A. Additional Experimental Details

The outer loop loss function is domain specific. In the supervised experiments on MNIST and CIFAR, the outer loop loss was cross-entropy for logistic regression on real MNIST or CIFAR data. The inner-loop loss matches the outer-loop loss, but with synthetic data instead of real data. Appendix H describes the losses for the RL experiments.

The following equation defines the inner-loop SGD with momentum update for the learner parameters θ_t.

$$\theta_{t+1} = \theta_t - \alpha \sum_{0 \leq t' \leq t} \beta^{t-t'} \nabla \ell_{\text{inner}}(G(\mathbf{z}_{t'}, \mathbf{y}_{t'}), \mathbf{y}_{t'}, \theta_{t'}), \tag{1}$$

where α and β are the learning rate and momentum hyperparameters, respectively. $\mathbf{z}_t$ is a batch of noise vectors that are input to the generator and are sampled from a unit-variance Gaussian. $\mathbf{y}_t$ are a batch of labels for each generated sample/input and are sampled uniformly from all available class labels. Instead of randomly sampling $\mathbf{z}_t$, we can also learn a curriculum by additionally optimizing $\mathbf{z}_t$ directly and keeping $\mathbf{y}_t$ fixed throughout all of training. Results for both approaches (and additional curriculum ablations) are reported in Section 3.2.

A.1. MNIST Experiments:

For the GTN training for MNIST we sampled architectures from a distribution that produces architectures with convolutional (conv) and fully-connectd (FC) layers. All architectures had 2 conv layers, but the number of filters for each layer was sampled uniformly from the ranges $U([32, 128])$ and $U([64, 256])$, respectively. After each conv layer there is a max pooling layer for dimensionality reduction. After the last conv layer, there is a fully-connected layer with number of filters sampled uniformly from the range $U([64, 256])$. We used Kaiming Normal initialization (He et al., 2015) and LeakyReLUs (Maas et al., 2013) (with $\alpha = 0.1$). We use BatchNorm (Ioffe & Szegedy, 2015) for both the generator and the learners. The BatchNorm momentum for the learner was set to 0 (meta-training consistently converged to small values and we saw no significant gain from learning the value).

The generator consisted of 2 FC layers (1024 and $128 * H/4 * H/4$ filters, respectively, where H is the final width of the synthetic image). After the last FC layer there are 2 conv layers. The first conv has 64 filters. The second conv has 1 filter followed by a Tanh. We found it particularly important to normalize (mean of zero and variance of one) all datasets. Hyperparameters are shown in Table 2.

Hyperparameter	Value
Learning Rate	0.01
Initial LR	0.02
Initial Momentum	0.5
Adam Beta_1	0.9
Adam Beta_2	0.999
Size of latent variable	128
Inner-Loop Batch Size	128
Outer-Loop Batch Size	128

Table 2. Hyperparameters for MNIST experiments

A.2. CIFAR10 Experiments:

For GTN training for CIFAR-10, the template architecture is a small learner with 5 convolutional layers followed by a global average pooling and an FC layer. The second and fourth convolution had stride=2 for dimensionality reduction. The number of filters of the first conv layer was sampled uniformly from the range $U([32, 128])$ while all others were sampled uniformly from the range $U([64, 256])$. Other details including the generator architecture were the same as the MNIST experiments, except the CIFAR generator's second conv layer had 3 filters instead of 1. Hyperparameters used can be found in Table 3. For CIFAR10 we augmented the real training set when training GTNs with random crops and horizontal flips. We do not add weight normalization to the final architectures found during architecture search, but we do so when we train architectures with GTN-generated data during architecture search to provide an estimate of their asymptotic performance.

Hyperparameter	Value
Learning Rate	0.002
Initial LR	0.02
Initial Momentum	0.5
Adam Beta_1	0.9
Adam Beta_2	0.9
Adam ϵ	1e-5
Size of latent variable	128
Inner-loop Batch Size	128
Outer-loop Batch Size	256

Table 3. Hyperparameters for CIFAR10 experiments

B. Reasons GTNs are not expected to produce SOTA accuracy vs. asymptotic performance when training on real data

There are three reasons not to expect SOTA accuracy levels for the learners trained on synthetic data: (1) we train for very few SGD steps (32 or 128 vs. tens of thousands), (2) SOTA performance results from architectures explicitly designed (with much human effort) to achieve record accuracy, whereas GTN produces compressed training data optimized to generalize across diverse architectures with the aim of quickly evaluating a new architecture's potential, and (3) SOTA methods often use data outside of the benchmark dataset and complex data-augmentation schemes.

C. Cell Search Space

When searching for the operations in a CNN cell, the 11 possible operations are listed below.

- identity
- 1×1 convolution
- 3×3 convolution
- $1 \times 3 + 3 \times 1$ convolution
- $1 \times 7 + 7 \times 1$ convolution
- 2×2 max pooling
- 3×3 max pooling
- 5×5 max pooling
- 2×2 average pooling
- 3×3 average pooling
- 5×5 average pooling

D. Computation And Memory Complexity

With the traditional training of DNNs with back-propagation, the memory requirements are proportional to the size of the network because activations during the forward propagation have to be stored for the backward propagation step. With meta-gradients, the memory requirement also grows with the number of inner-loop steps because all activations and weights have to be stored for the 2nd order gradient to be computed. This becomes impractical for large networks and/or many inner-loop steps. To reduce the memory requirements, we utilize gradient-checkpointing (Griewank & Walther, 2000) by only storing the computed weights of learner after each inner-loop step and recomputing the activations during the backward pass. This trick allows us to compute meta-gradients for networks with 10s of millions of parameters over

hundreds of inner-loop steps in a single GPU. While in theory the computational cost of computing meta-gradients with gradient-checkpointing is 4x larger than computing gradients (and 12x larger than forward propagation), in our experiments it is about 2.5x slower than gradients through backpropagation due to parallelism. We could further reduce the memory requirements by utilizing reversable hypergradients (Maclaurin et al., 2015), but, in our case, we were not constrained by the number of inner-loop steps we can store in memory.

E. Extended NAS results

In the limited computation regime (less than 1 day of computation), the best methods were, in order, GHN, ENAS, GTN, and NAONet with a mean error of 2.84%, 2.89%, 2.92%, and 2.93%, respectively. A 0.08% difference on CIFAR10 represents 8 out of the 10k test samples. For that reason, we consider all of these methods as state of the art. Note that out of the four, GTN is the only one relying on Random Search for architecture proposal.

