Partial Scan Electron Microscopy with Deep Learning

Jeffrey M. Ede and Richard Beanland

{j.m.ede, r.beanland}@warwick.ac.uk

Abstract: We present a multi-scale conditional generative adversarial network that completes 512×512 electron micrographs from partial scans. This allows electron beam exposure and scan time to be reduced by 20× with a 2.6% intensity error. Our network is trained end-to-end on partial scans created from a new dataset of 16227 scanning transmission electron micrographs. High performance is achieved with adaptive learning rate clipping of outlier losses and an auxiliary trainer network. Source code and links to our new dataset and trained network have been made publicly available at https://github.com/Jeffrey-Ede/partial-STEM.

Keywords: electron microscopy, generative adversarial network, high resolution, partial scan

1 Introduction

Scanning transmission electron microscopy (STEM) can achieve 1-2 pm precision[1] and is able to resolve atom columns. Nonetheless, beam damage[2, 3] limits materials that can be studied to inorganic crystals and select organic structures. High-resolution STEM scans also take time, taxing experimenters and allowing microscope settings to drift. In response, we have developed a generative adversarial network[4] (GAN) to reduce electron beam exposure by completing realistic electron micrographs from partial scans. Examples are shown in fig. 1.

Conditional GANs[5] consist of sets of generators and discriminators that play an adversarial game. Generators learn to produce outputs that look realistic to discriminators.

Figure 1: Adversarial and non-adversarial neural network completions of test set 512×512 1/20 coverage blurred spiral partial scans. Adversarial completions have realistic noise characteristics and colouration whereas non-adversarial completions are blurry. Enlarged 64×64 regions from the top left of each image are inset to ease comparison.
Meanwhile, discriminators learn to distinguish between real and generated examples. Limitedly, discriminators only assess whether outputs look realistic; not if they are correct. This can lead to mode collapse\cite{6}: where generators only produce a subset of outputs. To lift the degeneracy, generator learning is conditioned on a distance between generated and correct outputs that is added to the adversarial loss. Meaningful distances can be learned automatically by considering differences between features imagined by discriminators for real and generated images\cite{7, 8}.

Deep learning has a history of successful applications to image infilling, including image completion\cite{9}, irregular gap infilling\cite{10} and supersampling\cite{11}. This motivates the application of deep learning to the completion of partial scans. Examples of spiral and grid-like partial scans with 1/10, 1/20, 1/40 and 1/100 coverage are in fig. 2. Most infilling networks use non-adversarial mean squared errors (MSEs) for training. However, this results in blurry and unnatural infilling for large gaps. Non-neural methods have the same issues and higher errors e.g. \cite{12}. In contrast, a conditional GAN can ensure realistic completions for partial scans with arbitrary coverage.

This paper presents a multi-scale conditional GAN that completes 512 × 512 STEM images from partial scans. Our network configuration, new 16227 STEM image training dataset and learning policy are described in section 2. Performance and example applications are presented in section 3. Architecture and learning policy experiments are detailed in section 4. Finally, adaptive learning rate clipping to stabilize low batch size training is presented in section 7, followed by detailed network architecture in section 8.

2 Training

In this section, we discuss training with the TensorFlow\cite{13} deep learning framework. Training was performed using ADAM\cite{14} optimized stochastic gradient decent and takes over a week on an Nvidia GTX 1080 Ti GPU with an i7-6700 CPU.

2.1 Data Pipeline

A new dataset of 16227 32-bit floating point STEM images saved to University of Warwick data servers was collated for training. The dataset consists of individual micrographs made by dozens of scientists working on hundreds of projects and therefore has a diverse constitution. The dataset is available by request\footnote{Contact: \{j.m.ede, r.beanland\}@warwick.ac.uk} and will be published as part of the 226862 labelled micrograph Warwick Large Electron Microscopy Dataset (WLEMD) in a following publication.

The dataset was split into 12170 training, 1622 validation and 2435 test micrographs. Micrographs were

![Figure 2: 512 × 512 1/10, 1/20, 1/40 and 1/100 coverage partial scans. Spiral scans are blurred; grid-like scans have small random perturbations.](image)
Partial scan paths, \( I_{\text{path}} \), were drawn by adjusting the signal-to-noise ratios of traversed pixels. Partial scan signal-to-noise was reduced where pixels were traversed partially or quickly. Full details are in our source code. Partial scans, \( I_{\text{scan}} \), were simulated by combining scan paths with low-pass filtered micrographs

\[
I_{\text{scan}} = I_{\text{path}}^{2} \text{low-pass} + N(I_{\text{path}}, I_{\text{blur}}),
\]

where \( N \) is a function that simulates STEM noise[15]. For simplicity, we chose

\[
N(I_{\text{path}}, I_{\text{blur}}) = (1 - I_{\text{path}}) I_{\text{path}} \text{low-pass} U,
\]

where \( U \) is a uniform random variate distributed in \([0, 2)\).

### 2.2 Network Configuration

To generate realistic images, we employ a multi-scale conditional GAN. This can be partitioned into the six subnetworks shown in fig. 3: an inner generator, \( G_{\text{inner}} \), outer generator, \( G_{\text{outer}} \), inner generator trainer, \( T \), and small, medium and large scale discriminators, \( D_1, D_2 \) and \( D_3 \). We refer to the compound network \( G(I_{\text{scan}}) = G_{\text{outer}}(G_{\text{inner}}(I_{\text{scan}})), I_{\text{scan}} \) as the generator. The generator is the only network needed for inference. Multi-scale discriminators refer to the collection \( D = \{D_1, D_2, D_3\} \). Detailed architecture is in section 8.

