Project Report : CS 7643

Dan Brickner
dbrickner3@gatech.edu

David Buchanan
williamdbuchanan@gatech.edu

Jason Reed
jreed71@gatech.edu

Ryan Angelo
rangelo6@gatech.edu

Abstract

fastMRI is a project supported by Facebook AI Research (FAIR) and NYU Langone Health to increase the speed of MRI scans through the use of Artificial Intelligent and Deep Learning. The curated data-set provided for training these algorithms is a collection of MRI data provided by NYU Langone, a first of its kind. This investigation aims to analyze three central elements of the fastMRI project: investigating possible novel metrics to analyze performance, computational resource consumption enhancements, and analysis of the use of texture or shape within the training of the model. We propose the use of EfficientNets and Squeeze U-Nets to alleviate use of computational resources. We also provided an analysis of various different loss metrics. Finally, we implemented stylized images to help the network focus on the shape of the image. Although our computational resources were quicker, the accuracy was not worth the speed gains. Lastly, we did see some minor improvements using style transfer and compound loss functions.

1. Introduction/Background/Motivation

One of the major issues with MRI (Magentic Resonance Imaging) scans is that they can take a long time. This can cause extreme discomfort for the patient and raises the costs of the procedure [11]. There are methods that can be used today to reduce processing time, but they result in lower quality images. The data-set provided by Facebook called fastMRI can be used in conjunction with Deep Learning to ensure quality images can be used in conjunction with faster MRI processing methods [11]. The goal of this paper is to present several modifications to enhance the learning process. Specifically, we aim to reduce processing and resource consumption with minimal effect to the resulting image quality. We also aim to increase the accuracy of the fastMRI project.

Today, most CNNs (Convolutional Neural Networks) use deeper and deeper networks as the network of choice for their training. Although these networks have been enhanced to produce better performance, they are becoming more computationally expensive as we deepen our networks. This causes concerns both from a cost perspective and from an environmental perspective [3]. CNNs also can have issues when training where they can focus too much on the texture of an image rather than its shape. Research has shown that it can be easy to fool a network by perturbing the texture of an image [15]. Finally, many CNNs take advantage of common loss functions like MSE (Mean-Squared Error) and SSIM (Structural Similiarity). Although these loss functions have been successful, it has been suggested that there may be better loss functions available for use by the fastMRI project [11].

By offering enhancements to these problems, there is a large potential for monetary savings, time savings, better model performance, and ultimately better patient care. It may also allow other researchers working on the fastMRI project to use some of these ideas in their development, thus helping to propel the field forward.

For this project, our team used the fastMRI data-set, which was created by Facebook AI Research (FAIR) to democratize neural networks for MRI reconstructions. Within the data-set, there are a number of different MRI scans and imaging types to select from. Our team opted to select the knee single-coil MRI data. When we say single-coil, this refers to the type of MRI that scanned the knee. Single-coil machines have one coil that produce a single complete image of whatever is being scanned. (They have one channel) This is in comparison to multi-coil machines, which have many different coils that combine to produce an image [11] (They have many channels). An example of a single-coil image is provided in Figure 1. Note that the images are not originally provided in this format, but require an inverse Fourier Transform and absolute value transformation to visualize properly. The images are not labelled but have a ground truth image (which is full sized) since the reconstructed images are compared against the original images. The data includes various knee images from different MRI scanners totaling up to about 115 GB of data. To reduce

