt/conda/conda-bld/decorator_1643638310831/work
defusedxml @ file:///tmp/build/80754af9/defusedxml_1615228127516/work
diff-match-patch @ file:///Users/ktietz/demo/mc3/conda-bld/diff-match-patch_1630511840874/work
docutils @ file:///C:/ci/docutils_1620828217025/work
entrypoints==0.4
flake8 @ file:///tmp/build/80754af9/flake8_1615834841867/work
future==0.18.2
google-auth==2.22.0
google-auth-oauthlib==0.4.6
grpcio==1.48.2
h5py==2.10.0
idna @ file:///tmp/build/80754af9/idna_1637925883363/work
imageio @ file:///tmp/build/80754af9/imageio_1617700267927/work
imagesize @ file:///tmp/build/80754af9/imagesize_1637939814114/work
importlib-metadata==4.8.3
intervaltree @ file:///Users/ktietz/demo/mc3/conda-bld/intervaltree_1630511889664/work
ipykernel==5.5.6
ipython @ file:///C:/ci/ipython_1593446240034/work
ipython-genutils @ file:///tmp/build/80754af9/ipython_genutils_1606773439826/work
isort @ file:///tmp/build/80754af9/isort_1628603791788/work
jedi @ file:///C:/ci/jedi_1606914470086/work
Jinja2 @ file:///opt/conda/conda-bld/jinja2_1647436528585/work
jsonschema @ file:///Users/ktietz/demo/mc3/conda-bld/jsonschema_1630511932244/work
jupyter-client==7.1.2
jupyter-core==4.9.2
keyring @ file:///C:/ci/keyring_1629321701302/work
kiwisolver @ file:///C:/ci/kiwisolver_1612282446297/work
lazy-object-proxy @ file:///C:/ci/lazy-object-proxy_1616529300868/work
Markdown==3.3.7
MarkupSafe @ file:///C:/ci/markupsafe_1621528313524/work
matplotlib @ file:///C:/ci/matplotlib-suite_1613408055530/work
mccabe==0.6.1
mistune==0.8.4
mkl-fft==1.3.0
mkl-random==1.1.1
mkl-service==2.3.0
nbconvert==5.6.1
nbformat @ file:///tmp/build/80754af9/nbformat_1617383369282/work
nest-asyncio==1.6.0
networkx @ file:///tmp/build/80754af9/networkx_1598376031484/work

numpy==1.19.5

numpydoc @ file:///tmp/build/80754af9/numpydoc_1605117425582/work

oauthlib==3.2.2

olefile==0.46

packaging @ file:///tmp/build/80754af9/packaging_1637314298585/work

pandocfilters @ file:///opt/conda/conda-bld/pandocfilters_1643405455980/work

paramiko @ file:///opt/conda/conda-bld/paramiko_1640109032755/work

parso==0.7.0

pexpect @ file:///tmp/build/80754af9/pexpect_1605563209008/work

pickleshare @ file:///tmp/build/80754af9/pickleshare_1606932040724/work

Pillow @ file:///C:/ci/pillow_1625663293114/work

pluggy @ file:///C:/ci/pluggy_1633715371909/work

prompt-toolkit @ file:///tmp/build/80754af9/prompt-toolkit_1633440160888/work

protobuf==3.19.6

psutil @ file:///C:/ci/psutil_1612298125479/work

ptyprocess @ file:///tmp/build/80754af9/ptyprocess_1609355006118/work/dist/ptyprocess-0.7.0-py2.py3-none-any.whl

pyasn1==0.5.1

pyasn1-modules==0.3.0

pycodestyle @ file:///home/ktietz/src/ci_mi/pycodestyle_1612807597675/work

pycparser @ file:///tmp/build/80754af9/pycparser_1636541352034/work

pyDeprecate==0.3.2

pydocstyle @ file:///tmp/build/80754af9/pydocstyle_1621600989141/work

pyflakes @ file:///home/ktietz/src/ci_ipy2/pyflakes_1612551159640/work

Pygments @ file:///opt/conda/conda-bld/pygments_1644249106324/work

pylint @ file:///C:/ci/pylint_1627536908981/work

pyls-black @ file:///tmp/build/80754af9/pyls-black_1607553132291/work

pyls-spyder @ file:///tmp/build/80754af9/pyls-spyder_1613849700860/work

PyNaCl @ file:///C:/ci/pynacl_1595009245871/work

pyOpenSSL @ file:///opt/conda/conda-bld/pyopenssl_1643788558760/work

pyparsing @ file:///tmp/build/80754af9/pyparsing_1635766073266/work

pyreadline==2.1

pyrsistent @ file:///C:/ci/pyrsistent_1600141799440/work

PySocks @ file:///C:/ci/pysocks_1605305839978/work

python-dateutil @ file:///tmp/build/80754af9/python-dateutil_1626374649649/work

python-jsonrpc-server @ file:///tmp/build/80754af9/python-jsonrpc-server_1600278539111/work

python-language-server @ file:///tmp/build/80754af9/python-language-server_1607972495879/work