Table 4. Performance of different architecture search methods. Search with our method required 16h total, of which 8h were spent training the GTN and 8h were spent evaluating 800 architectures with GTN-produced synthetic data. Our results report mean ± SD of 5 evaluations of the same architecture with different initializations. It is common to report scores with and without Cutout (DeVries & Taylor, 2017), a data augmentation technique used during training.We found better architectures compared to other methods using random search (Random-WS and GHN-Top) and are competitive with algorithms that benefit from more advanced search methods (e.g. NAONet and ENAS employ non-random architecture proposals for performance gains; GTNs could be combined with such non-random proposals, which would likely further improve performance). Increasing the width of the architecture found (F=128) further improves performance.
* Used a slightly different random search method that is not directly comparable with the others.

Model	Error(%)	#params	Random	GPU Days
NASNet-A (Zoph & Le, 2017)	3.41	3.3M	✗	2000
AmoebaNet-B + Cutout (Real et al., 2019)	2.13	34.9M	✗	3150
DARTS + Cutout (Liu et al., 2018b)	2.83	4.6M	✗	4
NAONet + Cutout (Luo et al., 2018)	2.48	10.6M	✗	200
NAONet-WS (Luo et al., 2018)	3.53	2.5M	✗	0.3
NAONet-WS + Cutout (Luo et al., 2018)	2.93	2.5M	✗	0.3
ENAS (Pham et al., 2018)	3.54	4.6M	✗	0.45
ENAS + Cutout (Pham et al., 2018)	2.89	4.6M	✗	0.45
GHN Top-Best + Cutout (Zhang et al., 2018)	2.84 ± 0.07	5.7M	✓*	0.84
Random Search + Weight Sharing (Li & Talwalkar, 2019)	2.71	4.3M	✓*	9.7
GHN Top (Zhang et al., 2018)	4.3 ± 0.1	5.1M	✓	0.42
Random-WS (Luo et al., 2018)	3.92	3.9M	✓	0.25
Random Search + Real Data (baseline)	3.88 ± 0.08	12.4M	✓	10
RS + Real Data + Cutout (baseline)	3.02 ± 0.03	12.4M	✓	10
RS + Real Data + Cutout (F=128) (baseline)	2.51 ± 0.13	151.7M	✓	10
Random Search + GTN (ours)	**3.84** ± 0.06	8.2M	✓	0.67
Random Search + GTN + Cutout (ours)	**2.92** ± 0.06	8.2M	✓	0.67
RS + GTN + Cutout (F=128) (ours)	**2.42** ± 0.03	97.9M	✓	0.67

F. Conditioned Generator vs. XY-Generator

Our experiments in the main paper conditioned the generator to create data with given labels, by concatenating a one-hot encoded label to the input vector. We also explored an alternative approach where the generator itself produced a target probability distribution to label the data it generates. Because more information is encoded into a soft label than a one-hot encoded one, we expected an improved training set to be generated by this variant. Indeed, such a "dark knowledge" distillation setup has been shown to perform better than learning from labels (Hinton et al., 2015). However, the results in Figure 10 indicate that jointly generating both images and their soft labels under-performs generating only images, although the result could change with different hyperparameter values and/or innovations that improve the stability of training.

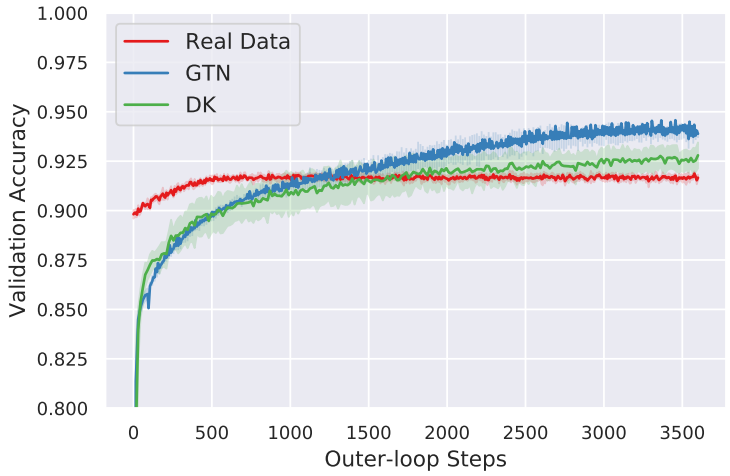


Figure 10. Comparison between a conditional generator and a generator that outputs an image/label pair. We expected the latter "dark knowledge" approach to outperform the conditional generator, but that does not seem to be the case. Because initialization and training of the dark knowledge variant were more sensitive, we believe a more rigorous tuning of the process could lead to a different result.

G. GTN generates (seemingly) endless data

While optimizing images directly (i.e. optimizing a fixed tensor of images) would result in a fixed number of samples, optimizing a generator can potentially result in an unlimited amount of new samples. We tested this generative capability by generating more data during evaluation (i.e. with no change to the meta-optimization procedure) in two ways. In the first experiment, we increase the amount of data in each inner-loop optimization step by increasing the batch size (which results in lower variance gradients). In the second experiment, we keep the number of samples per batch fixed, but increase the number of inner-loop optimization steps for which a new network is trained. Both cases result in an increased amount of training data. If the GTN generator has overfit to the number of inner-loop optimization steps during meta-training and/or the batch size, then we would not expect performance to improve when we have the generator produce more data. However, an alternate hypothesis is that the GTN is producing a healthy distribution of training data, irrespective of exactly how it is being used. Such a hypothesis would be supported by performance increase in these experiments.

Figure 11 shows performance as a function of increasing batch size (beyond the batch size used during meta-optimization, i.e. 128). The increase in performance of GTN means that we can sample larger training sets from our generator (with diminishing returns) and that we are not limited by the choice of batch size during training (which is constrained due to both memory and computation requirements).

Figure 12 shows the results of generating more data by increasing the number of inner-loop optimization steps. Generalization to more inner-loop optimization steps is important when the number of inner-loop optimization steps used during meta-optimization is not enough to achieve maximum performance. This experiment also tests the generalization of the optimizer hyperparameters because they were optimized to maximize learner performance after a fixed number of steps. There is an increase in performance of the learner trained on GTN-generated data as the number of inner-loop optimization steps is increased, demonstrating that the GTN is producing generally useful data instead of overfitting to the number of inner-loop optimization steps during training (Figure 12). Extending the conclusion from Figure 5, in the very low data regime, GTN is significantly better than training on real data ($p < 0.05$). However, as more inner-loop optimization steps are taken and thus more unique data is available to the learner, training on the real data becomes more effective than learning from synthetic data ($p < 0.05$) (see Figure 12).