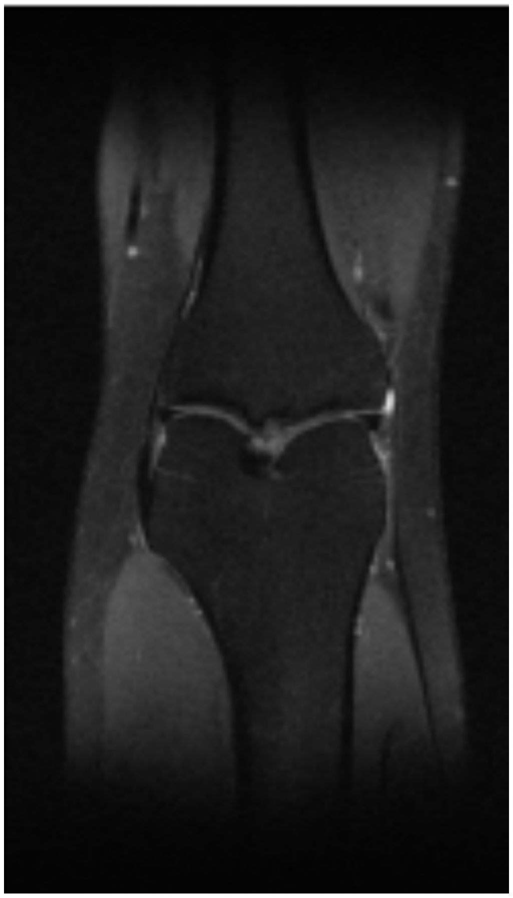

Figure 1. A single-coil knee data sampled after being transformed

time and resource usage, we opted to work with a reduced data-set that is 28 percent of the original. Note this data-set is fully self-contained and keeps the patient's knee data fully confidential.

2. Approach

The following section is split up into three areas of focus: Computational Efficiency Enhancements, Alternative Loss Functions, and Shape Based Recognition.

2.1. Computational Efficiency Enhancements

Some of the best networks that have been created recently are also extremely inefficient in terms of storage space and computational requirements. In working to explore computational efficiency with the dataset, we implemented two new architectures, EfficientNet and Squeeze U-Net.

One of the most powerful networks is called EfficientNet, which seeks to mitigate the problem of computational complexity. EfficientNet is able to achieve over 85 percent accuracy on ImageNet with far less parameters than most deep networks that exist today [16]. Instead of using the UNet that comes by default with the fastMRI project, we replaced it with an EfficientNet model that also includes a decoding phase.

The reason why this could be successful is because of three major reasons. First, EfficientNet is as effective as many models that are close to state of the art. In fact, EfficientNetV2 as of this date, is the state of the art for performance on ImageNet [17]. The second reason this should work is because there are far less parameters necessary than with the standard UNet network. The default UNet uses about 8 million parameters while the B-0 version of EfficientNet uses about 4 million parameters. The final and most critical reason is that even the largest EfficientNets are 8.4x smaller and 6.1x faster than most of today's best networks [16].

Getting the EfficientNet to work took many tries. First, the output of a UNet is an image of the same size, first encoded and then decoded. Because EfficientNets were implemented for classification, the final output is supposed to be one of n classes. This was the first major problem encountered and was fully expected. To make up for this, we needed to remove the final fully connected layer and the last max-pooling layer, instead implementing a decoder to output the reconstructed image. There are existing implementations of this style of EfficientNet, most notably one called Eff-UNet [5]. We adopted this model for our implementation.

The part that gave us the most trouble was implementing this model into the MriModule, a type of PyTorch Lightning module. This is essentially a wrapper class around the network that implements some high level functions and hyper parameters. Given that this was our first exposure to PyTorch Lightning, this caused some confusion around implementation.

We used the following repositories for our implementation. For the main setup, we used the fastMRI repository. This was the basis of all modifications we made [4]. We also used the EfficientNet GitHub repo [12], which is further enhanced by the EffuNet repository on GitHub [13]. For the last repository, we made changes to the EffUNet model to support single channel images because as a default, it only supports 3 channels.

A second network that we implemented was Squeeze U-Net, introduced by Beheshti and Johnsson in 2020. [6] Squeeze U-Net was created with embedded devices in mind that do not have significant computational resources. By reducing the model size and the operations that are required, Squeeze U-Net has been shown to achieve 17% faster performance than U-Net while maintaining the same accuracy [6]. These improvements allow for low energy devices to operate effectively on smaller models. The design of Squeeze U-Net is based on two other architectures,

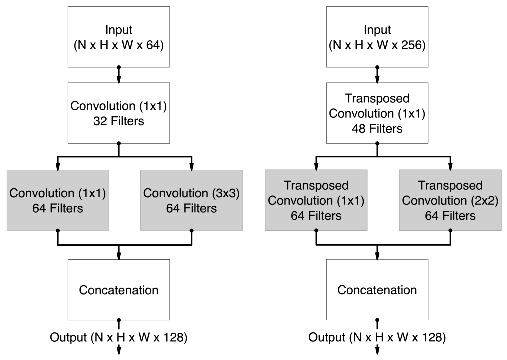

Figure 2. Left: Squeeze U-Net Fire Module. Right: Transposed Convolution in the expansive path [6]

SqueezeNet [10] and U-Net [5].