pytz==2021.3

PyWavelets @ file:///C:/ci/pywavelets_1601658410782/work

pywin32==305

pywin32-ctypes==0.2.0

PyYAML==5.4.1

pyzmq==25.1.2

QDarkStyle==2.8.1

QtAwesome @ file:///tmp/build/80754af9/qtawesome_1637160816833/work

qtconsole @ file:///opt/conda/conda-bld/qtconsole_1643819126524/work

QtPy @ file:///opt/conda/conda-bld/qtpy_1649073884068/work

requests @ file:///opt/conda/conda-bld/requests_1641824580448/work
requests-oauthlib==1.3.1
rope @ file:///opt/conda/conda-bld/rope_1643788605236/work
rsa==4.9
Rtree @ file:///C:/ci/rtree_1618421017076/work
scikit-image==0.17.2
scipy @ file:///C:/ci/scipy_1597675683670/work
six @ file:///tmp/build/80754af9/six_1644875935023/work
snowballstemmer @ file:///tmp/build/80754af9/snowballstemmer_1637937080595/work
sortedcontainers @ file:///tmp/build/80754af9/sortedcontainers_1623949099177/work
Sphinx==4.2.0
sphinxcontrib-applehelp @ file:///home/ktietz/src/ci/sphinxcontrib-applehelp_1611920841464/work
sphinxcontrib-devhelp @ file:///home/ktietz/src/ci/sphinxcontrib-devhelp_1611920923094/work
sphinxcontrib-htmlhelp @ file:///tmp/build/80754af9/sphinxcontrib-htmlhelp_1623945626792/work
sphinxcontrib-jsmath @ file:///home/ktietz/src/ci/sphinxcontrib-jsmath_1611920942228/work
sphinxcontrib-qthelp @ file:///home/ktietz/src/ci/sphinxcontrib-qthelp_1611921055322/work
sphinxcontrib-serializinghtml @ file:///tmp/build/80754af9/sphinxcontrib-serializinghtml_1624451540180/work
spyder @ file:///C:/ci/spyder_1616775991046/work
spyder-kernels==2.2.0
tb-nightly==2.11.0a20220816
tensorboard-data-server==0.6.1
tensorboard-plugin-wit==1.8.1
testpath @ file:///tmp/build/80754af9/testpath_1624638946665/work
textdistance @ file:///tmp/build/80754af9/textdistance_1612461398012/work
three-merge @ file:///tmp/build/80754af9/three-merge_1607553261110/work
tifffile==2020.10.1
toml @ file:///tmp/build/80754af9/toml_1616166611790/work
toolz @ file:///tmp/build/80754af9/toolz_1636545406491/work
torch==1.8.0+cu111
torchaudio==0.8.0
torchmetrics==0.8.2
torchvision==0.9.0+cu111
tornado @ file:///C:/ci/tornado_1606942379977/work
tqdm @ file:///opt/conda/conda-bld/tqdm_1647339053476/work
traitlets @ file:///C:/ci/traitlets_1632759765830/work
typed-ast @ file:///C:/ci/typed-ast_1624953776872/work
typing_extensions @ file:///opt/conda/conda-bld/typing_extensions_1647553014482/work
ujson @ file:///C:/ci/ujson_1611259572767/work
urllib3 @ file:///opt/conda/conda-bld/urllib3_1643638302206/work
watchdog @ file:///C:/ci/watchdog_1612471247473/work
wcwidth @ file:///Users/ktietz/demo/mc3/conda-bld/wcwidth_1629357192024/work
webencodings==0.5.1
Werkzeug==2.0.3
win-inet-pton @ file:///C:/ci/win_inet_pton_1605306197271/work
wincertstore==0.2
wrapt==1.12.1

The code was then ran using the Spyder API.

The code used to load the data and train the model is hinet_arch.py.