Another interesting test for our generative model is to test the distribution of learners after they have trained on the synthetic data. We want to know, for instance, if training on synthetic samples from one GTN results in a functionally similar set of learner weights regardless of learner initialization (this phenomena can be called learner mode collapse). Learner mode collapse would prevent the performance gains that can be achieved through ensembling diverse learners. We tested for learner mode collapse by evaluating the performance (on held-out data and held-out architecture) of an ensemble of 32

randomly initialized learners that are trained on independent batches from the *same GTN*. To construct the ensemble, we average the predicted probability distributions across the learners to compute a combined prediction and accuracy. The results of this experiment can be seen in Figure 15, which shows that the combined performance of an ensemble is better (on average) than an individual learner, providing additional evidence that the distribution of synthetic data is healthy and allows ensembles to be harnessed to improve performance, as is standard with networks trained on real data.

H. GTN for RL

To demonstrate the potential of GTNs for RL, we tested our approach with a small experiment on the classic CartPole test problem (see Brockman et al. (2016) for details on the domain. We conducted this experiment before the discovery that weight normalization improves GTN training, so these experiments do not feature it; it might further improve performance. For this experiment, the meta-objective the GTN is trained with is the advantage actor-critic formulation: $\log \pi(a|\theta_\pi)(R - V(s; \theta_v))$ (Mnih et al., 2016). The state-value V is provided by a separate neural network trained to estimate the average state-value for the learners produced so far during meta-training. The learners train on synthetic data via a single-step of SGD with a batch size of 512 and a mean squared error regression loss, meaning the inner loop is supervised learning. The outer-loop is reinforced because the simulator is non-differentiable. We could have also used an RL algorithm in the inner loop. In that scenario the GTN would have to learn to produce an entire synthetic world an RL agent would learn in. Thus, it would create the initial state and then iteratively receive actions and generate the next state and optionally a reward.

For example, a GTN could learn to produce an entire MDP that an agent trains on, with the meta-objective being that the trained agent then performs well on a target task. We consider such synthetic (PO)MDPs an exciting direction for future research.

The score on CartPole is the number of frames out of 200 for which the pole is elevated. Both GTN and an A2C (Mnih et al., 2016) control effectively solve the problem (Figure 16). Interestingly, training GTNs takes the same number of simulator steps as training a single learner with policy-gradients (Figure 16). Incredibly, however, once trained, the synthetic data from a GTN can be used to train a learner to maximum performance in a single SGD step! While that is unlikely to be true for harder target RL tasks, these results suggest that the speed-up for architecture search from using GTNs in the RL domain can be even greater than in supervised domain.

The CartPole experiments feature a single-layer neural network with 64 hidden units and a tanh activation function for both the policy and the value network. The inner-loop batch size was 512 and the number of inner-loop training iterations was 1. The observation space of this environment consists of a real-valued vector of size 4 (Cart position, Cart velocity, Pole position, Pole velocity). The action space consists of 2 discrete actions (move left or move right). The outer loop loss is the reward function for the target domain (here, pole-balancing). The inner loop loss is mean squared error (i.e. the network is doing supervised learning on the state-action mapping pairs provided by the GTN).

I. Solving mode collapse in GANs with GTNs

We created an implementation of generative adversarial networks (GANs) (Goodfellow et al., 2014) and found they tend to generate the same class of images (e.g. only 1s, Figure 17), which is a common training pathology in GANs known as mode collapse (Srivastava et al., 2017). While there are techniques to prevent mode collapse (e.g. minibatch discrimination and historical averaging (Salimans et al., 2016)), we hypothesized that combining the ideas behind GTNs and GANs might provide a different, additional technique to help combat mode collapse. The idea is to add a discriminator to the GTN forcing the data it generates to both be realistic and help a learner perform well on the meta-objective of classifying MNIST. The reason this approach should help prevent mode collapse is that if the generator only produces one class of images, a learner trained on that data will not be able to classify all classes of images. This algorithm (GTN-GAN) was able to produce realistic images with no identifiable mode collapse (Figure 18). GTNs offer a different type of solution to the issue of mode collapse than the many that have been proposed, adding a new tool to our toolbox for solving that problem. Note we do not claim this approach is better than other techniques to prevent mode collapse, only that it is an interesting new type of option, perhaps one that could be productively combined with other techniques.

J. Additional Motivation

There is an additional motivation for GTNs that involves long-term, ambitious research goals: GTN is a step towards algorithms that generate their own training environments, such that agents trained in them eventually solve tasks we otherwise do not know how to train agents to solve (Clune, 2019). It is important to pursue such algorithms because our capacity to conceive of effective training environments on our own as humans is limited, yet for our learning algorithms to achieve their full potential they will ultimately need to consume vast and complex curricula of learning challenges and data. Algorithms for generating curricula, such as the the paired open-ended trailblazer (POET) algorithm (Wang et al., 2019a), have proven effective for achieving behaviors that would otherwise be out of reach, but no algorithm yet can generate completely unconstrained training conditions. For example, POET searches for training environments within a highly restricted preconceived space of problems. GTNs are exciting because they can encode a rich set of possible environments with minimal assumptions, ranging from labeled data for supervised learning to (in theory) entire complex virtual RL domains (with their own learned internal physics). Because RNNs are Turing-complete (Siegelmann & Sontag, 1995), GTNs should be able to theoretically encode all possible learning environments. Of course, while what is theoretically possible is different from what is achievable in practice, GTNs give us an expressive environmental encoding to begin exploring what potential is unlocked when we can learn to generate sophisticated learning environments. The initial results presented here show that GTNs can be trained end-to-end with gradient descent through the entire learning process; such end-to-end learning has proven highly scalable before, and may similarly in the future enable learning expressive GTNs that encode complex learning environments.

K. On the realism of images

There are two phenomenon related to the recognizability of the GTN-generated images that are interesting. (1) Many of the images generated by GTNs are unrecognizable (e.g. as digits), yet a network trained on them still performs well on a real, target task (e.g. MNIST). (2) Some conditions increase the realism (recognizability) of the images. We will focus on the MNIST experiments because that is where we have conducted experiments to investigate this phenomenon.

Figure 20 shows all of the images generated by a GTN with a curriculum. Most of the images do not resemble real MNIST digits, and many are alien and unrecognizable. Interestingly, there is a qualitative change in the recognizability of the images at the very end of the curriculum (the last 4-5 rows, which show the last two training batches). Both phenomena are interesting, and we do not have satisfactory explanations for either. Here we present many hypothesis we have generated that could explain these phenomenon. We also present a few experiments we have done to shed light on these issues. A more detailed investigation is an interesting area for future research.