Squeeze U-Net uses Fire modules which perform convolutions in parallel, and then combines these convolutions as the output of the fire module. The Fire module replaces the up sampling and down sampling layers that are in UNet, and reduces the parameters to 2.5 million, down from 7 million with the traditional UNet. The Fire Modules were introduced in SqueezeNet in 2016[10] and are described in 2. Based on this reduction in the computational requirements of the model, we felt that this would be a good architecture to implement to explore opportunities for computational performance enhancements. Additionally, this is a new algorithm which makes it a good option for analysis using the fastMRI dataset.

For Squeeze U-Nets, we started by analyzing the differences between U-Nets and the Squeeze U-Nets which focused on the reduction of computation by using the fire modules for convolutions. The paper notes that further investigation into the accuracy of Squeeze U-Net with different types of classes is required, so we were not sure how the algorithm would perform on the fastMRI data.

The most challenging aspect of implementing Squeeze U-Net was understanding how to properly manage the layer size between each operation. The original paper is relatively sparse on implementation details because it is structurally based on SqueezeNet and U-Net, so additional work had to be done to understand SqueezeNet and UNet, while also comparing implementations of SqueezeNet and UNet against the descriptions provided in the paper.

Our implementation of Squeeze-UNet was derived from the only identifiable implementation of this model, which uses Keras[1].

Additionally, support for implementing the Fire module was derived from the Pytorch implementation of a Fire Module[2].

To date, we have not seen an implementation of EfficientNet B-0 or Squeeze U-Net used for the fastMRI project.

2.2. Shape Based Recognition

Previous research has been done examining shape versus texture bias in CNNs [9]. A majority of this research was done on the ImageNet data set as this contains annotations, variety of classes, and numerous training examples. While findings varied for whether CNNs prefer shape or texture bias it was found that the ImageNet data set was biased heavily towards texture. For this project our goal was to examine if the MRI sliced images had either a texture or shape bias. Depending on which bias the images had, we were interested in tuning our model towards the lesser bias so the model generalizes better. A key component of figuring out the texture versus shape bias in [9] was through human experiments compared against neural networks in [9].

As we did not have time to re-do these experiments with the fastMRI data set we decided to utilize performance increases in MSE and SSIM as an indicator of better or worse generalization of the data and thus a reduction in bias for texture and/or shape. There was a concern that adding texture to these images would negatively impact performance for reconstruction as the new images would not resemble the target goal. In section III we discuss the issues in more depth.

Style transfer in the original paper was used more for shape/texture analysis but in our case we are not predicting a class, we are re-constructing images and minimizing the loss between the reconstruction MRI image and the traditionally constructed MRI image. After more analysis, the main application of the style transfer is to add types of noises to the image so the model is better able to handle incorrect scans. While the approach of augmenting data sets to account for unlikely scenarios has been used in other papers, we believe using style transfer for the fastMRI is a unique way of generating noise that will help the model generalize better.

2.3. Alternative Loss Functions

The objective of MRI reconstruction is to create radiologically interpretable images, and there is a sentiment among MRI researchers that standard image similarity metrics do not fully capture interpretability. Currently, L1 loss, MSE and SSIM are commonly used as loss functions in MRI reconstruction [11]. Zhao et al proposed a weighted combination of L1 loss and MS-SSIM (Multi-Scale Structural Similarity) [18]. Weighted combinations of a simple loss like L1 or MSE and a structural loss like SSIM or MS-SSIM have been used in MRI reconstruction models [14].

SSIM measures the structural similarity of two image patches. The human visual system is sensitive to structural differences in small image patches. Experiments have shown that SSIM has advantages in measuring image similarity to humans over simple losses like L1 or MSE. MS-SSIM is a combination of SSIM measures at different scales, to capture structure at multiple levels of locality [18].

We compared the effects of four loss functions, L1 loss, MSE, SSIM and MS-SSIM, on the fastMRI UNet baseline model. We also tested weighted combinations of one simple loss, either L1 or MSE, with one structural loss, either SSIM or MS-SSIM, with different weights. We took metrics and compared reconstruction images and visualizations to determine if the different loss functions would result in a quantitative or qualitative difference in results. In general image reconstruction, MSE loss has been found to produce small local textural artifacts, since the loss is not as sensitive to small local differences as the human visual system is. SSIM and MS-SSIM capture structure better than simple losses, but are not very sensitive to global shifts, and have been found to result in changes to brightness or overall color. The simple loss component of the compound losses has been observed to prevent these global shifts [18]. We examined whether these losses had similar effects in MRI reconstruction, and whether compound losses might result in higher quality images.