**Discriminators:** Multi-scale discriminators examine real and generated STEM images to predict whether they are real or generated. Essentially, discriminators adapt to the generator as it learns. Each discriminator assesses a different-sized crop with size \( 280 \times 280 \), \( 140 \times 140 \) or \( 70 \times 70 \) from \( 512 \times 512 \) images. Typically, discriminators are applied to fractions of the full image size e.g. \( 1/2^0 \), \( 1/2^1 \) and \( 1/2^2 \) times the output side length in [7]. However, we found that larger-scale discriminators have difficulty restoring realistic high-frequency STEM noise characteristics.

Using multiple discriminators at a single scale is proposed in [16] and extended to multiple scales in [7]. Following the assumption that images can be modelled as Markovian random fields, discriminators are applied to an array of non-overlapping image patches in [17]. However, discriminator arrays produce periodic artefacts[7] that have to be corrected by larger-scale discriminators. Instead, we prevent artefacts by applying multiple discriminators to random, possibly overlapping, regions at each scale.

If regions are selected using uniform random variates, pixels towards the edges of images will be examined less frequently. For a region of size \( u \times v \) in an image of size \( h \times w \), the number of regions covering a pixel at \((i,j), i \in \)
[0, h), j ∈ [0, w), i, j ∈ N_0 is
\[ c_{i,j} = \rho_h \rho_w \]
\[ \rho_h = \min(i + 1, h - i, u) \]
\[ \rho_w = \min(j + 1, w - j, v) \]

Uniform random region selection would therefore scale an effective generator learning rate at \((i, j)\) in proportion to \(c_{i,j}\). However, each output pixel has similar importance. To impose isotropy, we set the probability of each pixel being covered to \(c_{i,j}^{-1}/\text{mean}(c^{-1})\). Another option is to directly scale losses by \(c_{i,j}^{-1}/\text{mean}(c^{-1})\). However, this would increase gradient variance, potentially destabilizing learning. Reflection or other padding can also be used to adjust coverage; however, it would introduce discriminators to unnatural artefacts.

For \(N = 3\) discriminator scales with numbers, \(N_1, N_2\) and \(N_3\), of discriminators, \(D_1, D_2\) and \(D_3\), respectively, the total discriminator loss is
\[ L_D = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{N_i} \left[ D_i(G(I_{\text{scan}}))^2 + (D_i(I_N) - 1)^2 \right]. \]

Here, discriminators learn to predict 1 and 0 labels for real and generated images, respectively. Following [18], losses are squared differences from labels; rather than the binomial cross entropy introduced in [4], as logarithms can increase gradient variance. We found that \(N_1 = N_2 = N_3 = 1\) is sufficient to produce realistic images. However, higher performance might be achieved with more discriminators e.g. 2 large, 8 medium and 32 small discriminators.

**Auxiliary trainer:** Following Inception, we introduce an auxiliary trainer network[19, 20] to provide a more direct path for gradients to back-propagate to the inner generator. Our auxiliary trainer learns to generate half-size completions, \(T(G_{\text{inner}}(I_{\text{scan}}))\), that minimize Huberised[21] MSEs from bilinearly downsampled, half-size blurred ground truths, \(I_{\text{half, blur}}\).
\[ L_{aux} = 2H(\lambda_{\text{trainer}} \text{MSE}(T(G_{\text{inner}}(I_{\text{scan}})) \), I_{\text{half, blur}}), \]

where \(H(x) := \min(x, x^{1/2})\) and \(\lambda_{\text{trainer}} = 200\).

**Generator:** Our generator consists of two subnetworks, similar to [7]. An inner generator generates large-scale features from a half-size partial scan that are combined with input embedded by an outer generator to generate a full-size completion. The generator subnetworks are cooperative as they try to generate realistic completions that minimize the adversarial loss
\[ L_{adv} = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{N_i} D_i(G(I_{\text{scan}}) - 1)^2. \]

We chose a hinge loss[22–24]; which has no \(D_i(I_N)\) term, to improve stability in the early stages of training.

Discriminators only assess the realism of generated micrographs; not if they are correct. To the lift degeneracy and prevent mode collapse, we condition adversarial training on a Huberised mean squared error between generated and blurred ground truth images,
\[ L_{\text{cond}} = H(\lambda_{\text{cond}} \text{MSE}(G(I_{\text{scan}}), I_{\text{blur}})), \]

where \(\lambda_{\text{cond}} = 200\). To compensate for varying noise levels, ground truth images were blurred by a 5×5 symmetric Gaussian kernel with a 2.5 px standard deviation. We also tried natural statistics losses, similar to [7, 8]. However, we found that non-adversarial MSE guidance converges to slightly lower MSEs and similar structural similarity indexes[25] for greyscale STEM images.

In addition, the inner generator subnetwork cooperates with the auxiliary trainer subnetwork to minimize \(L_{aux}\). Added together, the total generator loss is
\[ L_G = \lambda_{\text{adv}} L_{adv} + L_{\text{cond}} + \lambda_{\text{aux}} L_{aux}, \]

where \(\lambda_{adv}\) and \(\lambda_{aux}\) control the contributions of the adversarial and auxiliary losses, respectively. In our experiments, we chose \(\lambda_{adv} = 5\) and \(\lambda_{aux} = 0.5\).

### 2.3 Learning Policy

In this subsection, we discuss our training hyperparameters and learning protocol for the multi-scale conditional GAN summarized in fig. 3. Experiments can be found in section 4 and detailed architecture is in section 8.

**Optimizer:** Training is ADAM optimized[14] and has two stages. In the first stage, the generator and auxiliary trainer learn to minimize mean squared errors between their outputs and ground truth images. For the first 250000 iterations, we use a constant learning rate \(\eta_0 = 0.0003\) and a decay rate for the first moment of the momentum \(\beta_1 = 0.9\). The learning rate is then stepwise decayed to zero in eight steps over the next 250000 iterations i.e. to \(\eta = \eta_0 (1 – \text{floor}(8 \times \text{iter} – 250000)/250000)/8\). Similarly, \(\beta_1\) is stepwise linearly decayed to 0.5 in eight steps.