Importantly, the recognizable images at the end of the curriculum are not *required* to obtain high performance on MNIST. The evidence for that fact is in Figure 5, which shows that the performance of a learner trained on GTN-data is already high after around 23 inner-loop iterations, before the network has seen the recognizable images in the last 4-5 rows (which are shown in the last two training batches, i.e. training iterations 31 and 32). Thus, a network can learn to get over 98% accuracy on MNIST training only on unrecognizable images.

At a high level, there are three possible camps of explanation for these phenomenon.

Camp 1. Performance would be higher with *higher* realism, but optimization difficulties (e.g. vanishing/exploding gradients) prevent learning a generator that produces such higher-performing, more realistic images. Evidence in favor of this camp of hypotheses is that the realistic images come at the end of the curriculum, where the gradient flow is easiest (as gradients do not have to flow back through multiple inner-loop steps of learning). A prediction of this hypothesis is that as we improve our ability to train GTNs, the images will become more realistic.

Camp 2. Performance is higher with lower realism (at least when not late in the curriculum), which is why unrealistic images are produced. There are at least two reasons why unrealistic images could generate higher performance. (A) Compression enables *faster* learning (i.e. learning with fewer samples). Being able to produce unrealistic images allows much more information to be packed into a single training example. For example, imagine a single image that could teach a network about many different styles of the digit 7 all at the same time (and/or different translations, rotations, and scales of a 7). It is well known that data augmentation improves performance because it teaches a network, for example, that the same image at different locations in the image is of the same class. It is conceivable that a single image could do something similar by showing multiple 7s at different locations. (B) Unrealistic images allow better *generalization*. When trying to produce high performance with very few samples, the risk of performance loss due to overfitting is high. A small set of

realistic images may not have enough variation in non-essential aspects of the image (e.g. the background color) that allow a network to reliably learn the class of interest in a way that will generalize to instances of that class not in the training set (e.g. images of that class with a background color not in the training set). With the ability to produce unrealistic images (e.g. 7s against many different artificial backdrops, such as by adding seemingly random noise to the background color), GTNs could prevent the network from overfitting to spurious correlations in the training set (e.g. background color). In other words, GTNs could *learn* to produce something similar to domain randomization (Andrychowicz et al., 2018; Tobin et al., 2017) to improve generalization, an exciting prospect.

Camp 3. It makes no difference on performance whether the images are realistic, but there are more unrealistic images that are effective than realistic ones, explaining why they tend to be produced. This hypothesis is in line with the fact that deep neural networks are easily fooled (Nguyen et al., 2015) and susceptible to adversarial examples (Szegedy et al., 2013). The idea is that images that are unrecognizeable to us are surprisingly meaningful to (i.e. impactful on) DNNs. This hypothesis is also in line with the fact that images can be generated that hack a trained DNN to cause it to perform other functions it was not trained to do (e.g. to perform a different function entirely, such as hacking an ImageNet classification network to perfom a counting task like counting the number of occurences of Zebras in an image) (Elsayed et al., 2018). This hypothesis is also in line with recent research into meta-learning, showing that an initial weight vector can be carefully chosen such that it will produce a desired outcome (including implementing any learning algorithm) once subjected to data and SGD (Finn & Levine, 2017; Finn et al., 2017). One thing not explained by this hypothesis is why images at the end of the curriculum are more recognizable.

Within this third camp of hypotheses is the possibility that the key features required to recognize a type of image (e.g. a 7) could be broken up across images. For example, one image could teach a network about the bottom half of a 7 and another about the top half. Recognizing either on its own is evidence for a seven, and if across a batch or training dataset the network learned to associate both features with the class 7, there is no reason that both the top half and bottom half ever have to co-occur. That could lead to unrealistic images with partial features. One prediction of this hypothesis (although one not exclusive to this hypothesis), is that averaging all of the images for each class across the entire GTN-produced training set should reveal recognizable digits. The idea is that no individual image contains a full seven, but on average the images combine to produce sevens (and the other digits). Figure 19 shows the results of this experiment. On average the digits are recognizable. This result is also consistent with Camp 1 of hypotheses: perhaps performance would increase further if the images were individually more recognizable. It is also consistent with Camp 2: perhaps the network is forced to combine many sevens into each image, making them individually unrecognizeable, but recognizable as 7s on average. Additionally, in line with Camp 2, if the network has learned to produce something like domain randomization, it could add variation across the dataset in the background (making each individual image less recognizable), but Hypothesis 2 would predict that, on average, the aspects of the image that do not matter (e.g. the background) average out to a neutral value or the true dataset mean (for MNIST, black), whereas the true class information (e.g. the digit itself) would be recognizable on average, exactly as we see in Figure 19. Thus, the average images shed light on the overall subject, but do not provide conclusive results regarding which camp of hypotheses is correct.

An additional experiment we performed was to see if the alien images somehow represent the edges of the decision boundaries between images. The hypothesis is that images in the center of a cluster (e.g. a Platonic, archetypal 7) are not that helpful to establish neural network decision boundaries between classes, and thus GTN does not need to produce many of them. Instead, it might benefit by generating mostly edge cases to establish the decision boundaries, which is why the digits are mostly difficult to recognize. To rephrase this hypothesis in the language of support vector machines, the GTN could be mostly producing the support vectors of each class, instead of more recognizable images well inside of each class (i.e. instead of producing many Platonic images with a high margin from the decision boundary). A prediction of this hypothesis is that the unrecognizable GTN-generated images should be closer to the decision boundaries than the recognizable GTN-generated images.