We modified the fastMRI UNet training code to use various loss functions. We implemented compound loss functions and wrote visualization scripts to examine the reconstructions. The structural losses are both sensitive to hyper parameters involving the range of the data, so we had to tune these and rescale the data to allow the network to train effectively with these losses. One challenge was the computational resources required to perform the significant number of runs needed for comparison. With the full dataset, each run took multiple days, so a reduced dataset or 26 MRI scans was used for the loss function trials.

We used the fastMRI repository for the base UNet model and original training scripts. We also used an implementation of MS-SSIM loss for PyTorch[7].

3. Experiments and Results

The following section is split up into three areas of focus: Computational Efficiency Enhancements, Alternative Loss Functions, and Shape Based Recognition.

3.1. Computational Efficiency Enhancements

We defined success for this part of the experiment through accuracy and average epoch elapsed time. More specifically, to measure the accuracy we used the MSE and SSIM. To measure elapsed time, we recorded the average time each epoch took. We performed experiments using the UNet as the standard to beat, several variations of EfficientNets, and our Squeeze U-Net implementation to test with. We tested 3 configurations of the EfficientNets, versions B-0, B-3, and B-4, each one subsequently having more parameters and thus taking more time.

The tests were performed entirely on the exact same set of training and validation data. The training and validation split was 75/25 respectively. The validation set was further split up again into a true validation set and a test set 70/30 respectively. Finally, to reduce the amount of time to train the model, we used a reduced data-set which was 28 percent of the original data-set provided by FAIR.

The tests were performed sequentially on the GCP VM. We started with the UNet test, training the UNet. At the end of the training cycle, we recorded all of the epoch time elapsed measurements and averaged them together. We then ran the test images through the trained network to produce reconstructions. These reconstructions were then tested against the ground truth images using both MSE and SSIM. The final accuracy results were recorded through the final MSE and SSIM given. The results in terms of MSE, SSIM, and average epoch time can be seen in Table 1 below. On the whole, the experiment could be seen as both a failure and a success.

The speed of an EfficientNet is quite obvious. The best EfficientNet takes only about 61 percent of the time it takes to train a UNet. This should add up to considerable cost and time savings. Even the B4 EfficientNet, which has over twice as many parameters as a UNet, still manages to only take about 84 percent of the time UNets take.

On the other hand, there is a noticeable difference in the accuracy between UNets and EfficientNets. For EfficientNet B0, the UNet is over 4 times better in terms of MSE. Even the B4 EfficientNet is still about twice as bad as the UNet in terms of MSE. The difference can be seen in the images in Figure 3.

For Squeeze U-Net we saw lackluster performance, and even though there were only 2.5 million parameters, the number of average seconds per epoch was the same as EfficientNet-B3, while only providing an SSIM value of 0.4928. Two potential reasons for this poor performance include hyper parameters that are causing poor performance in the model, or an implementation issue, as visualized in Figure 4. Although the Squeeze U-Net boasts much smaller model sizes compared to UNet, based on the performance that we were able to achieve, it didn't work effectively on the MRI dataset.

3.2. Shape Based Recognition

First, to reduce the shape bias we needed to augment our data set by performing style transfer on fastMRI training data images using the code provided by the paper [8] and minor tweaks. This style transfer obfuscates the original texture of the images and forces the network to learn other aspects of the images (ideally the shape). Unlike ImageNet however, the MRI images and slices they provide do not have a wide variety of textures. During our experiments it was shown that some style transfers did negatively im-

Network	Number of Parameters	Average Epoch Training	MSE	SSIM
UNet	7 Million	710 seconds	2.121e-10	.9064
EfficientNet-B0	4 Million	440 seconds	9.197e-10	.8237
EfficientNet-B3	11 Million	540 seconds	5.761e-10	.8424
EfficientNet-B4	18 Million	600 seconds	4.608e-10	.8287
Squeeze-UNet	2.5 Million	540 seconds	1.827e-0	.4928

