GENERATING INFORMATIVE SAMPLES USING GENERATIVE ADVERSARIAL NETWORKS

Anirudha Jitani, Deepak Sharma & Amit Sinha
School of Computer Science, Electrical and Computer Engineering
McGill University
{anirudha.jitani,deepak.sharma,amit.sinha}@mail.mcgill.ca

ABSTRACT

We explore the use of active learning in image classification task, where the images are drawn both from the real dataset and synthetic dataset. We evaluate different acquisition functions by training a CNN that uses them on three different training sets - only real images, only synthetic images, both real and synthetic images. We use MC dropout to get the model prediction values for multiple stochastic runs, which we then use to score informative samples using the activation functions. We observe that active learning can be helpful in achieving higher accuracy with lower training samples as compared to traditional training methods in two cases and provide an explanation as to why it doesn't work in the case of synthetic data. In our next step, we plan to incorporate active learning in the generation process of the synthetic images, which could further enhance our classifier's performance. The code for our experimental setup is available on github[1].

1 INTRODUCTION

Active learning is a technique used to train machine learning models when there is a scarcity of labelled data compared to unlabelled data. Model uncertainty measures are used to identify the most useful data points for training a model and a trusted annotator named as 'oracle' labels those training instances. Gal et al. (2017) shows that active learning can be more efficient in training convolutional neural networks in terms of training data points needed as compared to picking data points randomly from the learning pool. As an initial step, we compare different pool-based acquisition functions on original and synthetic versions of the standard MNNIST LeCun (1998) dataset, and prepare a synthetic CIFAR-10 Krizhevsky et al. dataset for future work.

Our motivation for this project is to explore the possibility of training of deep Bayesian CNNs (active learner) faster using synthetic training data generated by an Generative Adversarial Networks (GAN). As an intial step, we seek to demonstrate that the active learning scenario is useful in accelerating learning of deep CNNs when trained and tested on the original MNIST dataset. Subsequently, we conduct the same experiments by training on a mixture of fake and real MNIST data, as well as on purely fake data. Finally, our intention is to modify the cost function of the generator of a GAN to produce fake images that can accelerate the CNN training even further. We aim to measure the information of the generated images indirectly by tracking the performance of a deep bayesian CNN on the MNIST/CIFAR-10 test datasets against the ratio of fake against real data used for training. If the quality of the fake images is better than real images, the active learner should pick fake images over real ones and the test performance should improve at least as fast as the normal scenario.

The applications of active learning in GANs could be beneficial in medical imaging where there is a shortage of labelled data. The ability to generate informative synthetic samples would allow models to train on their own, with minimal manual labelling effort.

2 RELATED WORK

Active learning has been an active area of research for more than a decade now. Settles (2012) provides a good literature review of active learning scenarios, their advantages and disadvantages, the

[1]https://github.com/deepaks4077/comp652-activelearning

computational costs of different query strategies as well as suggestions for different contexts. Gal & Ghahramani (2016) proposes a simple method, named MC dropout, which involves adding a single dropout layer before each weighted layer to derive model uncertainty values with a predictive mean and variance. Gal et al. (2017) combines the same Bayesian approximation with active learning techniques to demonstrate improved performance in medical image classification tasks over a smaller set of samples as compared to a learner that picks training samples randomly. The acquisition functions BALD and Mean Standard Deviation were borrowed from this paper.

Generative Adversarial Networks GANs Goodfellow et al. (2014) are implemented by a system of two neural networks competing with each other in a zero-sum game. The DC-GAN architecture Radford et al. (2015) proposed some modifications on the vanilla GAN to make it work well with large images. We use the architecture as described in that paper. Studies of GANs under the active learning setting in the image domain have been sparse. Zhu & Bento (2017) uses adversarial training procedure where the classifier is iteratively trained on the adversarial example generated by the GAN decoder. The decoder samples images using latent variables that are close to the decision boundary in the hyper-plane induced by the active learner. Mayer & Timofte (2018) proposes a pool-based adversarial active learning technique for multi-label classification using deep convolutional classifiers using uncertainty sampling, adversarial sample generation and sample matching. Mehrjou et al. (2018) proposes a query criterion for active learning that uses Variational Autoencoders to estimate the distribution of the data, resulting in a learner that is more robust against outliers.