Hinet_arch.py

HINet: Half Instance Normalization Network for Image Restoration

```
@inproceedings{chen2021hinet,
  title={HINet: Half Instance Normalization Network for Image Restoration},
  author={Liangyu Chen and Xin Lu and Jie Zhang and Xiaojie Chu and Chengpeng Chen},
  booktitle={IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops},
  year={2021}
}
'''
# Load required libraries
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import TensorDataset, DataLoader
import numpy as np
import h5py
import matplotlib.pyplot as plt

def force_cudnn_initialization():
    s = 32
    dev = torch.device('cuda')
    torch.nn.functional.conv2d(torch.zeros(s, s, s, s, device=dev), torch.zeros(s, s, s, s, device=dev))

# Define common layers used in HINet architecture
def conv3x3(in_chn, out_chn, bias=True):
    layer = nn.Conv2d(in_chn, out_chn, kernel_size=3, stride=1, padding=1, bias=bias)
    return layer

def conv_down(in_chn, out_chn, bias=False):
    layer = nn.Conv2d(in_chn, out_chn, kernel_size=4, stride=2, padding=1, bias=bias)
    return layer

def conv(in_channels, out_channels, kernel_size, bias=False, stride = 1):
    return nn.Conv2d(
        in_channels, out_channels, kernel_size,
        padding=(kernel_size//2), bias=bias, stride = stride)
```

```python
## Supervised Attention Module
class SAM(nn.Module):
    def __init__(self, n_feat, kernel_size=3, bias=True):
        super(SAM, self).__init__()
        self.conv1 = conv(n_feat, n_feat, kernel_size, bias=bias)
        self.conv2 = conv(n_feat, 3, kernel_size, bias=bias)
        self.conv3 = conv(3, n_feat, kernel_size, bias=bias)

    def forward(self, x, x_img):
        x1 = self.conv1(x)
        img = self.conv2(x) + x_img
        x2 = torch.sigmoid(self.conv3(img))
        x1 = x1*x2
        x1 = x1+x
        return x1, img

# Define HINet Architecture
class HINet(nn.Module):

    def __init__(self, in_chn=4, wf=64, depth=3, relu_slope=0.2, hin_position_left=0, hin_position_right=4):
        super(HINet, self).__init__()
        self.depth = depth
        self.down_path_1 = nn.ModuleList()
        self.down_path_2 = nn.ModuleList()
        self.conv_01 = nn.Conv2d(in_chn, wf, 3, 1, 1)
        self.conv_02 = nn.Conv2d(in_chn, wf, 3, 1, 1)

        prev_channels = self.get_input_chn(wf)
        for i in range(depth): #0,1,2,3,4
            use_HIN = True if hin_position_left <= i and i <= hin_position_right else False
            downsample = True if (i+1) < depth else False
            self.down_path_1.append(UNetConvBlock(prev_channels, (2**i) * wf, downsample, relu_slope,
use_HIN=use_HIN))
            self.down_path_2.append(UNetConvBlock(prev_channels, (2**i) * wf, downsample, relu_slope,
use_csff=downsample, use_HIN=use_HIN))
            prev_channels = (2**i) * wf

        self.up_path_1 = nn.ModuleList()
        self.up_path_2 = nn.ModuleList()
        self.skip_conv_1 = nn.ModuleList()
        self.skip_conv_2 = nn.ModuleList()
        for i in reversed(range(depth - 1)):
            self.up_path_1.append(UNetUpBlock(prev_channels, (2**i)*wf, relu_slope))
            self.up_path_2.append(UNetUpBlock(prev_channels, (2**i)*wf, relu_slope))
            self.skip_conv_1.append(nn.Conv2d((2**i)*wf, (2**i)*wf, 3, 1, 1))
            self.skip_conv_2.append(nn.Conv2d((2**i)*wf, (2**i)*wf, 3, 1, 1))
```

```python
            prev_channels = (2**i)*wf
        self.sam12 = SAM(prev_channels)
        self.cat12 = nn.Conv2d(prev_channels*2, prev_channels, 1, 1, 0)

        self.last = conv3x3(prev_channels, in_chn, bias=True)

    def forward(self, x):
        image = x
        #stage 1
        x1 = self.conv_01(image)
        encs = []
        decs = []
        for i, down in enumerate(self.down_path_1):
            if (i+1) < self.depth:
                x1, x1_up = down(x1)
                encs.append(x1_up)
            else:
                x1 = down(x1)

        for i, up in enumerate(self.up_path_1):
            x1 = up(x1, self.skip_conv_1[i](encs[-i-1]))
            decs.append(x1)

        sam_feature, out_1 = self.sam12(x1, image)
        #stage 2
        x2 = self.conv_02(image)
        x2 = self.cat12(torch.cat([x2, sam_feature], dim=1))
        blocks = []
        for i, down in enumerate(self.down_path_2):
            if (i+1) < self.depth:
                x2, x2_up = down(x2, encs[i], decs[-i-1])
                blocks.append(x2_up)
            else:
                x2 = down(x2)

        for i, up in enumerate(self.up_path_2):
            x2 = up(x2, self.skip_conv_2[i](blocks[-i-1]))

        out_2 = self.last(x2)
        out_2 = out_2 + image
        return [out_1, out_2]

    def get_input_chn(self, in_chn):
        return in_chn

    def _initialize(self):
        gain = nn.init.calculate_gain('leaky_relu', 0.20)
```

```python
        for m in self.modules():
            if isinstance(m, nn.Conv2d):
                nn.init.orthogonal_(m.weight, gain=gain)
                if not m.bias is None:
                    nn.init.constant_(m.bias, 0)


class UNetConvBlock(nn.Module):
    def __init__(self, in_size, out_size, downsample, relu_slope, use_csff=False, use_HIN=False):
        super(UNetConvBlock, self).__init__()
        self.downsample = downsample
        self.identity = nn.Conv2d(in_size, out_size, 1, 1, 0)
        self.use_csff = use_csff

        self.conv_1 = nn.Conv2d(in_size, out_size, kernel_size=3, padding=1, bias=True)
        self.relu_1 = nn.LeakyReLU(relu_slope, inplace=False)
        self.conv_2 = nn.Conv2d(out_size, out_size, kernel_size=3, padding=1, bias=True)
        self.relu_2 = nn.LeakyReLU(relu_slope, inplace=False)

        if downsample and use_csff:
            self.csff_enc = nn.Conv2d(out_size, out_size, 3, 1, 1)
            self.csff_dec = nn.Conv2d(out_size, out_size, 3, 1, 1)

        if use_HIN:
            self.norm = nn.InstanceNorm2d(out_size//2, affine=True)
        self.use_HIN = use_HIN

        if downsample:
            self.downsample = conv_down(out_size, out_size, bias=False)

    def forward(self, x, enc=None, dec=None):
        out = self.conv_1(x)

        if self.use_HIN:
            out_1, out_2 = torch.chunk(out, 2, dim=1)
            out = torch.cat([self.norm(out_1), out_2], dim=1)
        out = self.relu_1(out)
        out = self.relu_2(self.conv_2(out))

        out += self.identity(x)
        if enc is not None and dec is not None:
            assert self.use_csff
            out = out + self.csff_enc(enc) + self.csff_dec(dec)
        if self.downsample:
            out_down = self.downsample(out)
            return out_down, out
        else:
```

```python
        return out


class UNetUpBlock(nn.Module):
    def __init__(self, in_size, out_size, relu_slope):
        super(UNetUpBlock, self).__init__()
        self.up = nn.ConvTranspose2d(in_size, out_size, kernel_size=2, stride=2, bias=True)
        self.conv_block = UNetConvBlock(in_size, out_size, False, relu_slope)

    def forward(self, x, bridge):