Table 1. UNet vs. EfficientNet vs. Squeeze-UNet

Loss	Alpha	MSE	SSIM
L1		1.04e-10	0.7401
MSE		1.071e-10	0.738
SSIM		1.131e-10	0.7432
MS-SSIM		1.362e-10	0.7336
L1, SSIM	0.1	1.105e-10	0.7444
MSE, SSIM	0.1	1.105e-10	0.7443
L1, MS-SSIM	0.1	1.092e-10	0.7441
MSE, MS-SSIM	0.1	1.113e-10	0.7421
L1, SSIM	0.5	1.07e-10	0.7457
MSE, SSIM	0.5	1.083e-10	0.744
L1, MS-SSIM	0.5	1.069e-10	0.7417
MSE, MS-SSIM	0.5	1.08e-10	0.7404
L1, SSIM	0.9	1.055e-10	0.7413
MSE, SSIM	0.9	1.09e-10	0.7398
L1, MS-SSIM	0.9	1.055e-10	0.7406
MSE, MS-SSIM	0.9	1.065e-10	0.7387

Table 2. Loss Function Comparison

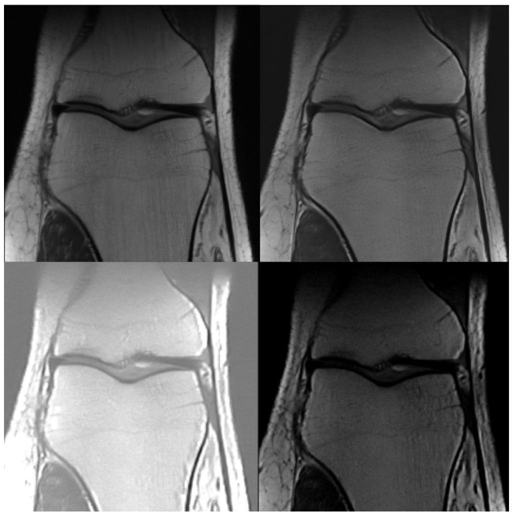

Figure 3. Top-Left: The original knee image. Top-Right: The reconstruction from the default UNet. Although it is slightly lower in quality, the reconstruction contains much of the same information. Bottom-Left: The reconstruction from EfficientNet-B0. The image still has good detail, but is washed out and does not capture as much. Bottom-Right: The reconstruction from EfficientNet-B4. An improvement over B0, but is slightly underexposed and is still less clear than the UNet.

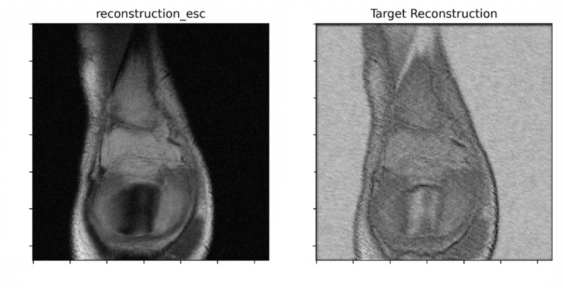

Figure 4. Left: Ground Truth image. Right: Reconstruction (Output) image. Note the inverse shading, indicating a potential algorithmic or data handling issue.

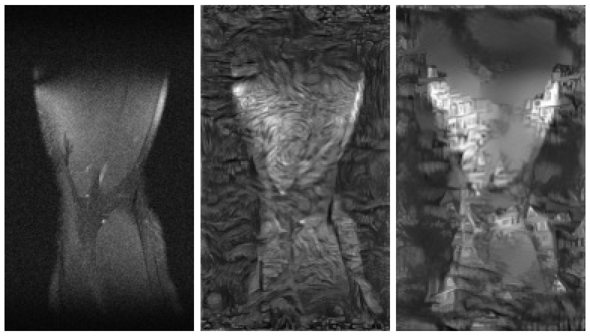

Figure 5. Single Coil Sliced Images. Left: Original, Middle: Van Gogh Style, Right: Tuebingen Style

pact performance while other styles slightly improved performance. This is likely due to the fact that some textures that were added resemble typical image noise (salt/pepper, banding, etc...) and provides a good learning example to the model while other textures provide a completely foreign shape and thus, bad performance. This can be seen in Figure 5 where the Van Gogh styling keeps the overall shape of the original image but the Tuebingen styling almost completely obfuscates the muscle tissue.