We hypothesize that by modifying the value function of our GAN to incorporate the density distribution of our actual dataset and the uncertainty estimates of our active learner, we can train the active learner faster.

3 Active Learning

Active learning systems attempt to overcome the labeling bottleneck by asking queries in the form of unlabeled instances to be labeled by a trusted annotator called the oracle. Active learning is specially useful in applications where the data is present in abundance, but obtaining the labels for such datasets is expensive. A good active learner achieves high test performance using a significantly less number of labelled samples for training.

There are three different settings of active learning used widely - pool-based, stream-based, and query synthesis methods. In case of pool-based active learning, all the data is available and the learner queries the data points that would result in the largest increase in test performance. In case of stream-based methods, the data points are coming one by one in a stream and the active learning algorithm needs to decide if it wants to query the oracle for the incoming instance. In the query-synthesis method, the learner constructs an instance for the oracle to label.

3.1 Acquisition Functions

Active learners employ acquisition functions to select training samples. We compare the following acquisition functions.

3.1.1 Least Confident

The Least Confidence (ϕ^{LC}) acquisition function chooses the samples with the most uncertainty (this is also referred to as Variation Ratios in Gal et al. (2017)):

$$\phi^{LC}(\mathbf{x}) = 1 - \max_{y \in Y} P(y|\mathbf{x})$$

3.1.2 Least Margin

The Least Margin (ϕ^{M}) acquisition function chooses samples which have the largest difference between the highest and second highest class prediction:

$$\phi^{M}(\mathbf{x}) = -(\max_{y \in C} P(y|\mathbf{x}) - \max_{\substack{y \in Y \\ y \notin y_{max}}} P(y|\mathbf{x}))$$

3.1.3 TOKEN ENTROPY

Token Entropy (ϕ^{TE}) finds the entropy among the classes:

$$\phi^{TE}(\mathbf{x}) = -\sum_{y_i \in Y} P(y_i|\mathbf{x}) \log(P(y_i|\mathbf{x}))$$

3.1.4 DENSITY MAX UNCERTAINTY

Density Max Uncertainty (ϕ^{DMU}) uses the same scoring function as max along with a function that estimates outlier uncertainty (based on cosine similarity of data)

$$\phi^{DMU}(\mathbf{x}) = \frac{\sum_{s_i \in S(\mathbf{x})} \cos(\mathbf{x}, s_i)}{Num(S(\mathbf{x}))} \times \phi^{LC}(\mathbf{x})$$

3.1.5 SIMILARLY DENSITY ENTROPY

Similarly Density Entropy (ϕ^{DE}) uses the same function as Token Entropy with the outlier scoring function:

$$\phi^{DE}(\mathbf{x}) = \frac{\sum_{s_i \in S(\mathbf{x})} \cos(\mathbf{x}, s_i)}{Num(S(\mathbf{x}))} \times \phi^{TE}(\mathbf{x})$$

3.1.6 BALD

We also use the BALD (ϕ^{BALD}) acquisition function to pick samples which maximize the mutual information as done in Gal et al. (2017):

$$\phi^{BALD}(\mathbf{x}) = -\sum_{y_i \in Y} P(y_i|\mathbf{x}) \log(P(y_i|\mathbf{x})) + \mathbb{E}_{p(\theta)}(\sum_{y_i \in Y} P(y_i|\mathbf{x}, \theta) \log(P(y_i|\mathbf{x}, \theta)))$$