In the second stage, the generator and discriminators play an adversarial game conditioned on non-adversarial MSE guidance. For the next 250000 iterations, we use \(\eta = 0.0001\) and \(\beta_1 = 0.9\) for the generator and discriminators. In the final 250000 iterations, the generator learning rate is decayed to zero in eight steps while the discriminator learning rate remains constant. Similarly, generator and discriminator \(\beta_1\) is stepwise decayed to 0.5 in eight steps.

The advantage of \(\beta_1 = 0.5\) for adversarial training is demonstrated in [23]. Our decision to start at \(\beta_1 = 0.9\) aims to improve the initial rate of convergence. In the first stage, generator and auxiliary trainer parameters are both updated once per training step. In the second stage, all parameters are updated once per training step.
Our $10^6$ iteration learning policy is in-line with with other GANs, which reuse data for 200 epochs e.g. [7]. However, we note that validation errors do not plateau even if training is increased to $2 \times 10^6$ iterations. This suggests that performance may be substantially improved by further training. All training is performed with batch size 1 due to the large model size needed to complete 512 x 512 scans.

**Adaptive learning rate clipping:** To stabilize batch size 1 training, adaptive learning rate clipping (ALRC) was applied to limit outlier MSEs. Details are in section 7.

**Input normalization:** Partial scans, $I_{\text{scan}}$, input to the generator are linearly transformed to $I_{\text{scan}}' = (I_{\text{scan}} + 1)/2$, where $I_{\text{scan}}' \in [0, 1]$. The generator is trained to output ground truth crops in $[0, 1]$, which are linearly transformed to $[-1, 1]$. Generator outputs and ground truth crops in $[-1, 1]$ are directly input to the discriminators.

**Weight normalization:** All generator weights are weight normalized [26]. Following [26, 27], running mean-only batch normalization is applied to the output channels of every convolutional layer except the last. Channel means are tracked by exponential moving averages with decay rates of 0.99. Similar to [28], running mean-only batch normalization is frozen in the second half of training to improve stability.

**Spectral normalization:** Spectral normalization [23] is applied to the weights of each convolutional layer in the discriminators to control the Lipschitz constants of the discriminators. We use the power iteration method with one iteration per training step to enforce a spectral norm of 1 for each weight matrix.

Spectral normalization stabilizes training, reduces susceptibility to mode collapse and is independent of rank, encouraging discriminators to use more input features to inform decisions [23]. In contrast, weight normalization [26] and Wasserstein weight clipping [29] impose more arbitrary model distributions that may only partially match the target distribution.

**Activation:** In the generator, ReLU [30] non-linearities are applied after running mean-only batch normalization. In the discriminators, slope 0.2 leaky ReLU [31] non-linearities are applied after every convolution layer. Rectifier leakage encourages discriminators to use more of their inputs to inform decisions. Our choice of generator and discriminator non-linearities follows [7].

**Initialization:** Generator weights were initialized from a normal distribution with mean 0.00 and standard deviation 0.05. To apply weight normalization, an example scan is then propagated through the network. Each layer output is divided by its L2 norm and the layer weights assigned their division by the square root of the L2 normalized output’s standard deviation. There are no biases in the generator as running mean-only batch normalization would allow biases to grow unbounded c.f. batch normalization [32].

Discriminator weights were initialized from a normal distribution with mean 0.00 and standard deviation 0.03. Discriminator biases were zero initialized.

**Experience replay:** To reduce destabilizing discriminator oscillations [33], we used an experience replay [34, 35] with 50 examples. Prioritizing the replay of difficult experiences improves reinforcement learning [36] so we only replay hard examples. We define hard examples to be those that the generator has the highest conditional losses for. Examples were swapped into the experience replay on average $p_{\text{in}} = 0.2$ times per iteration and independently sampled without removal with probability $p_{\text{use}} = 0.2$.

In detail, examples were added to the experience replay if their conditional losses were higher than a threshold, $L_t$. This threshold was calculated from the first and second raw moments of the conditional losses, $L_{\text{cond},1}$ and $L_{\text{cond},2}$, using exponential moving averages

$$L_{\text{cond},1} = \beta_{\text{cond},1}L_{\text{cond},1} + (1 - \beta_{\text{cond},1})L_{\text{cond}} \quad (11)$$

$$L_{\text{cond},2} = \beta_{\text{cond},2}L_{\text{cond},2} + (1 - \beta_{\text{cond},2})L_{\text{cond}}^2 \quad (12)$$

$$\sigma_{\text{cond}}^2 = L_{\text{cond},2} - L_{\text{cond},1}^2 \quad (13)$$

where we chose $\beta_{\text{cond},1} = \beta_{\text{cond},2} = 0.99$.

To calculate $L_t$, we also use a moving average to monitor the rate that new examples were added to the replay,

$$p_{\text{in}} \rightarrow \begin{cases} \beta_{\text{in}}p_{\text{in}}, & L_{\text{cond}} < L_t \\ \beta_{\text{in}}p_{\text{in}} + (1 - \beta_{\text{in}}), & L_{\text{cond}} \geq L_{\text{in}} \end{cases} \quad (14)$$

where we chose $\beta_{\text{in}} = 0.97$.