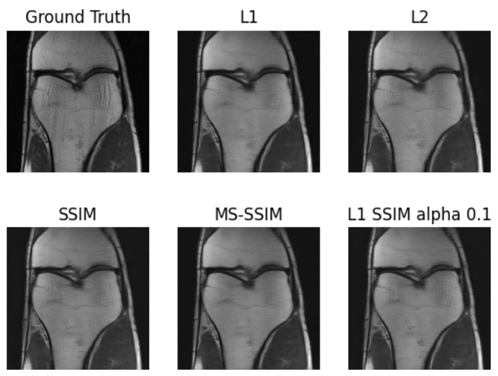

Figure 6. Comparison of reconstructions produced with different loss functions.

This was reflected in the performance of the UNet model as the additional data with the Van Gogh styling performed better than the one with Tuebingen styling (see Table 3). When training the models 6% of the data was selected for augmentation and the other 92% was kept the same as the original model. The decision to use less augmented data was chosen because the performance of the model lowers as more of the data has these ”negative” training examples. This aligns with our hypothesis earlier that the style transfers do not always improve performance.

3.3. Loss Function Comparison

We trained UNet on a dataset of 26 MRI scans with L1, MSE, SSIM and MS-SSIM loss functions. We also trained with compound losses combining a simple loss function (either L1 or MSE) with a structural loss function (either SSIM or MS-SSIM). The losses were combined with the linear combination:

$$\alpha * simple + (1 - \alpha) * structured$$

We tested α values 0.1, 0.5 and 0.9. We were looking for differences in similarity metrics or qualitative results between the networks trained with the different loss functions.

We found that the choice of loss function did not have a large effect on the resulting reconstruction similarity metrics. Compound losses with balanced weights or weights that were skewed towards the structural loss consistently performed slightly better in terms of SSIM than all the single losses or the compound losses that were weighted more towards the simple loss. The choice of simple loss and structural loss did not appear to have a significant effect on the resulting metrics. The networks trained purely with structural losses seemed to perform slightly worse on the MSE metric, which is not unexpected as the other loss functions contain elements of MSE or the related L1 loss. However, the differences in all these metrics were surprisingly small, perhaps because all the loss functions are sufficient to train the given network architecture to create reconstructions that are, at least visually, close to ground truth.

Examining the images qualitatively, all the networks trained with the different loss functions produced solid reconstructions. Differences from ground truth were mostly in blurring or smoothing of fine detail. This appeared to be common across all the trained networks. Differences between the ground truth and the different reconstructions were minute enough that it was difficult to identify a distinct qualitative pattern. Examining the MSE between ground truth and reconstructions as well as between different reconstructions showed that the reconstructions tended to be more similar to each other than they were to the ground truth, often by an order of magnitude. This suggests that all the loss functions result in similar reconstructions, with the main difference from ground truth being some blurriness of fine details. During training, we did find that early on networks with structural losses tended to capture structure before shade, as described in [18], but by the time the network was fully trained the shade matched the ground truth closely. We conclude that all these loss functions can train networks of similar capability with the available data.

4. Reflections

The structure of the problem is a problem about properly upsampling an image ultimately. This suggests we want to compare a ground truth upscaled image to a reconstructed image. In effect, this informs us about the type of model we need to select: a self-supervised encoder-decoder network.

For EfficientNets, there is only one layer that does not have learned parameters. This is a Max Pooling layer that lies between the last convolution and the last fully connected layer. Other than that, EfficientNet's heavily rely around Convolutional Layers and a single fully connected layer.

Likewise, Squeeze U-Net's model has the simultaneous convolution layers inside of a Fire Module that are concatenated. The fire modules ”squeeze” the output channels and use the convolutions in order to acquire features that were not collected in the previous layers, and then concatenate them. This is in contrast to the traditional UNet contracting path that uses a convolution followed by a Max-Pooling layer, which generates significantly more parameters.