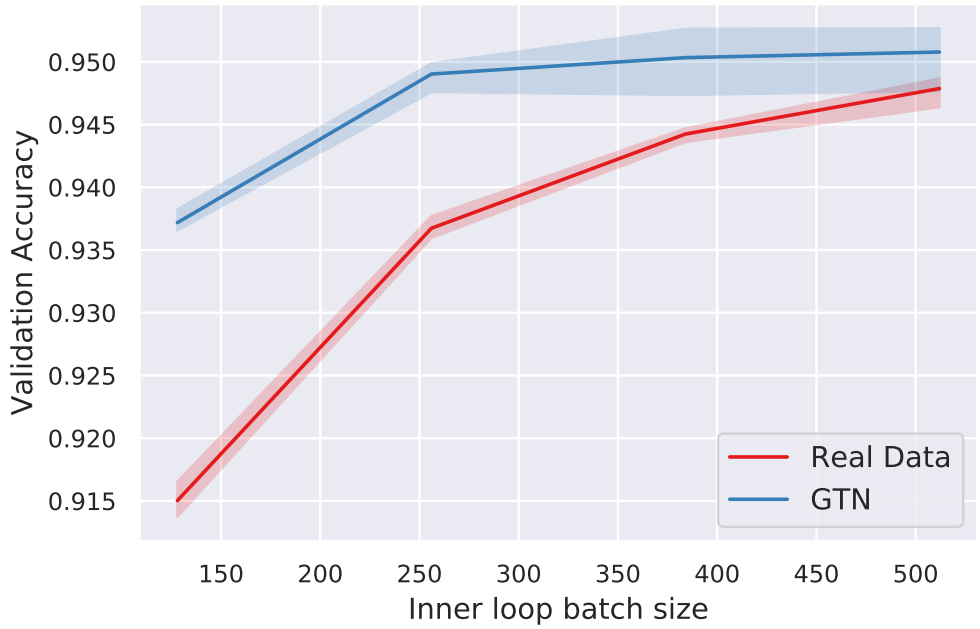
To test this hypothesis, we borrow an idea and technique from (Toneva et al., 2018), which argues that one way to identify images near (or far) from a decision boundary is to count the number of times that, during the training of a neural network, images in the training set have their classification labels change. The intuition is that Platonic images in the center of a class will not have their labels change often across training, whereas images near the boundaries between classes will change labels often as the decision boundaries are updated repeatedly during training. We trained a randomly initialized network on real images (the results are qualitatively the same if the network is trained on the GTN-produced images). After each training step we classify the images in Figure 20 with the network being trained. We then rank the synthetic images from Figure 20 on the frequency that their classification changed between adjacent SGD steps.

Figure 23 presents these images reordered (in row-major order) according to the number of times the output label for that image changed during training. The recognizable images are all tightly clustered in this analysis, showing that there is a strong relationship between how recognizable an image is and how often its label changes during training. Interestingly, the images are not all the way at one end of the spectrum. However, keep in mind that many images in this sorted list are tied with respect to the number of changes (with ties broken randomly), and the number of flips does not go up linearly with each row of the image. Figure 22 shows the number of label flips vs. the order in this ranked list. The recognizable images on average have 2.0 label flips (Figure 22, orange horizontal line), meaning that they are towards the extreme of images whose labels do not change often. This result is in line with the hypothesis that these are Platonic images well inside the class boundary. However, there are also many unrecognizable images whose labels do not flip often, which is not explained by this hypothesis. Overall, this analysis suggests the discovery of something interesting, although much future work needs to be done to probe this question further.

Why are images only realistic at the end of the curriculum? Separate from, but related to, the question of why most images are unrecognizable, is why the recognizable images are only produced at the end of the curriculum. We have come up with a few different hypotheses, but we do not know which is correct. (1) The gradients flow best to those samples, and thus they become the most realistic (in line with Camp 1 of hypotheses above). (2) It helps performance for some reason to have realistic images right at the end of training, but realism does not help (Camp 3) or even hurts (Camp 2) earlier in the curriculum. For example, perhaps the Platonic images are the least likely to change the decision boundaries, allowing them to be used for final fine-tuning of the decision boundaries (akin to an annealed learning rate). In line with this hypothesis is that, when optimization cannot create a deterministic curriculum, realism seems to be higher on average (Figure 21). (3) The effect is produced by the decision to take the batch normalization (Ioffe & Szegedy, 2015) statistics from the final batch of training. Batch normalization is a common technique to improve training. Following normal batch norm procedures, during inner-loop training the batch norm statistics (mean and variance) are computed per batch. However, during inner-loop testing/inference, the statistics are instead computed from the training set. In our experiments, we calculate these statistics from the *last* batch in the curriculum. Thus, if it helps performance on the meta-training test set (the inner loop test set performance the GTN is being optimized for) to have the statistics of that batch match the statistics of the target data set (which contains real images), there could be a pressure for those images to be more realistic. Contrary to this hypothesis, however, is the fact that realism increases in the last two batches of the curriculum, not just the last batch (most visible in Figure 6, which shows sample from each batch in a separate row).

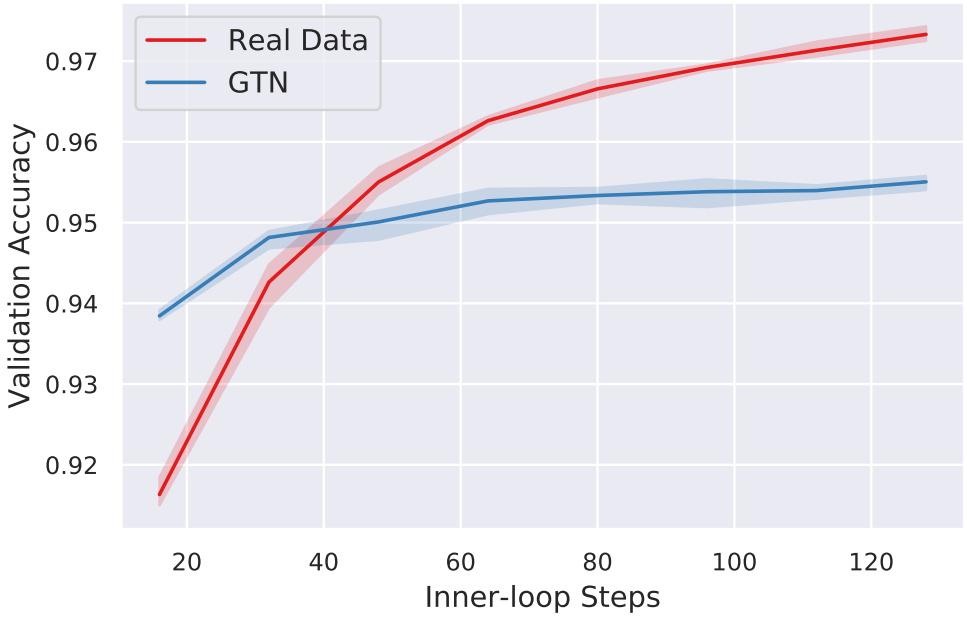
Another hypothesis (consistent with Camp 1 and Camp 3), is that producing first unrealistic then realistic images might reflect how neural networks learn (e.g. first learning low-level filters before moving to more complex examples). However, that hypothesis would presumably predict a gradual increase in realism across the curriculum, instead of realism only sharply increasing in the last few batches. Finally, we did not observe this phenomenon in the CIFAR experiments with a full curriculum: the last few batches are not realistic in that experiment (Figure 8). We do not know why the results on this front are different between MNIST and CIFAR experiments.