The raw moments and rate that new examples are being added to the replay are combined to inform the update

$$L_t \rightarrow \begin{cases} \beta_{t}L_t + (1 - \beta_{t})(L_{\text{cond},1} + \Delta L^\dagger), & p_{\text{in}} < p_{\text{use}} \\ \beta_{t}L_t + (1 - \beta_{t})(L_{\text{cond},1} + \Delta L^\dagger), & p_{\text{in}} \geq p_{\text{use}} \end{cases} \quad (15)$$

where we chose $\beta_{t} = 0.99$ and incremented the buffer threshold by $\Delta L^\dagger = -\Delta L^\dagger = 5\sigma_{\text{cond}}$.

The calculation of $L_t$ might be improved by using small geometric; rather than arithmetic, adjustments, restricting it to being positive by adding the update $L_t \rightarrow \max(L_t, 0)$ and accounting for asymmetric conditional losses, $\Delta L^\dagger \neq -\Delta L^\dagger$. However, further improvements to $L_t$ accuracy are unlikely to have a significant effect on training.

Our method can be extended to the sampling probabilities of individual examples in the experience replay. This would allow the replay probabilities to be increased for examples with higher conditional losses. However, we did not feel this was necessary as we already select hard examples for the experience replay. In addition, high losses may be caused by momentary quirks in the early stages of training.
Figure 4: Non-adversarial completions of 512×512 1/20 coverage blurred spiral and grid-like partial scans. Images with regular patterns or predictable structure are accurately completed. Circles accentuate that the network cannot reliably complete irregular images where there is no information. Spiral coverage is more uniform and produces fewer artefacts.
Figure 5: More adversarial and non-adversarial completions of test set $512 \times 512$ 1/20 coverage blurred spiral partial scans. Adversarial completions have realistic noise characteristics and colouration whereas non-adversarial completions are blurry. The bottom row shows a failure case where detail is too fine for the generator to resolve. Enlarged $64 \times 64$ regions from the top left of each image are inset to ease comparison.
Figure 6: Mean squared errors on 20000 512×512 test set micrographs for training with adaptive learning rate clipping (ALRC), increased weighting for pixels with high running mean errors, ALRC and running weights, ALRC and fixed weights, and adversarial training. For comparison, errors are linearly transformed to have the same mean and variance. Non-adversarial errors increase away from the electron beam path. In contrast, adversarial errors are less structured. Enlarged 64×64 regions from the top left of each image are inset to ease comparison.

|                | Mean \((×10^{-3})\) | Std Dev \((×10^{-3})\) | Var Lap \((×10^{-3})\) |
|----------------|----------------------|------------------------|------------------------|
| Clipped        | 2.83                 | 1.17                   | 5.6                    |
| Running        | 3.20                 | 1.27                   | 5.7                    |
| Clipped, Running | 2.81               | 1.11                   | 5.6                    |
| Clipped, Fixed | 3.01                 | 1.11                   | 5.6                    |
| Adversarial    | 20.95                | 11.56                  | 394.8                  |

Table 1: Means and standard deviations of mean squared errors for each pixel on 20000 512×512 test set micrographs with intensities in [-1, 1]. Variances of Laplacians are for mean squared errors after linear transformation to unit variance.

3 Performance

To characterize our generator’s performance, we map its per pixel MSEs for adversarial and non-adversarial training. We also present example applications of our adversarial and non-adversarial networks to grid-like and spiral partial scans. Training inference time is 50 ms on our GTX 1080 Ti GPU, enabling live partial scan completion.

3.1 Network Error

Mean square errors for each pixel of our generator’s output for 20000 test set images are shown in fig. 6 and tabulated in table 1. Non-adversarial training with ALRC, weighting pixel errors by their running mean errors, ALRC running means or ALRC weighted by the errors for ALRC and running means all have similar structured errors: Errors increase away from the electron beam path are especially high at the output edges. In contrast, adversarial errors are higher as images have realistic noise characteristics. Adversarial errors are also less structured: the variance of the Laplacian for unit variance adversarial errors is 70× higher than for non-adversarial errors.

3.2 Example Scans

Examples application of a non-adversarial network to spiral and grid-like partial scans are shown in fig. 4. In practice, 1/20 scan coverage is sufficient to complete most micrographs from spiral partial scans. However, our network cannot reliably complete images with unpredictable structure in regions where there is no coverage. Performance is higher for spiral scans than grid-like scans as they have smaller gaps.

Adversarial and non-adversarial test set completions are compared in fig. 5 (and fig. 1). Adversarial completions have realistic noise characteristics whereas non-adversarial completions are blurry. Adversarial completions also have more accurate colouration and less structured spatial error variation.

4 Experiments

In this section, we present learning curves for some of our non-adversarial architecture and learning policy experiments. All learning curves are 2500 iteration boxcar averaged. For clarity, the first \(10^4\) iterations before the dashed lines, where loss rapidly decrease, are not shown.

Following [7], we used a multi-stage training protocol for our initial experiments. Inner and outer generator subnetworks were trained separately, then together. An alternative approach uses an auxiliary loss network for end-to-end training, similar to Inception [19, 20]. This can provide a more direct path for gradients to back-propagate to the start of the network and introduces an additional regularization mechanism. Experimenting, we connected an auxiliary trainer to the inner generator and trained the network in a single stage. As shown by fig. 7, auxiliary network supported end-to-end training is more stable and converges to lower errors.
Figure 7: Auxiliary inner generator guided training is more stable and reaches a lower error than two-stage training with fine-tuning.

Figure 8: Errors are lower if electron beam path information is concatenated to the network input. Errors are higher for extra residual connections between inner generator strided convolutions and symmetric transpositional convolutions.

Figure 9: Performance is higher for small first convolution kernels; 3×3 for the inner generator and 7×7 for the outer generator or both 3×3, than for large first convolution kernels; 7×7 for the inner generator and 17×17 for the outer generator.

For multi-stage learning curves, the first losses are reported for the inner generator. These are followed by an error spike where losses are reported for outer generator training while the inner generator is frozen. In the final stage, outer generator losses are reported as the inner and outer generator are fine-tuned together.