        up = self.up(x)
        out = torch.cat([up, bridge], 1)
        out = self.conv_block(out)
        return out

class Subspace(nn.Module):

    def __init__(self, in_size, out_size):
        super(Subspace, self).__init__()
        self.blocks = nn.ModuleList()
        self.blocks.append(UNetConvBlock(in_size, out_size, False, 0.2))
        self.shortcut = nn.Conv2d(in_size, out_size, kernel_size=1, bias=True)

    def forward(self, x):
        sc = self.shortcut(x)
        for i in range(len(self.blocks)):
            x = self.blocks[i](x)
        return x + sc


class skip_blocks(nn.Module):

    def __init__(self, in_size, out_size, repeat_num=1):
        super(skip_blocks, self).__init__()
        self.blocks = nn.ModuleList()
        self.re_num = repeat_num
        mid_c = 128
        self.blocks.append(UNetConvBlock(in_size, mid_c, False, 0.2))
        for i in range(self.re_num - 2):
            self.blocks.append(UNetConvBlock(mid_c, mid_c, False, 0.2))
        self.blocks.append(UNetConvBlock(mid_c, out_size, False, 0.2))
        self.shortcut = nn.Conv2d(in_size, out_size, kernel_size=1, bias=True)

    def forward(self, x):
        sc = self.shortcut(x)
        for m in self.blocks:
            x = m(x)
```

```python
        return x + sc


if __name__ == "__main__":
    CUDA_LAUNCH_BLOCKING=1
    # Define empty torch array to load data into
    data_in = torch.empty([5000, 3, 66, 277], dtype=torch.float32)
    data_gt = torch.empty([5000, 3, 66, 277], dtype=torch.float32)
    # Define path to dataset.
    path = 'C:\\Users\\emnel\\Documents\\HINet-main\\datasets\\AE MAT Data Files'

    # Parse the ground truth data (depends on how .mat file is saved)
    gt_path = path+'\\AEtestGroundTruth.mat'
    g = h5py.File(gt_path, 'r')
    datagt1 = g.get('nVAE_LE1').value
    datagt1 = np.array(datagt1)
    datagt1 = torch.from_numpy(datagt1)
    datagt2 = g.get('nVAE_LE2').value
    datagt2 = np.array(datagt2)
    datagt2 = torch.from_numpy(datagt2)
    datagt3 = g.get('nVAE_LE3').value
    datagt3 = np.array(datagt3)
    datagt3 = torch.from_numpy(datagt3)

    # Loop through .mat files in directory and parse .mat files.
    for x in range(50):
        # mat = scipy.io.loadmat(path+'\\AEtest'+x+'.mat')
        fpath = path+'\\AEtest'+str(x+1)+'.mat'
        f = h5py.File(fpath, 'r')
        data1 = f.get('totVAE_1').value
        data1 = np.array(data1)
        data1 = torch.from_numpy(data1)
        data_in[100*x:100*x+100, 0, :, :] = data1
        data2 = f.get('totVAE_2').value
        data2 = np.array(data2)
        data2 = torch.from_numpy(data2)
        data_in[100*x:100*x+100, 1, :, :] = data2
        data3 = f.get('totVAE_3').value
        data3 = np.array(data3)
        data3 = torch.from_numpy(data3)
        data_in[100*x:100*x+100, 2, :, :] = data3

        data_gt[100*x:100*x+100, 0, :, :] = datagt1
        data_gt[100*x:100*x+100, 1, :, :] = datagt2
        data_gt[100*x:100*x+100, 2, :, :] = datagt3

    # Shape data to fit through model without clipping
```

```python
data_in = data_in[:,:,0:64,0:272]
data_gt = data_gt[:,:,0:64,0:272]

# Shuffle input and ground truth data in parallel
indices = torch.randperm(data_in.size(0))
data_in = data_in[indices,:,:,:]
data_gt = data_gt[indices,:,:,:]

# Split data into training and test sets (80/20 split)
train_in = data_in[0:4000,:,:,:]
test_in = data_in[4000:5000,:,:,:]
train_gt = data_gt[0:4000,:,:,:]
test_gt = data_gt[4000:5000,:,:,:]


# Create dataloader for trainig set
class SimpleCustomBatch:
    def __init__(self, train_in):
        transposed_data = list(zip(*train_in))
        self.inp = torch.stack(transposed_data[0], 0)
        self.tgt = torch.stack(transposed_data[1], 0)
        # custom memory pinning method on custom type
        def pin_memory(self):
            self.inp = self.inp.pin_memory()
            self.tgt = self.tgt.pin_memory()
            return self

def collate_wrapper(batch):
    return SimpleCustomBatch(batch)

dataset_train = TensorDataset(train_in, train_gt)
train_loader = DataLoader(dataset_train, batch_size=25, collate_fn=collate_wrapper,
                pin_memory=True)

# Create dataloader for test set
class SimpleCustomBatch2:
    def __init__(self, test_in):
        transposed_data = list(zip(*test_in))
        self.inp = torch.stack(transposed_data[0], 0)
        self.tgt = torch.stack(transposed_data[1], 0)
        # custom memory pinning method on custom type
        def pin_memory(self):
            self.inp = self.inp.pin_memory()
            self.tgt = self.tgt.pin_memory()
            return self

def collate_wrapper(batch):
```

```python
        return SimpleCustomBatch2(batch)

dataset_test = TensorDataset(test_in, test_gt)
test_loader = DataLoader(dataset_test, batch_size=25, collate_fn=collate_wrapper,
        pin_memory=True)

# Create an instance of your model
net = HINet( in_chn=3, wf=64, depth=3, relu_slope=0.2, hin_position_left=0, hin_position_right=4)

# Define loss function and optimizer
criterion = nn.MSELoss()
optimizer = optim.Adam(net.parameters(), lr=0.0005)

# Define the number of training parameters
num_epochs = 6
torch.cuda.empty_cache()
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
net.to(device)
# Prepare arrays to save training and validation losses
val_loss = torch.empty(0,dtype=torch.float32)
val_loss = 0
trainingEpoch_loss = []
validationEpoch_loss = []

# Training loop
for epoch in range(num_epochs):
    step_loss = []
    net.train()
    for batch_ndx, sample in enumerate(train_loader):

        # Get the batch and move it to the GPU
        inputs = sample.inp.cuda()
        target = sample.tgt.cuda()

        # Forward pass
        outputs = net(inputs)

        # Compute the loss
        loss = criterion(outputs[0], target)

        # Backward pass and optimization
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        step_loss.append(loss.item())
    trainingEpoch_loss.append(np.array(step_loss).mean())
    net.eval()  # Switch to eval mode for validation
```

```
#%%calculate validation loss
    with torch.no_grad():
        for batch_ndx2, sample2 in enumerate(test_loader):
            validationStep_loss = []
            test_out = net(sample2.inp.cuda())
            test_in = sample2.tgt.cuda()
            validationLoss = criterion(test_out[0],test_in)
            validationStep_loss.append(validationLoss.item())
        temp = np.array(validationStep_loss).mean()
        validationEpoch_loss.append(temp)


    # Print the train loss and validation for each epoch
    print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {loss.item()}, Val Loss: {temp.item()}')
#%%
    plt.plot(trainingEpoch_loss, label='train_loss')
    plt.plot(validationEpoch_loss,label='val_loss')
    plt.legend()
    plt.savefig('TrainLossVsValLoss3.png', bbox_inches='tight')
    plt.show
    # Save your trained model if needed
    torch.save(net.state_dict(), 'your_model3.pth')
```