For EfficientNets and Squeeze U-Nets, the input of the network expects an image of size nxrxc (channels, rows, columns). For the EfficientNets output, they typically produce softmax classifier predictions. In the case of fastMRI, we needed the output to be a reconstructed image matching the input image size. This is why we augmented it with a decoder function using an Eff-UNet. For data preprocessing, the input data is in k-space. This k-space data requires an Inverse Fourier Transform operation and a Complex Absolute Value function to be applied. This ensures the image is

Style	MSE	NMSE	PSNR	SSIM
Van Gogh	1.517e-10 +/- 4.599e-11	0.01089 +/- 0.003704	33.67 +/- 0.9137	0.9093 +/- 0.02151
Tuebingen	3.259e-10 +/- 2.251e-11	0.02384 +/- 0.01357	30.32 +/- 1.935	0.8931 +/- 0.002783
Tuebingen and Van Gogh	2.808e-10 +/- 2.002e-11	0.02076 +/- 0.01457	30.97 +/- 2.545	0.9015 +/- 0.0358

Table 3. Augmented Data Model Performance

in the grayscale format we expect. For data post-processing, the output is already in grayscale and is ready to compare with the transformed input. The loss function used was an L1 Loss between the output and the transformed grayscale input.

For EfficientNets, the model overfit after about 20 epochs. For Squeeze U-Nets, we saw similar characteristics where the model overfit after about 20 epochs. Although PyTorch is intelligent enough to save only the best checkpoint, we found the models produced after epoch 20 tended to be overfit.

For EfficientNets, there were a number of hyperparameters including the learning rate, dropout probability, and incremental learning rate decay. We chose the hyperparameters first based on default values, then by trail and error. The EfficientNets were highly sensitive to dropout rates, thus I used a low value of 5%. For the learning rate, I found empirically that a learning rate of .001 worked best.

For Squeeze U-Nets, the hyperparameters include the deconvolution kernel size, the dropout probability, learning rate, gamma, step size, batch size and the squeeze planes used in the Fire module for convolution. For the deconvolution kernel size we used the same values discussed in the paper (3x3, 2x2 and 1x1). For batch size, we trained using the same value we used for the UNet base model, using a batch size equal to our GPU count (1).

Both EfficientNet and Squeeze U-Net leveraged the RMSprop optimization algorithm.

We used PyTorch for the implementation of our networks.

All of us started with the GitHub repository that Facebook provided for fastMRI. This provided the default UNet model and a pretrained example we could test with. It also provided a library of common functions for data loading, transforms, training, and evaluation. For the specific enhancements, the repositories and changes we made were mentioned in the Approach section.

Although we were unable to get the EfficientNet or Squeeze U-Net to outperform the base UNets, it may be that further optimizations could allow for better performance. Recently EffcientNetV2 have been released, and they contain the state-of-the-art for performance while also being much lighter. This could provide grounds for both better speed and improved accuracy.

Our code is available at the following location for reference: https://gatech.box.com/s/tvt42ot92ejnd5ws1jqzggzzsxfpu8y3

5. Work Division

Table 4 shows how the work was divided up by our group.

Student Name	Contributed Aspects	Details
Dan Brickner	EfficientNet Infrastructure First Spin Up of Default Model	Implemented EfficientNet into fastMRI, evaluating performance and accuracy for the modified network. Worked with the group to create the GCP VM. Trained and evaluated the initial default UNet model provided by Facebook.
David Buchanan	Pytorch Style Transfer Model Tuning Shape and Texture Bias Analysis Infrastructure	Implemented a pipeline to ingest the fastMRI single coil knee images and apply style transformation on the images and then re-construct them back into a format that fastMRI can process. Provided a discussion on the texture and bias analysis with respect to the fastMRI images and the applicability of style transfer for this use case.
Jason Reed	Loss Function Comparison Infrastructure	Implemented training UNet model with different loss functions and implemented compound loss functions. Tuned hyperparameters and data scaling to make training effective with structural loss functions. Visualized, analyzed and discussed results. Helped set up several Azure VMs for training.
Ryan Angelo	Dataset Management Squeeze U-Net Metrics Visualizations	Broke apart Data set for different training scenarios. Created visualization tool for comparing output single-coil data with the ground truth single-coil data. Implemented Squeeze U-Net, comparing against the base UNet implementation to explore opportunities for reductions in computational complexity. Helped support the Infrastructure.

Table 4. Contributions of team members.