In short, we do not have a good understanding for why realism increases towards the end of the curriculum. Shedding more light on this issue is an interesting area for future research.



0.5

Figure 11. Increasing inner-loop batch size



0.5

Figure 12. Increasing inner-loop optimization steps

Figure 13. (a) The left figure shows that even though GTN was meta-trained to generate synthetic data of batch size 128, sampling increasingly larger batches results in improved learner performance (the inner-loop optimization steps are fixed to 16). (b) The right figure shows that increasing the number of inner-loop optimization steps (beyond the 16 steps used during meta-training) improves learner performance. The performance gain with real data is larger in this setting. This improvement shows that GTNs do not overfit to a specific number of inner-loop optimization steps.

Figure 14. GTN samples w/o curriculum.

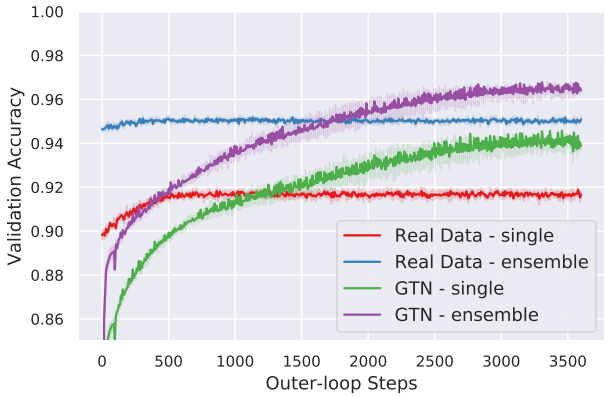


Figure 15. Performance of an ensemble of GTN learners vs. individual GTN learners. Ensembling a set of neural networks that each had different weight initializations, but were trained on data from the *same GTN* substantially improves performance. This result provides more evidence that GTNs generate a healthy distribution of training data and are not somehow forcing the learners to all learn a functionally equivalent solution.

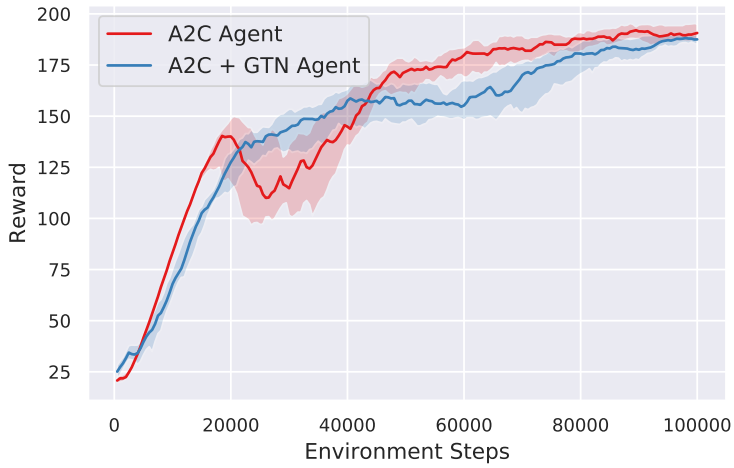


Figure 16. An A2C Agent control trains a single policy throughout all of training, while the GTN method starts with a new, randomly initialized network *at each iteration* and produces the plotted performance *after a single step of SGD*. This plot is difficult to parse because of that difference: it compares the accumulated performance of A2C across all environment steps up to that point vs. the performance achieved with GTN data *in a single step of SGD from a single batch of synthetic data*. Thus, at the 100,000th step of training, GTNs enable training a newly initialized network to the given performance (of around 190) 100,000 times faster with GTN synthetic data than with A2C from scratch. With GTNs, we can therefore train many new, high-performing agents quickly. That would be useful in many ways, such as greatly accelerating architecture search algorithms for RL. Of course, these results are on a simple problem, and (unlike our supervised learning experiments) have not yet shown that the GTN data works with different architectures, but these results demonstrate the intriguing potential of GTNs for RL. One reason we might expect even larger speedups for RL vs. supervised learning is because a major reason RL is sample inefficient is because it requires exploration to figure out how to solve the problem. However, once that exploration has been done, the GTN can produce data to efficiently teach that solution to a new architecture. RL thus represents an exciting area of future research for GTNs. Performing that research is beyond the scope of this paper, but we highlight the intriguing potential here to inspire such future work.

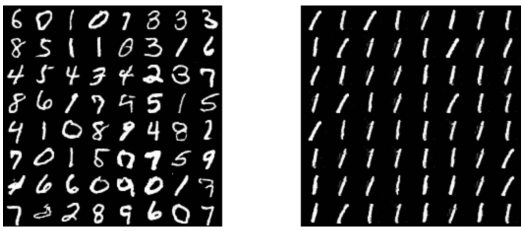


Figure 17. **Images generated by a basic GAN on MNIST before and after mode collapse.** The left image shows GAN-produced images early in GAN training and the right image shows GAN samples later in training after mode collapse has occurred due to training instabilities.

Figure 18. **Images generated by a GTN with an auxiliary GAN loss.** Combining GTNs with GANs produces far more realistic images than GTNs alone (which produced alien, unrecognizable images, Figure 14). The combination also stabilizes GAN training, preventing mode collapse.

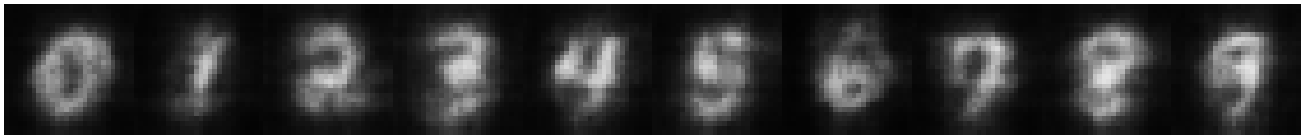


Figure 19. Pixel-wise mean per class of all GTN-generated images from the full curriculum treatment.