==To load the saved model and evaluate the performance, run LoadModelPlot.py==

LoadModelPlot.py

```
import torch
import torch.nn as nn
from torchmetrics.image import PeakSignalNoiseRatio
import numpy as np
import h5py
import matplotlib.pyplot as plt

# Define netork architecture
def force_cudnn_initialization():
    s = 32
    dev = torch.device('cuda')
    torch.nn.functional.conv2d(torch.zeros(s, s, s, s, device=dev), torch.zeros(s, s, s, s, device=dev))

def conv3x3(in_chn, out_chn, bias=True):
    layer = nn.Conv2d(in_chn, out_chn, kernel_size=3, stride=1, padding=1, bias=bias)
    return layer

def conv_down(in_chn, out_chn, bias=False):
```

```python
        layer = nn.Conv2d(in_chn, out_chn, kernel_size=4, stride=2, padding=1, bias=bias)
        return layer

def conv(in_channels, out_channels, kernel_size, bias=False, stride = 1):
    return nn.Conv2d(
        in_channels, out_channels, kernel_size,
        padding=(kernel_size//2), bias=bias, stride = stride)


## Supervised Attention Module
class SAM(nn.Module):
    def __init__(self, n_feat, kernel_size=3, bias=True):
        super(SAM, self).__init__()
        self.conv1 = conv(n_feat, n_feat, kernel_size, bias=bias)
        self.conv2 = conv(n_feat, 3, kernel_size, bias=bias)
        self.conv3 = conv(3, n_feat, kernel_size, bias=bias)

    def forward(self, x, x_img):
        x1 = self.conv1(x)
        img = self.conv2(x) + x_img
        x2 = torch.sigmoid(self.conv3(img))
        x1 = x1*x2
        x1 = x1+x
        return x1, img

class HINet(nn.Module):

    def __init__(self, in_chn=4, wf=64, depth=3, relu_slope=0.2, hin_position_left=0, hin_position_right=4):
        super(HINet, self).__init__()
        self.depth = depth
        self.down_path_1 = nn.ModuleList()
        self.down_path_2 = nn.ModuleList()
        self.conv_01 = nn.Conv2d(in_chn, wf, 3, 1, 1)
        self.conv_02 = nn.Conv2d(in_chn, wf, 3, 1, 1)

        prev_channels = self.get_input_chn(wf)
        for i in range(depth): #0,1,2,3,4
            use_HIN = True if hin_position_left <= i and i <= hin_position_right else False
            downsample = True if (i+1) < depth else False
            self.down_path_1.append(UNetConvBlock(prev_channels, (2**i) * wf, downsample, relu_slope,
use_HIN=use_HIN))
            self.down_path_2.append(UNetConvBlock(prev_channels, (2**i) * wf, downsample, relu_slope,
use_csff=downsample, use_HIN=use_HIN))
            prev_channels = (2**i) * wf

        self.up_path_1 = nn.ModuleList()
        self.up_path_2 = nn.ModuleList()
        self.skip_conv_1 = nn.ModuleList()
```

```python
        self.skip_conv_2 = nn.ModuleList()
        for i in reversed(range(depth - 1)):
            self.up_path_1.append(UNetUpBlock(prev_channels, (2**i)*wf, relu_slope))
            self.up_path_2.append(UNetUpBlock(prev_channels, (2**i)*wf, relu_slope))
            self.skip_conv_1.append(nn.Conv2d((2**i)*wf, (2**i)*wf, 3, 1, 1))
            self.skip_conv_2.append(nn.Conv2d((2**i)*wf, (2**i)*wf, 3, 1, 1))
            prev_channels = (2**i)*wf
        self.sam12 = SAM(prev_channels)
        self.cat12 = nn.Conv2d(prev_channels*2, prev_channels, 1, 1, 0)

        self.last = conv3x3(prev_channels, in_chn, bias=True)

    def forward(self, x):
        image = x
        #stage 1
        x1 = self.conv_01(image)
        encs = []
        decs = []
        for i, down in enumerate(self.down_path_1):
            if (i+1) < self.depth:
                x1, x1_up = down(x1)
                encs.append(x1_up)
            else:
                x1 = down(x1)

        for i, up in enumerate(self.up_path_1):
            x1 = up(x1, self.skip_conv_1[i](encs[-i-1]))
            decs.append(x1)

        sam_feature, out_1 = self.sam12(x1, image)
        #stage 2
        x2 = self.conv_02(image)
        x2 = self.cat12(torch.cat([x2, sam_feature], dim=1))
        blocks = []
        for i, down in enumerate(self.down_path_2):
            if (i+1) < self.depth:
                x2, x2_up = down(x2, encs[i], decs[-i-1])
                blocks.append(x2_up)
            else:
                x2 = down(x2)

        for i, up in enumerate(self.up_path_2):
            x2 = up(x2, self.skip_conv_2[i](blocks[-i-1]))

        out_2 = self.last(x2)
        out_2 = out_2 + image
        return [out_1, out_2]
```

```python
    def get_input_chn(self, in_chn):
        return in_chn

    def _initialize(self):
        gain = nn.init.calculate_gain('leaky_relu', 0.20)
        for m in self.modules():
            if isinstance(m, nn.Conv2d):
                nn.init.orthogonal_(m.weight, gain=gain)
                if not m.bias is None:
                    nn.init.constant_(m.bias, 0)


class UNetConvBlock(nn.Module):
    def __init__(self, in_size, out_size, downsample, relu_slope, use_csff=False, use_HIN=False):
        super(UNetConvBlock, self).__init__()
        self.downsample = downsample
        self.identity = nn.Conv2d(in_size, out_size, 1, 1, 0)
        self.use_csff = use_csff

        self.conv_1 = nn.Conv2d(in_size, out_size, kernel_size=3, padding=1, bias=True)
        self.relu_1 = nn.LeakyReLU(relu_slope, inplace=False)
        self.conv_2 = nn.Conv2d(out_size, out_size, kernel_size=3, padding=1, bias=True)
        self.relu_2 = nn.LeakyReLU(relu_slope, inplace=False)