In encoder-decoders, residual connections [37] between strided convolutions and symmetric strided transpositional convolutions can be used to reduce information loss. This is common in noise removal networks where the output is similar to the input e.g. [38, 39]. However, symmetric residual connections are also used in encoder-decoder networks for semantic image segmentation [40] where the input and output are different. Consequently, we tried adding symmetric residual connections between strided and transpositional inner generator convolutions. As shown by fig. 8, extra residuals accelerate initial inner generator training. However, final errors are slightly higher and initial inner generator training converged to similar errors with and without symmetric residuals. Taken together, this suggests that symmetric residuals initially accelerate training by enabling the final inner generator layers to generate crude outputs though their direct connections to the first inner generator layers. However, the symmetric connections also provide a direct path for low-information outputs of the first layers to get to the final layers, obscuring the contribution of the inner generator’s skip-3 residual blocks (section 8) and lowering performance in the final stages of training.

Path information is concatenated to the partial scan input to the generator. In principle, the generator can infer electron beam paths from partial scans. However, the input signal is attenuated as it travels through the network [41]. In addition, path information would have to be deduced; rather than informing calculations in the first inner generator layers, decreasing efficiency. To compensate, paths used to generate partial scans from full scans are concatenated to inputs. As shown by fig. 8, concatenating path information reduces errors throughout training. Performance might be further improved by explicitly building sparsity into the network [42].

Large kernels are often used at the start of neural networks to increase their receptive field. This allows their first convolutions to be used more efficiently. The receptive
Learning curves for different learning rate (LR) schedules. Performance is higher after more iterations and for a learning rate of 0.0004; rather than 0.0002.

Training and validation errors for adaptive learning rate clipped quartic errors. Validation errors have not diverged.

Errors are lower for outputs in [0,1]; rather than [-1,1].

Errors are lower for small first convolution kernels; 3×3 for the inner generator and 7×7 for the outer generator or both 3×3, than for large first convolution kernels; 7×7 for the inner generator and 17×17 for the outer generator. This suggests that the generator does not make effective use of the larger 17×17 kernel receptive field and that the variability of the extra kernel parameters harms learning.

Learning curves for different learning rate schedules are shown in fig. 10. Increasing training iterations and doubling the learning rate from 0.0002 to 0.0004 lowers errors. Validation errors do not plateau for 10⁶ iterations in fig. 11, suggesting that continued training would improve performance. Validation errors were calculated once every 50 training iterations for all experiments.

The choice of output domain can affect performance. Training with a [0, 1] output domain is compared against [-1,1] for slope 0.01 leaky ReLU activation after every
Figure 14: Nearest neighbour infilling reduces error. Noise was not added to low duration path segments for this experiment.

Figure 15: Errors are similar with and without adding uniform noise to low-duration path segments.

Figure 16: Learning is more stable and converges to lower errors at lower learning rates. Errors are lower for spirals than grid-like paths and lowest when no noise is added to low-intensity path segments.

generator convolution in fig. 12. Although $[-1,1]$ is supported by leaky ReLUs, requiring orders of magnitude differences in scale for $[-1,0]$ and $(0,1]$ hinders learning. To limit dependence on the choice output domain, we do not apply batch normalization or activation after the last generator convolutions in our final architecture.

The $[0,1]$ outputs of fig. 12 were linearly transformed to $[-1,1]$ and passed through a tanh non-linearity. This ensured that $[0,1]$ output errors were on the same scale as $[-1,1]$ output errors, maintaining the same effective learning rate. Initially, outputs were clipped by a tanh non-linearity to limit outputs far from the target domain from perturbing training. However, fig. 13 shows that errors are similar without end non-linearities so they were removed. Fig. 13 also shows that replacing slope 0.01 leaky ReLUs with ReLUs and changing all kernel sizes to $3 \times 3$ has little effect. Swapping to ReLUs and $3 \times 3$ kernels is therefore an option to reduce computation. Nevertheless, we continue to use larger kernels throughout as we think they would usefully increase the receptive field with more stable, larger batch size training.

To more efficiently use the first generator convolutions, we nearest neighbour infilled noiseless partial scans. As shown by fig. 14, infilling reduces error. However, infilling is expected to be of limited use for low-dose applications as scans can be noisy, making meaningful infilling difficult. Nevertheless, nearest neighbour partial scan infilling is a computationally inexpensive method to improve generator performance for high-dose applications.

To investigate our generator’s ability to handle STEM noise[15], we combined uniform noise with partial scans of Gaussian blurred STEM images, as described by eqn. 1. More noise was added to low intensity path segments and low-intensity pixels. As shown by fig. 15, ablating the noise associated with low-duration path segments increases performance.

Fig. 16 shows that spiral path training is more stable and reaches lower errors at lower learning rates. At the same learning rate, spiral paths converge to lower errors than grid-like paths as spirals have more uniform coverage. Errors are much lower for spiral paths when both intensity- and duration-dependent noise is ablated.

To choose a training optimizer, we completed training with stochastic gradient descent, momentum, Nesterov-accelerated momentum [44, 45], RMSProp[46] and
Figure 17: Adaptive momentum-based optimizers, ADAM and RMSProp, outperform non-adaptive momentum optimizers, including Nesterov-accelerated momentum. ADAM outperforms RMSProp; however, training hyperparameters and learning protocols were tuned for ADAM.

Figure 18: Increasing partial scan coverage decreases training errors.

ADAM[14]. Adaptive momentum optimizers, ADAM and RMSProp, outperform the non-adaptive optimizers. Non-adaptive momentum-based optimizers outperform momentumless stochastic gradient decent. ADAM slightly outperforms RMSProp; however, architecture and learning policy were tuned for ADAM. This suggests that RMSProp optimization may also be a good choice.