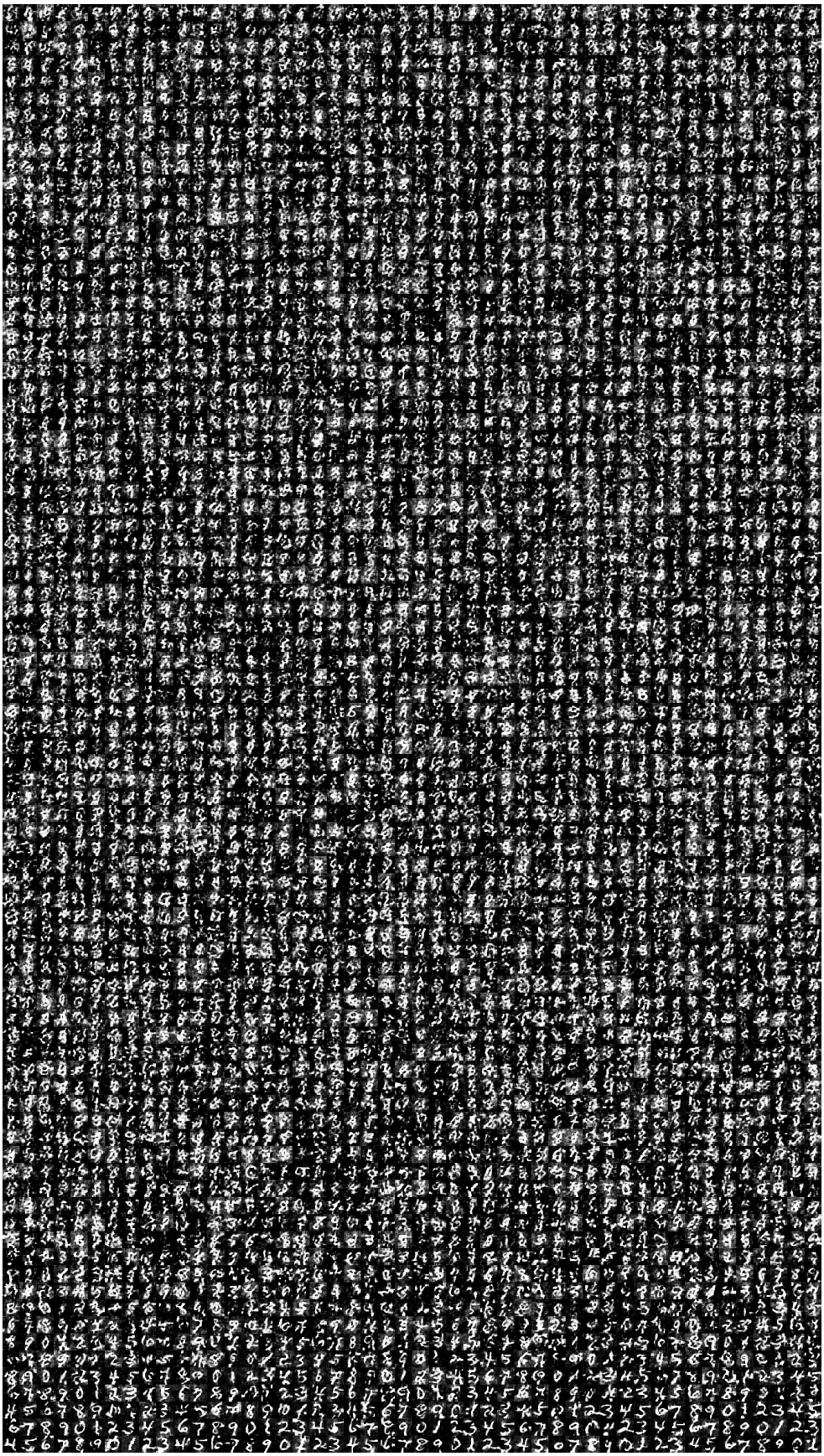


Figure 20. All images generated by the full-curriculum GTN. The images are shown in the order they are presented to the network, with the first batch of images in the curriculum in the top row and the last batch of data in the last row. The batch size does not correspond to the number of samples per row, so batches wrap from the right side of one row to the left side of the row below.