        if downsample and use_csff:
            self.csff_enc = nn.Conv2d(out_size, out_size, 3, 1, 1)
            self.csff_dec = nn.Conv2d(out_size, out_size, 3, 1, 1)

        if use_HIN:
            self.norm = nn.InstanceNorm2d(out_size//2, affine=True)
        self.use_HIN = use_HIN

        if downsample:
            self.downsample = conv_down(out_size, out_size, bias=False)

    def forward(self, x, enc=None, dec=None):
        out = self.conv_1(x)

        if self.use_HIN:
            out_1, out_2 = torch.chunk(out, 2, dim=1)
            out = torch.cat([self.norm(out_1), out_2], dim=1)
        out = self.relu_1(out)
        out = self.relu_2(self.conv_2(out))

        out += self.identity(x)
        if enc is not None and dec is not None:
```

```python
            assert self.use_csff
            out = out + self.csff_enc(enc) + self.csff_dec(dec)
        if self.downsample:
            out_down = self.downsample(out)
            return out_down, out
        else:
            return out


class UNetUpBlock(nn.Module):
    def __init__(self, in_size, out_size, relu_slope):
        super(UNetUpBlock, self).__init__()
        self.up = nn.ConvTranspose2d(in_size, out_size, kernel_size=2, stride=2, bias=True)
        self.conv_block = UNetConvBlock(in_size, out_size, False, relu_slope)

    def forward(self, x, bridge):
        up = self.up(x)
        out = torch.cat([up, bridge], 1)
        out = self.conv_block(out)
        return out

class Subspace(nn.Module):

    def __init__(self, in_size, out_size):
        super(Subspace, self).__init__()
        self.blocks = nn.ModuleList()
        self.blocks.append(UNetConvBlock(in_size, out_size, False, 0.2))
        self.shortcut = nn.Conv2d(in_size, out_size, kernel_size=1, bias=True)

    def forward(self, x):
        sc = self.shortcut(x)
        for i in range(len(self.blocks)):
            x = self.blocks[i](x)
        return x + sc


class skip_blocks(nn.Module):

    def __init__(self, in_size, out_size, repeat_num=1):
        super(skip_blocks, self).__init__()
        self.blocks = nn.ModuleList()
        self.re_num = repeat_num
        mid_c = 128
        self.blocks.append(UNetConvBlock(in_size, mid_c, False, 0.2))
        for i in range(self.re_num - 2):
            self.blocks.append(UNetConvBlock(mid_c, mid_c, False, 0.2))
        self.blocks.append(UNetConvBlock(mid_c, out_size, False, 0.2))
```

```python
        self.shortcut = nn.Conv2d(in_size, out_size, kernel_size=1, bias=True)

    def forward(self, x):
        sc = self.shortcut(x)
        for m in self.blocks:
            x = m(x)
        return x + sc

if __name__ == "__main__":
    # Load data as before
    data_in = torch.empty([5000, 3, 66, 277], dtype=torch.float32)
    data_gt = torch.empty([5000, 3, 66, 277], dtype=torch.float32)
    path = 'C:\\Users\\emnel\\Documents\\HINet-main\\datasets\\AE MAT Data Files'

    gt_path = path+'\\AEtestGroundTruth.mat'
    g = h5py.File(gt_path, 'r')
    datagt1 = g.get('nVAE_LE1').value
    datagt1 = np.array(datagt1)
    datagt1 = torch.from_numpy(datagt1)
    datagt2 = g.get('nVAE_LE2').value
    datagt2 = np.array(datagt2)
    datagt2 = torch.from_numpy(datagt2)
    datagt3 = g.get('nVAE_LE3').value
    datagt3 = np.array(datagt3)
    datagt3 = torch.from_numpy(datagt3)

    for x in range(50):
        fpath = path+'\\AEtest'+str(x+1)+'.mat'
        f = h5py.File(fpath, 'r')
        data1 = f.get('totVAE_1').value
        data1 = np.array(data1)
        data1 = torch.from_numpy(data1)
        data_in[100*x:100*x+100, 0, :, :] = data1
        data2 = f.get('totVAE_2').value
        data2 = np.array(data2)
        data2 = torch.from_numpy(data2)
        data_in[100*x:100*x+100, 1, :, :] = data2
        data3 = f.get('totVAE_3').value
        data3 = np.array(data3)
        data3 = torch.from_numpy(data3)
        data_in[100*x:100*x+100, 2, :, :] = data3

        data_gt[100*x:100*x+100, 0, :, :] = datagt1
        data_gt[100*x:100*x+100, 1, :, :] = datagt2
        data_gt[100*x:100*x+100, 2, :, :] = datagt3

    data_in = data_in[:,:,0:64,0:272]
```

```python
    data_gt = data_gt[:,:,0:64,0:272]
    GroundTruth = data_gt[0:99,:,:,:]
    GroundTruthAvg = torch.empty([3, 64, 272], dtype=torch.float32)
    GroundTruthAvg[0,:,:] = data_gt[0,0,:,:]
    GroundTruthAvg[1,:,:] = data_gt[0,1,:,:]
    GroundTruthAvg[2,:,:] = data_gt[0,2,:,:]


    # Specify the path to your saved model
    PATH = 'C:\\Users\\emnel\\Documents\\HINet-
main\\basicsr\\models\\archs\\your_model.pth'

    # Create an instance of your model
    net = HINet( in_chn=3, wf=64, depth=3, relu_slope=0.2, hin_position_left=0, hin_position_right=4)

    # Load the state dictionary from the saved file (loaded the trained model weights into the architecture)
    net.load_state_dict(torch.load(PATH))

    # Set the model to evaluation mode
    net.eval()

    with torch.no_grad():
        # Select the data to be evaluated
        test_in1 = data_in[0:99,:,:,:]
        # Run the input data through the model
        a = net(test_in1)
        test_out1 = a[0]

        test_in2 = data_in[1000:1099,:,:,:]
        b = net(test_in2)
        test_out2 = b[0]
        test_in3 = data_in[2000:2099,:,:,:]
        c = net(test_in3)
        test_out3 = c[0]
        test_in4 = data_in[3000:3099,:,:,:]