References

[1] lhelontra/squeeze-unet. 2021-05-02. https://github.com/lhelontra/squeeze-unet. 3

[2] pytorch/vision. 2021-05-02. https://github.com/pytorch/vision. 3

[3] Shrinking deep learning's carbon footprint. 2021-05-02. https://news.mit.edu/2020/shrinking-deep-learning-carbon-footprint-0807. 1

[4] facebookresearch/fastMRI, May 2021. original-date: 2018-10-26T22:21:23Z. 2

[5] Bhakti Baheti, Shubham Innani, Suhas Gajre, and Sanjay Talbar. Eff-UNet: A Novel Architecture for Semantic Segmentation in Unstructured Environment. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 1473–1481, Seattle, WA, USA, June 2020. IEEE. 2, 3

[6] Nazanin Beheshti and Lennart Johnsson. Squeeze U-Net: A Memory and Energy Efficient Image Segmentation Network. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 1495–1504, Seattle, WA, USA, June 2020. IEEE. 2, 3

[7] Gongfan Fang. VainF/pytorch-msssim. 2021-05-02. https://github.com/VainF/pytorch-msssim. 4

[8] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. Image Style Transfer Using Convolutional Neural Networks. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2414–2423, Las Vegas, NV, USA, June 2016. IEEE. 4

[9] Robert Geirhos, Patricia Rubisch, Claudio Michaelis, Matthias Bethge, Felix A. Wichmann, and Wieland Brendel. ImageNet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness. *arXiv:1811.12231 [cs, q-bio, stat]*, Jan. 2019. arXiv: 1811.12231. 3

[10] Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, and Kurt Keutzer. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size. *arXiv:1602.07360 [cs]*, Nov. 2016. arXiv: 1602.07360. 3

[11] Florian Knoll, Jure Zbontar, Anuroop Sriram, Matthew J. Muckley, Mary Bruno, Aaron Defazio, Marc Parente, Krzysztof J. Geras, Joe Katsnelson, Hersh Chandarana,

Zizhao Zhang, Michal Drozdzalv, Adriana Romero, Michael Rabbat, Pascal Vincent, James Pinkerton, Duo Wang, Nafissa Yakubova, Erich Owens, C. Lawrence Zitnick, Michael P. Recht, Daniel K. Sodickson, and Yvonne W. Lui. fastMRI: A Publicly Available Raw k-Space and DICOM Dataset of Knee Images for Accelerated MR Image Reconstruction Using Machine Learning. *Radiology: Artificial Intelligence*, 2(1):e190007, Jan. 2020. 1, 3

[12] Luke Melas-Kyriazi. lukemelas/EfficientNet-PyTorch, May 2021. original-date: 2019-05-30T05:24:11Z. 2

[13] Pranshu Mishra. pranshu97/effunet, Apr. 2021. original-date: 2020-12-17T11:29:09Z. 2

[14] Matthew J. Muckley, Bruno Riemenschneider, Alireza Radmanesh, Sunwoo Kim, Geunu Jeong, Jingyu Ko, Yohan Jun, Hyungseob Shin, Dosik Hwang, Mahmoud Mostapha, Simon Arberet, Dominik Nickel, Zaccharie Ramzi, Philippe Ciuciu, Jean-Luc Starck, Jonas Teuwen, Dimitrios Karkalousos, Chaoping Zhang, Anuroop Sriram, Zhengnan Huang, Nafissa Yakubova, Yvonne Lui, and Florian Knoll. State-of-the-Art Machine Learning MRI Reconstruction in 2020: Results of the Second fastMRI Challenge. *arXiv:2012.06318 [cs, eess]*, Dec. 2020. arXiv: 2012.06318. 3

[15] Akshayvarun Subramanya, Vipin Pillai, and Hamed Pirsiavash. Fooling Network Interpretation in Image Classification. *arXiv:1812.02843 [cs]*, Sept. 2019. arXiv: 1812.02843. 1

[16] Mingxing Tan and Quoc V. Le. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. *arXiv:1905.11946 [cs, stat]*, Sept. 2020. arXiv: 1905.11946. 2

[17] Mingxing Tan and Quoc V. Le. EfficientNetV2: Smaller Models and Faster Training. *arXiv:2104.00298 [cs]*, Apr. 2021. arXiv: 2104.00298. 2

[18] Hang Zhao, Orazio Gallo, Iuri Frosio, and Jan Kautz. Loss Functions for Image Restoration With Neural Networks. *IEEE Transactions on Computational Imaging*, 3(1):47–57, Mar. 2017. 3, 4, 6