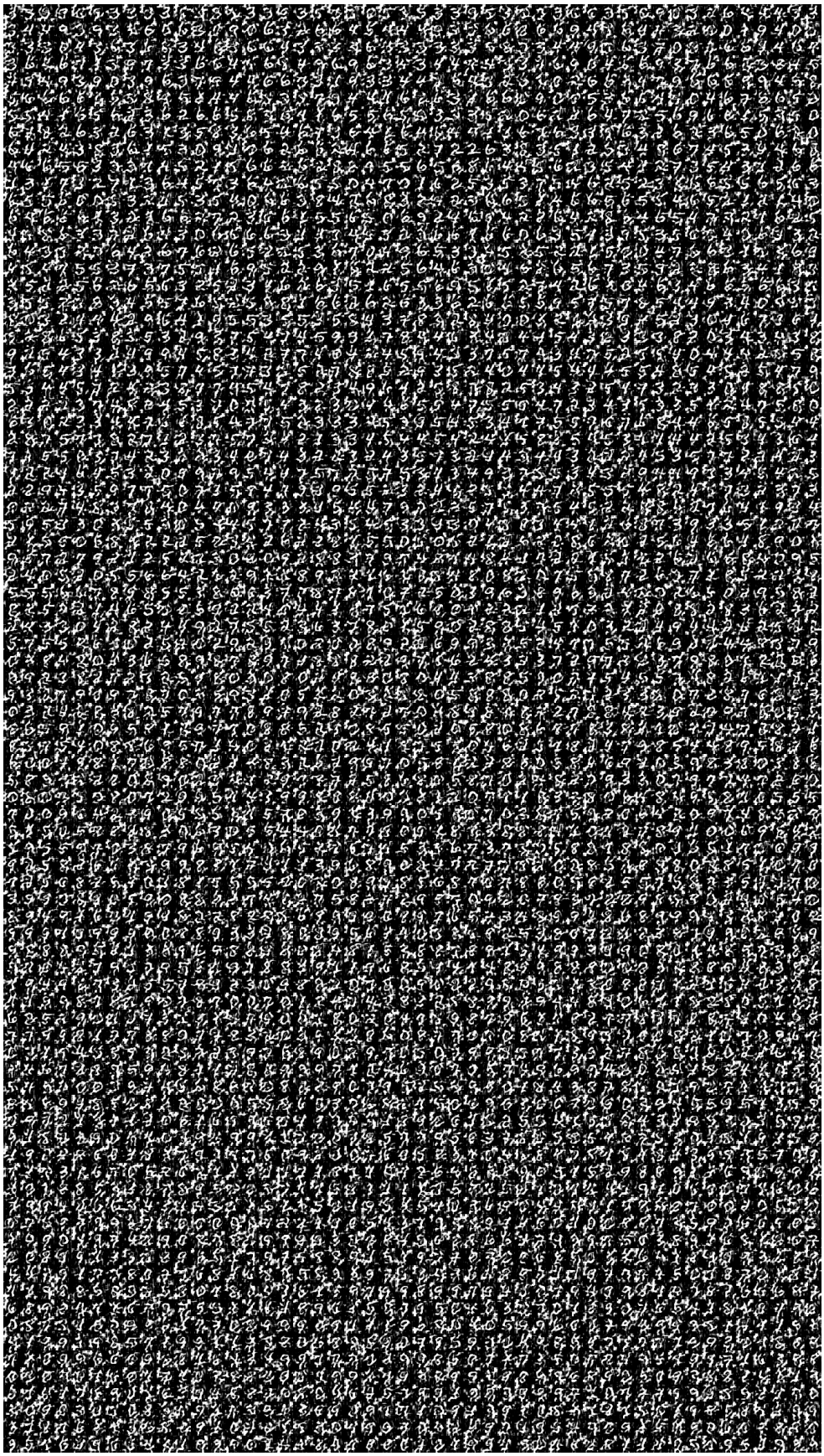


Figure 21. A sample of images generated by the no-curriculum GTN.

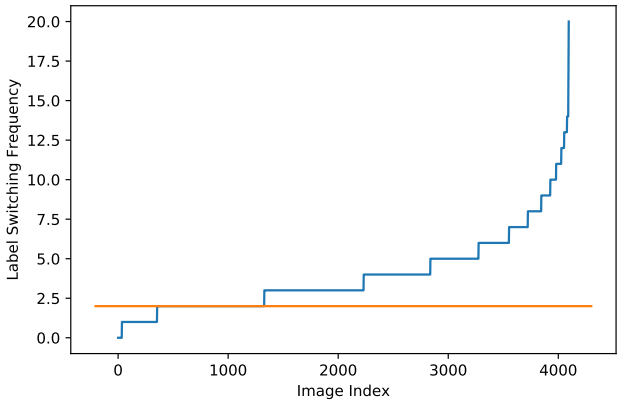


Figure 22. The number of times a class label changes across training for each GTN-generated sample (Y-axis) vs. the rank of that sample when ordered by that same statistic (X-axis). A relatively small fraction of the samples flip labels many times (in line with the idea that they are near the class decision boundaries), whereas most samples change labels only a few times (i.e. once the are learned, they stay learned, in line with them being more canonical examples). The orange line shows the average number of class changes for recognizable images (those in the red box in Figure 23). While not the images with the least number of flips, these recognizable images are towards the end of the spectrum of images whose labels do not change often, in line with the hypothesis that they are more canonical class exemplars.

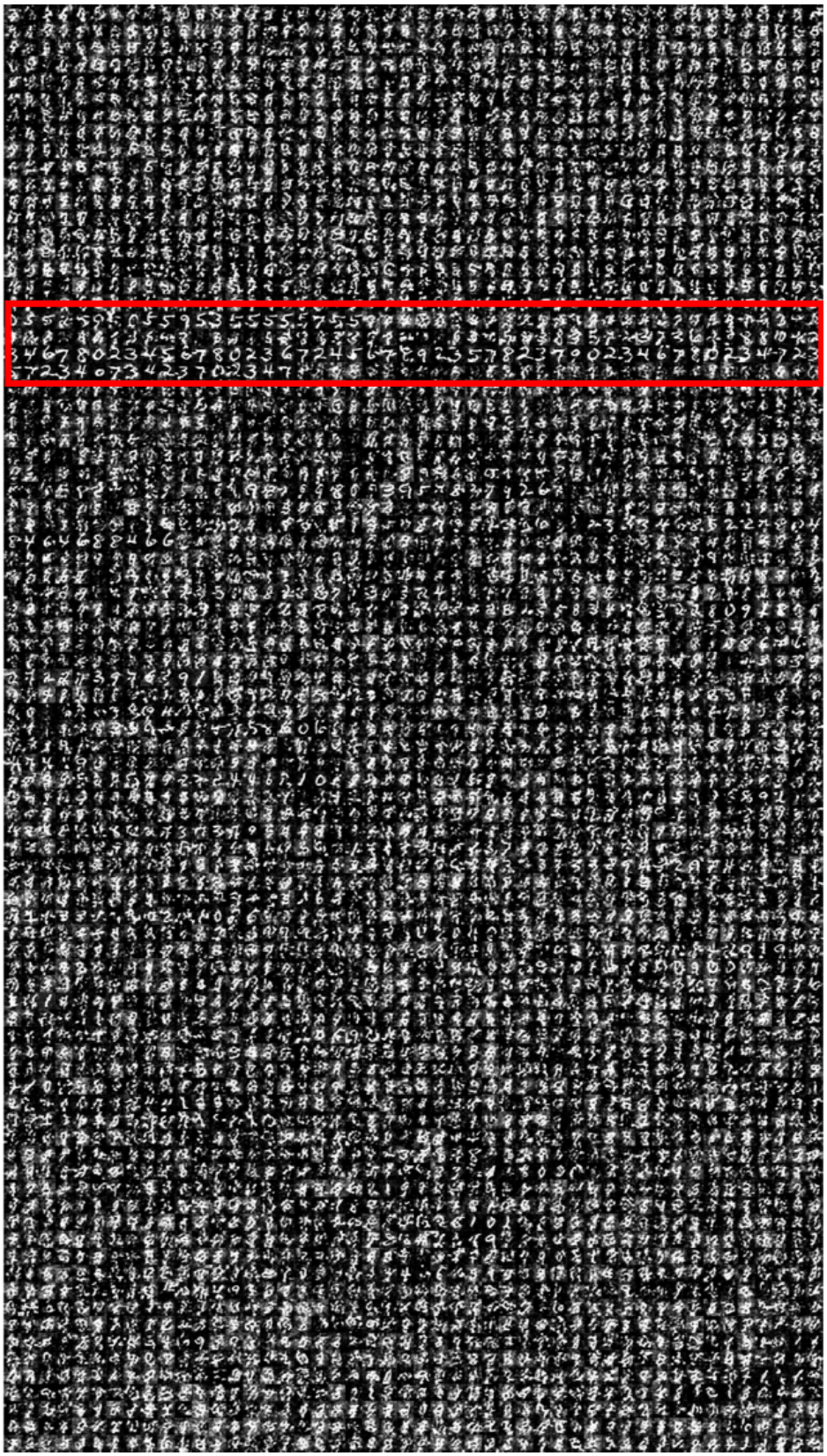


Figure 23. All images generated by the full-curriculum GTN ordered by the frequency that their labels change during training. Highlighted is a dense region of realistic images that we manually identified.