Learning curves for 1/10, 1/20, 1/40 and 1/100 coverage spiral scans are shown in fig. 18. In practice, 1/20 coverage is sufficient for most STEM images. A non-adversarial generator can complete test set 1/20 coverage partial scans with a 2.6% root mean squared intensity error. Nevertheless, higher coverage is needed to resolve fine detail in some images. Likewise, lower coverage may be appropriate for images without fine detail. Consequently, we are considering the development of an intelligent scan system that adjusts coverage based on micrograph content.

Training is performed with batch size 1 due to the large network size needed for 512 × 512 partial scans. However, MSE training is unstable and large error spikes destabilize training. To stabilize learning, we developed adaptive learning rate clipping (ALRC, section 7) to limit the magnitudes of outlier losses while preserving their distributions. ALRC is compared against MSE, Huberised MSE, and weighting each pixel’s error by its Huberised running mean and fixed final errors in fig. 19. ALRC results in more stable training with the fastest convergence and lowest errors. Similar improvements are confirmed for CIFAR-10 supersampling in section 7.

5 Conclusions

We have demonstrated that adversarial deep learning can realistically complete partial scans with less than 1/20 coverage. This will enable faster imaging and new beam-sensitive applications. High performance is achieved by the introduction of an auxiliary trainer network and adaptive learning rate clipping of outlier losses.

6 Acknowledgements

This research was funded by EPSRC grant EP/N035437/1.
References

[1] A. B. Yankovich, B. Berkels, W. Dahmen, P. Binev, and P. M. Voyles, “High-precision scanning transmission electron microscopy at coarse pixel sampling for reduced electron dose,” *Advanced Structural and Chemical Imaging*, vol. 1, no. 1, p. 2, 2015.

[2] S. G. Wolf, E. Shimoni, M. Elbaum, and L. Houben, “Stem tomography in biology,” in *Cellular Imaging*, pp. 33–60, Springer, 2018.

[3] A. Garcia, A. M. Raya, M. M. Mariscal, R. Esparza, M. Herrera, S. I. Molina, G. Scavello, P. L. Galindo, M. Jose-Yacaman, and A. Ponce, “Analysis of electron beam damage of exfoliated mos2 sheets and quantitative haadf-stem imaging,” *Ultramicroscopy*, vol. 146, pp. 33–38, 2014.

[4] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial nets,” in *Advances in neural information processing systems*, pp. 2672–2680, 2014.

[5] M. Mirza and S. Osindero, “Conditional generative adversarial nets,” *arXiv preprint arXiv:1411.1784*, 2014.

[6] D. Bang and H. Shim, “Mgan: Solving mode collapse using manifold guided training,” *arXiv preprint arXiv:1804.04391*, 2018.

[7] T.-C. Wang, M.-Y. Liu, J.-Y. Zhu, A. Tao, J. Kautz, and B. Catanzaro, “High-resolution image synthesis and semantic manipulation with conditional gans,” *arXiv preprint arXiv:1711.11585*, 2017.

[8] A. B. L. Larsen, S. K. Sønderby, H. Larochelle, and O. Winther, “Autoencoding beyond pixels using a learned similarity metric,” *arXiv preprint arXiv:1512.09300*, 2015.

[9] X. Wu, R.-L. Li, F.-L. Zhang, J.-C. Liu, J. Wang, A. Shamir, and S.-M. Hu, “Deep portrait image completion and extrapolation,” *arXiv preprint arXiv:1808.07757*, 2018.

[10] G. Liu, F. A. Reda, K. J. Shih, T.-C. Wang, A. Tao, and B. Catanzaro, “Image inpainting for irregular holes using partial convolutions,” in *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 85–100, 2018.

[11] W. Yang, X. Zhang, Y. Tian, W. Wang, and J.-H. Xue, “Deep learning for single image super-resolution: A brief review,” *arXiv preprint arXiv:1808.03344*, 2018.

[12] X. Li, O. Dyck, S. V. Kalinin, and S. Jesse, “Compressed sensing of scanning transmission electron microscopy (stem) with nonrectangular scans,” *Microscopy and Microanalysis*, vol. 24, no. 6, pp. 623–633, 2018.

[13] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, *et al.*, “Tensorflow: A system for large-scale machine learning,” in *OSDI*, vol. 16, pp. 265–283, 2016.

[14] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014.

[15] T. Seki, Y. Ikuhara, and N. Shibata, “Theoretical framework of statistical noise in scanning transmission electron microscopy,” *Ultramicroscopy*, vol. 193, pp. 118–125, 2018.

[16] I. Durugkar, I. Gemp, and S. Mahadevan, “Generative multi-adversarial networks,” *arXiv preprint arXiv:1611.01673*, 2016.

[17] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, “Image-to-image translation with conditional adversarial networks,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1125–1134, 2017.

[18] X. Mao, Q. Li, H. Xie, R. Y. Lau, Z. Wang, and S. Paul Smolley, “Least squares generative adversarial networks,” in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2794–2802, 2017.

[19] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going deeper with convolutions,” *arXiv abs/1409.4842*, URL http://arxiv.org/abs/1409.4842, 2014.

[20] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, “Rethinking the inception architecture for computer vision,” *arXiv 2015*, arXiv preprint arXiv:1512.00567, vol. 2015.

[21] P. J. Huber, “Robust estimation of a location parameter,” *The annals of mathematical statistics*, pp. 73–101, 1964.

[22] J. H. Lim and J. C. Ye, “Geometric gan,” *arXiv preprint arXiv:1705.02894*, 2017.