        d = net(test_in4)
        test_out4 = d[0]
        test_in5 = data_in[4000:4099,:,:,:]
        e = net(test_in5)
        test_out5 = e[0]

    # Calculate mean images to test against network output
    avg_in1 = torch.mean(data_in[0:99,:,:,:], dim=0)
    avg_in2 = torch.mean(data_in[1000:1099,:,:,:], dim=0)
    avg_in3 = torch.mean(data_in[2000:2099,:,:,:], dim=0)
    avg_in4 = torch.mean(data_in[3000:3099,:,:,:], dim=0)
    avg_in5 = torch.mean(data_in[4000:4099,:,:,:], dim=0)
```

```
psnr = PeakSignalNoiseRatio()

# Calculate and print the PSNR for the input, mean image, and network output.
psnr_in1 = psnr(test_in1,GroundTruth)
psnr_avg1 = psnr(avg_in1,GroundTruthAvg)
psnr_out1 = psnr(test_out1,GroundTruth)
print(psnr_in1)
print(psnr_avg1)
print(psnr_out1)

psnr_in3 = psnr(test_in3,GroundTruth)
psnr_avg3 = psnr(avg_in3,GroundTruthAvg)
psnr_out3 = psnr(test_out3,GroundTruth)
print(psnr_in3)
print(psnr_avg3)
print(psnr_out3)

psnr_in5 = psnr(test_in5,GroundTruth)
psnr_avg5 = psnr(avg_in5,GroundTruthAvg)
psnr_out5 = psnr(test_out5,GroundTruth)
print(psnr_in5)
print(psnr_avg5)
print(psnr_out5)

#%% Plotting the results
plt.rc('xtick', labelsize=25)
plt.rc('ytick', labelsize=25)


f, axarr = plt.subplots(3,3,figsize=(46,46), tight_layout=True)
plt.setp(axarr, xticks=[0,25,50,75], xticklabels=[0, 20, 40,60], yticks=[0, 0.25, 0.5, 0.75,1], yticklabels=[-20, -10,
0, 10, 20])

temp1 = axarr[0,0].imshow(test_in1[0,0,:,:], extent=[0,100,0,1], aspect=100, vmin=-0.82, vmax = 0.93, cmap =
'Greys')
axarr[0,0].set_title("Noisy Image", size = 50)
axarr[0,0].set_xlabel("depth [mm]", size = 30)
axarr[0,0].set_ylabel("x [mm]", size = 30)
cbar_int = f.colorbar(temp1, orientation='vertical', ax=axarr[0,0], fraction=0.046, pad=0.04)
cbar_int.ax.tick_params(labelsize=25)
axarr[0,0].text(50, -0.15, "PSNR=18.76 dB", size=42, ha="center")

temp2 = axarr[0,1].imshow(avg_in1[0,:,:], extent=[0,100,0,1], aspect=100, vmin=-0.82, vmax = 0.93, cmap =
'Greys')
axarr[0,1].set_title("Mean Image", size = 50)
axarr[0,1].set_xlabel("depth [mm]", size = 30)
```

```
axarr[0,1].set_ylabel("x [mm]", size = 30)
cbar_int = f.colorbar(temp2,orientation='vertical', ax=axarr[0,1],fraction=0.046, pad=0.04)
cbar_int.ax.tick_params(labelsize=25)
axarr[0,1].text(50, -0.15, "PSNR=38.52 dB", size=42, ha="center")


temp3 = axarr[0,2].imshow(test_out1[0,0,:,:], extent=[0,100,0,1], aspect=100, vmin=-0.82, vmax = 0.93, cmap
= 'Greys')
axarr[0,2].set_title("Network Output", size = 50)
axarr[0,2].set_xlabel("depth [mm]", size = 30)
axarr[0,2].set_ylabel("x [mm]", size = 30)
cbar_int = f.colorbar(temp3,orientation='vertical', ax=axarr[0,2],fraction=0.046, pad=0.04)
cbar_int.ax.tick_params(labelsize=25)
axarr[0,2].text(50, -0.15, "PSNR=50.22 dB", size=42, ha="center")


temp4 = axarr[1,0].imshow(test_in3[0,0,:,:], extent=[0,100,0,1], aspect=100, vmin=-0.82, vmax = 0.93, cmap =
'Greys')
axarr[1,0].set_xlabel("depth [mm]", size = 30)
axarr[1,0].set_ylabel("x [mm]", size = 30)
cbar_int = f.colorbar(temp4,orientation='vertical', ax=axarr[1,0],fraction=0.046, pad=0.04)
cbar_int.ax.tick_params(labelsize=25)
axarr[1,0].text(50, -0.15, "PSNR=26.77 dB", size=42, ha="center")


temp5 = axarr[1,1].imshow(avg_in3[0,:,:], extent=[0,100,0,1], aspect=100, vmin=-0.82, vmax = 0.93, cmap =
'Greys')
axarr[1,1].set_xlabel("depth [mm]", size = 30)
axarr[1,1].set_ylabel("x [mm]", size = 30)
cbar_int = f.colorbar(temp5,orientation='vertical', ax=axarr[1,1],fraction=0.046, pad=0.04)
cbar_int.ax.tick_params(labelsize=25)
axarr[1,1].text(50, -0.15, "PSNR=46.40 dB", size=42, ha="center")


temp6 = axarr[1,2].imshow(test_out3[0,0,:,:], extent=[0,100,0,1], aspect=100, vmin=-0.82, vmax = 0.93, cmap
= 'Greys')
axarr[1,2].set_xlabel("depth [mm]", size = 30)
axarr[1,2].set_ylabel("x [mm]", size = 30)
cbar_int = f.colorbar(temp6,orientation='vertical', ax=axarr[1,2],fraction=0.046, pad=0.04)
cbar_int.ax.tick_params(labelsize=25)
axarr[1,2].text(50, -0.15, "PSNR=56.90 dB", size=42, ha="center")


temp7 = axarr[2,0].imshow(test_in5[0,0,:,:], extent=[0,100,0,1], aspect=100, vmin=-0.82, vmax = 0.93, cmap =
'Greys')
axarr[2,0].set_xlabel("depth [mm]", size = 30)
axarr[2,0].set_ylabel("x [mm]", size = 30)
cbar_int = f.colorbar(temp7,orientation='vertical', ax=axarr[2,0],fraction=0.046, pad=0.04)
cbar_int.ax.tick_params(labelsize=25)
axarr[2,0].text(50, -0.15, "PSNR=40.70 dB", size=42, ha="center")
```

```
temp8 = axarr[2,1].imshow(avg_in5[0,:,:], extent=[0,100,0,1], aspect=100, vmin=-0.82, vmax = 0.93, cmap =
'Greys')
axarr[2,1].set_xlabel("depth [mm]", size = 30)
axarr[2,1].set_ylabel("x [mm]", size = 30)
cbar_int = f.colorbar(temp8,orientation='vertical', ax=axarr[2,1],fraction=0.046, pad=0.04)
cbar_int.ax.tick_params(labelsize=25)
axarr[2,1].text(50, -0.15, "PSNR=56.41 dB", size=42, ha="center")

temp9 = axarr[2,2].imshow(test_out5[0,0,:,:], extent=[0,100,0,1], aspect=100, vmin=-0.82, vmax = 0.93, cmap
= 'Greys')
axarr[2,2].set_xlabel("depth [mm]", size = 30)
axarr[2,2].set_ylabel("x [mm]", size = 30)
cbar_int = f.colorbar(temp9,orientation='vertical', ax=axarr[2,2],fraction=0.046, pad=0.04)
cbar_int.ax.tick_params(labelsize=25)
axarr[2,2].text(50, -0.15, "PSNR=62.29 dB", size=42, ha="center")

# Save the figure
plt.savefig("example.png",bbox_inches="tight")
```
-- --------------