[23] T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida, “Spectral normalization for generative adversarial networks,” *arXiv preprint arXiv:1802.05957*, 2018.
[24] A. Brock, J. Donahue, and K. Simonyan, “Large scale gan training for high fidelity natural image synthesis,” arXiv preprint arXiv:1809.11096, 2018.

[25] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, “Image quality assessment: from error visibility to structural similarity,” IEEE transactions on image processing, vol. 13, no. 4, pp. 600–612, 2004.

[26] T. Salimans and D. P. Kingma, “Weight normalization: A simple reparameterization to accelerate training of deep neural networks,” in Advances in Neural Information Processing Systems, pp. 901–909, 2016.

[27] E. Hoffer, R. Banner, I. Golan, and D. Soudry, “Norm matters: efficient and accurate normalization schemes in deep networks,” in Advances in Neural Information Processing Systems, pp. 2160–2170, 2018.

[28] L.-C. Chen, G. Papandreou, F. Schroff, and H. Adam, “Rethinking atrous convolution for semantic image segmentation,” arXiv preprint arXiv:1706.05587, 2017.

[29] S. C. Arjovsky, Martin and L. Bottou, “Wasserstein gan,” arXiv preprint arXiv:1701.07875, 2017.

[30] V. Nair and G. E. Hinton, “Rectified linear units improve restricted boltzmann machines,” in Proceedings of the 27th international conference on machine learning (ICML-10), pp. 807–814, 2010.

[31] A. L. Maas, A. Y. Hannun, and A. Y. Ng, “Rectifier nonlinearities improve neural network acoustic models,” in Proc. icml, vol. 30, p. 3, 2013.

[32] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” arXiv preprint arXiv:1502.03167, 2015.

[33] K. J. Liang, C. Li, G. Wang, and L. Carin, “Generative adversarial network training is a continual learning problem,” arXiv preprint arXiv:1811.11083, 2018.

[34] D. Pfau and O. Vinyals, “Connecting generative adversarial networks and actor-critic methods,” arXiv preprint arXiv:1610.01945, 2016.

[35] e. a. Srivastava, A., “Learning from simulated and unsupervised images through adversarial training,” arXiv preprint arXiv:1612.07828, 2016.

[36] e. a. Schaul, Tom, “Prioritized experience replay,” arXiv preprint arXiv:1511.05952, 2015.

[37] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 770–778, 2016.

[38] X.-J. Mao, C. Shen, and Y.-B. Yang, “Image restoration using convolutional auto-encoders with symmetric skip connections,” arXiv preprint arXiv:1606.08921, 2016.

[39] L. Casas, N. Navab, and V. Belagiannis, “Adversarial signal denoising with encoder-decoder networks,” arXiv preprint arXiv:1812.06555, 2018.

[40] V. Badrinarayanan, A. Kendall, and R. Cipolla, “Segnet: A deep convolutional encoder-decoder architecture for image segmentation (segnet),” The Computing Research Repository (CoRR), abs/1511.00561, 2015.

[41] H. Zheng, J. Yao, Y. Zhang, and I. W. Tsang, “Degeneration in vae: in the light of fisher information loss,” arXiv preprint arXiv:1802.06677, 2018.

[42] B. Graham, “Spatially-sparse convolutional neural networks,” arXiv preprint arXiv:1409.6070, 2014.

[43] H. W. Lin, M. Tegmark, and D. Rolnick, “Why does deep and cheap learning work so well?,” Journal of Statistical Physics, vol. 168, no. 6, pp. 1223–1247, 2017.

[44] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, “On the importance of initialization and momentum in deep learning,” in International conference on machine learning, pp. 1139–1147, 2013.

[45] Y. Nesterov, “A method of solving a convex programming problem with convergence rate o (1/k2),” in Soviet Mathematics Doklady, vol. 27, pp. 372–376, 1983.

[46] G. Hinton, N. Srivastava, and K. Swersky, “Neural networks for machine learning lecture 6a overview of mini-batch gradient descent,” Cited on, p. 14, 2012.

[47] A. Krizhevsky, V. Nair, and G. Hinton, “The cifar-10 dataset,” online: http://www.cs.toronto.edu/~kriz/cifar.html, vol. 55, 2014.

[48] A. Krizhevsky and G. Hinton, “The cifar-10 dataset,” online: http://www.cs.toronto.edu/~kriz/cifar.html, vol. 55, 2014.

[49] X. Glorot and Y. Bengio, “Understanding the difficulty of training deep feedforward neural networks,” in Proceedings of the thirteenth international conference on artificial intelligence and statistics, pp. 249–256, 2010.
7 Adaptive Learning Rate Clipping

To stabilize small batch size training, we developed adaptive learning rate clipping (ALRC, algorithm 1) as a computationally inexpensive method to limit outlier losses while preserving their distributions. In section 4, we showed that ALRC stabilized partial scan completion training converges faster and achieves lower final errors than other methods. To validate ALRC, we investigate its ability to stabilize the training of supersampling networks that upsample CIFAR-10[47, 48] images to $32 \times 32 \times 3$ after downsampling to $16 \times 16 \times 3$.

**Data pipeline:** In order, images were randomly flipped left or right, had their brightness distorted, had their contrast distorted, were linearly transformed to have zero mean and unit variance and bilinearly downsampled to $16 \times 16 \times 3$.

**Architecture:** Images were upsampled and passed through the convolutional network in fig. 20. Each convolution is followed by ReLU activation, except the last. All weights were Xavier[49] initialized. Biases were zero initialized.

**Learning policy:** ADAM optimization was used with the hyperparameters recommended in [14] and a base learning rate of $1/1280$ for 100000 iterations. The learning rate was constant in batch size 1, 4, 16 experiments and decreased to $1/12800$ after 54687 iterations in batch size 64 experiments. Networks were trained to minimize mean squared or quartic errors between restored and ground truth images. Adaptive learning rate clipping was applied to limit the magnitudes of losses to either 2, 3, 4 or $\infty$ standard deviations above their running means. For batch sizes above 1, ALRC was applied to each loss individually.

**Experiments:** Example learning curves for mean squared and quartic error training are shown in fig. 21. Training is more stable and converges to lower errors for larger batch sizes. Training is less stable for quartic errors than squared errors, allowing ALRC to be examined for loss functions with different stability.

Training was repeated 10 times for each combination of adaptive learning rate threshold and batch size. Means and standard deviations of the means of the last 5000 training losses for each experiment are tabulated in table 2. Adaptive learning rate clipping has no effect on squared error training, even for batch size 1. However, it decreases errors for batch sizes 1, 4 and 16 for quartic error training.

Putting the results together, ALRC is only effective if there are large error spikes that would destabilize convergence. This situation is often encountered when using a high learning rate. However, we are using a moderate learning rate so squared errors are not spiking high enough to destabilize convergence. ALRC is less effective for large batch sizes because averaging decreases the gradients of outlier losses.

**Algorithm 1** Adaptive learning rate clipping (ALRC) of outlier losses. In our experiments, $\mu_1 = 25$, $\mu_2 = 30^2$, $\beta_1 = \beta_2 = 0.999$ and $n = 3$ after initialization.

| Input Images, $16\times16\times3$ |
|-----------------------------------|
| Bilinear Upsamp, x2 |
| Conv 64, 5x5 |
| Conv 128, 3x3 |
| Conv 128, 3x3 |
| Conv 128, 3x3 |
| Conv 128, 3x3 |
| Conv 128, 3x3 |
| Conv 128, 3x3 |
| Conv 128, 3x3 |
| Conv 64, 3x3 |
| Conv 3, 5x5 |
| Output Images, $32\times32\times3$ |

**Figure 20:** Image $2 \times$ upsampling network with 3 residual blocks.

ALRC is easy to implement for arbitrary losses and batch sizes. An implementation is included in https://github.com/Jeffrey-Ede/partial-STEM. In addition, ALRC can be extended to other properties of loss distributions. We also experimented with transformations of error distributions to constant distributions. However, we found that this made networks numerically unstable partway through training.
Figure 21: Unclipped learning curves for $2 \times$ CIFAR-10 upsampling with batch sizes 1, 4, 16 and 64 with and without adaptive learning rate clipping of losses to 3 standard deviations above their running means. Training is more stable for squared errors than quartic errors. Learning curves are 500 iteration boxcar averaged.

Squred Errors

| Threshold | Batch Size 1 | Batch Size 4 | Batch Size 16 | Batch Size 64 |
|-----------|--------------|--------------|---------------|---------------|
|           | Mean         | Std Dev      | Mean          | Std Dev       |
| 2         | 5.55         | 0.048        | 4.96          | 0.016         |
| 3         | 5.52         | 0.054        | 4.96          | 0.029         |
| 4         | 5.56         | 0.048        | 4.97          | 0.017         |
| $\infty$ | 5.55         | 0.041        | 4.98          | 0.017         |

Quartic Errors

| Threshold | Batch Size 1 | Batch Size 4 | Batch Size 16 | Batch Size 64 |
|-----------|--------------|--------------|---------------|---------------|
|           | Mean         | Std Dev      | Mean          | Std Dev       |
| 2         | 3.54         | 0.084        | 3.02          | 0.023         |
| 3         | 3.59         | 0.055        | 3.08          | 0.024         |
| 4         | 3.61         | 0.054        | 3.13          | 0.023         |
| $\infty$ | 3.88         | 0.108        | 3.32          | 0.037         |

Table 2: Adaptive learning rate clipping for losses 2, 3, 4 and $\infty$ running standard deviations above their running means for batch sizes 1, 4, 16 and 64. Each squared and quartic error mean and standard deviation is the mean of the final 5000 training errors of 10 experiments. Adaptive learning rate clipping lowers errors for unstable quartic error training at low batch sizes and otherwise has little effect. Means and standard deviations are multiplied by 100.
Figure 22: Two-stage generator that completes $512 \times 512$ micrographs from partial scans. A dashed line indicates that the same image is input to the inner and outer generator. Large scale features developed by the inner generator are locally enhanced by the outer generator and turned into images. An auxiliary trainer network restores images from inner generator features to provide direct feedback.
8 Detailed Architecture

Figure 23: Discriminators examine random $w \times w$ crops to predict whether complete scans are real or generated. Generators are trained by multiple discriminators with different $w$.

Generator and inner generator trainer architecture is shown in fig. 22. Discriminator architecture is shown in fig. 23. The components in our networks are

**Bilinear Downsamp, $wxw$:** This is an extension of linear interpolation in one dimension to two dimensions. It is used to downsample images to $w \times w$.

**Bilinear Upsamp, $xs$:** This is an extension of linear interpolation in one dimension to two dimensions. It is used to upsample images by a factor of $s$.

**Conv $d$, $wxw$, Stride, $x$:** Convolution with a square kernel of width, $w$, that outputs $d$ feature channels. If the stride is specified, convolutions are only applied to every $x$th spatial element of their input, rather than to every element. Striding is not applied depthwise.

**Linear, $d$:** Flatten input and fully connect it to $d$ feature channels.

**Random Crop, $wxw$:** Randomly sample a $w \times w$ spatial location using an external probability distribution.

‡: Circled plus signs indicate residual connections where incoming tensors are added together. These help reduce signal attenuation and allow the network to learn perturbative transformations more easily.

All generator convolutions are followed by running mean-only batch normalization then ReLU activation, except output convolutions. All discriminator convolutions are followed by slope 0.2 leaky ReLU activation.