Citations

[1] Witte, R. S., Olafsson, R., Huang, S. -W., O'Donnell, M., "Imaging current flow in lobster nerve cord using the acoustoelectric effect", 2007 *Appl. Phys. Lett. 90, 163902*, DOI: 10.1063/1.2724901

[2] Li, Q., Olafsson, R., Ingram, P., Wang, Z., Witte, R., *"*Measuring the acoustoelectric interaction constant using ultrasound current source density imaging*"* 2012 *Phys. Med. Biol.* 57 5929, DOI 10.1088/0031-9155/57/19/5929.

[3] Barragan, A., Preston, C., Alvarez, A., Bera, T., Qin, Y., Weinand, M., Kasoff, W., Witte, R. S., "Acoustoelectric imaging of deep dipoles in a human head phantom for guiding treatment of epilepsy", 2020 *J. Neural Eng.* 17 056040. DOI 10.1088/1741-2552/abb63a.

[4] Kang, J., Huang, C., Perkins, C., Alvarez, A., Kunyansky, L., Witte, R.S., O'Donnell, M., "Current Source Density Imaging Using Regularized Inversion of Acoustoelectric Signals", 2022 *IEEE Transactions on Medical Imaging,* Vol. 42, Issue 3, pp. 739-749, DOI: 10.1109/TMI.2022.3215748

[5] Barragan, A., Preston, C., Alvarez, A., Ingram, C. P., Kanti Bera, T., and Witte, R. S., "4D Transcranial Acoustoelectric Imaging of Current Densities in a Human Head Phantom," *2019 IEEE International Ultrasonics Symposium (IUS)*, Glasgow, UK, 2019, pp. 2049-2051, DOI: 10.1109/ULTSYM.2019.8926286.

[6] Witte, R. S., Allard, M., Trujillo, T., Alvarez, A., Preston, C., Kang, J., O'Donnell, M., "Transcranial acoustoelectric imaging: Towards noninvasive mapping of current densities in the human brain.", March 2023, *J. Acoust. Soc. Am.* 1; 153 (3_supplement): A154. https://doi.org/10.1121/10.0018479.

[7] Olafsson, R. , Witte, R. S., Huang, S. -W., and O'Donnell, M., "Ultrasound Current Source Density Imaging," July 2008 in *IEEE Transactions on Biomedical Engineering*, vol. 55, no. 7, pp. 1840-1848, DOI: 10.1109/TBME.2008.919115.

[8] Holdefer, R. N., Sadleir, R., & Russell, M. J., "Predicted current densities in the brain during transcranial electrical stimulation", 2006, *Clinical neurophysiology : official journal of the International Federation of Clinical Neurophysiology*, *117*(6), 1388–1397. https://doi.org/10.1016/j.clinph.2006.02.020.

[9] Witte, R. S., Olafsson, R., O'Donnell, M., "1A-4 Acoustoelectric Detection of Current Flow in a Neural Recording Chamber", 2006 *IEEE Ultrasonics Symposium*, Vancouver, BC, Canada, DOI: 10.1109/ULTSYM.2006.16

[10] Preston, C., Kasoff, W.S., Witte, R. S., "Selective Mapping of Deep Brain Stimulation Lead Currents Using Acoustoelectric Imaging" 2018 *Ultrasound in Medicine & Biology*, DOI: 10.1016

[11] Preston, C., Alvarez, A., Barragan, A., Becker, J., Kasoff, W. S., Witte, R. S., "High resolution transcranial acoustoelectric imaging of current densities from a

directional deep brain stimulator", 2020 *Journal of Neural Engineering,* DOI

10.1088/1741-2552/ab6fc3

[12] Qin, Y., Li, Q., Ingram, P., Barber, C., Liu, Z., Witte, R. S., "Ultrasound

Current Source Density Imaging of the Cardiac Activation Wave Using a Clinical

Cardiac Catheter", 2015 *IEEE Transactions on Biomedical Engineering,* vol. 62, no. 1,

pp. 241-247, DOI: 10.1109/TBME.2014.2345771.

[13] Alvarez, A., Preston, C., Trujillo, T., Wilhite, C., Burton, A., Vohnout, S.,

Witte, R. S., "*In vivo* acoustoelectric imaging for high-resolution visualization of

cardiac electric spatiotemporal dynamics", 2020 *Appl. Opt.* 59, 11292-11300

[14] Chen, L., Lu, X., Zhang, J., Chu, X., & Chen, C., "HINet: Half Instance

Normalization Network for Image Restoration", 2021 *CVPRW2021,*

arXiv:2105.06086 [eess.IV].

[15] Kingma, D. P., & Ba, J., "Adam: A Method for Stochastic Optimization.",

2014, *CoRR,* arXiv:1412.6980 [cs.LG].

[16] Cawley, G.C., & Talbot, N.L., "On Over-fitting in Model Selection and

Subsequent Selection Bias in Performance Evaluation", 2010 *J. Mach. Learn. Res.,*

11, 2079-2107.

[17] Ronneberger, O., Fischer, P., Brox, T., "U-Net: Convolutional Networks for

Biomedical Image Segmentation", 2015 In: Navab, N., Hornegger, J., Wells, W.,

Frangi, A., (eds) *Medical Image Computing and Computer-Assisted Intervention –*

MICCAI 2015. MICCAI 2015. Lecture Notes in Computer Science(), vol 9351.

Springer, Cham, DOI:10.1007/978-3-319-24574-4_28.