3.1.7 MEAN STANDARD DEVIATION (STD)

The Mean STD (ϕ^{STD}) acquisition function picks the samples which show the highest average (over classes) standard deviation:

$$\phi^{STD} = \frac{1}{C} \sum_{y_i \in Y} \sqrt{\mathbb{E}_{p(\theta)}[P(y_i|\mathbf{x}, \theta)^2] - \mathbb{E}_{p(\theta)}[P(y_i|\mathbf{x}, \theta)]^2}$$

4 DEEP CONVOLUTIONAL GENERATIVE ADVERSARIAL NETWORKS (DC-GAN)

Generative Adversarial Networks GANs Radford et al. (2015) are implemented by a system of two neural networks competing with each other in a zero-sum game. The two neural networks are namely Generator (G) and Discriminator (D), where in the generator uses a noise vector z to create an image x = G(z) and the role of the discriminator is to identify whether the image generated by G is real or fake. At equilibrium, the generator is assumed to recover the true distribution that generates the data. GANs suffer from major problems such as non-convergence, finding equilibrium in high-dimension non-convex spaces, mode collapse, diminishing gradients, high sensitivity to hyper-parameters which makes it very unstable to train.

The authors of DC-GAN Radford et al. (2015) came up a few modifications on the vanilla GAN to improve its stability and make it work better with images : a) Use strided convolution instead of spatial pooling functions, allowing the network to learn its own spatial down-sampling, b) Eliminating fully connected layers on top of convolutional features, c) Batch Normalization to stabilize learning. However, to avoid model instability, batchnorm was not applied to generator output layer and discriminator input layer. We use the same architecture to generate synthetic MNIST and CIFAR datasets. The generator is a fractionally-strided convolution network whereas the discriminator is a convolutional neural network. Fig. 1 shows the architecture of the generator used in our experiments.

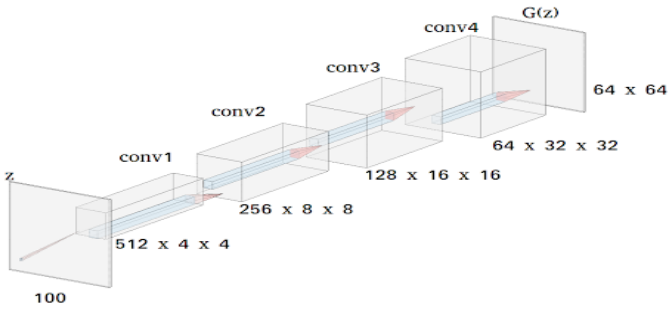

Figure 1: DCGAN Generator Radford et al. (2015) used for CIFAR-10 data generation. A 100 dimensional uniform distribution Z is projected to a small spatial extent convolutional representation with many feature maps. A series of four fractionally-strided convolutions then convert this high level representation into a 64 * 64 pixel image.

5 Experiment Setup

The basic experimental setup is displayed in the flow chart in Fig. 2. Multiple stochastic runs are carried out on the model (in our work we use N=10 forward stochastic runs for each example). The learner scans a sixth of the entire training dataset and selects 250 data points (without replacement) to train on. The predictive mean is given by the Monte Carlo estimate which is:

$$P(y|\mathbf{x}) = \int P(y|\mathbf{x};\theta)P(\theta)d\theta \approx \frac{1}{N}\sum_{n=1}^{N} P(y|\mathbf{x};\hat{\theta}_t)$$

Here $\hat{\theta}_t$ is a weights sample drawn from Bayesian CNN weight distribution (For MC Dropout Gal et al. (2017), it is equivalent to using dropout in the same way in test time as in train time). We use MC dropout (dropout probability = 0.2) to estimate the predictive mean and variance of our model for each data point. This procedure is repeated until the learner has trained on a total of 5000 data points. After each run of the acquisition function, the model is tested over all of the 10000 images in the MNIST test dataset.

5.1 Pool-based Active Learning

In the first part of this project, we use active learning to find the samples that our model is most uncertain about and train our model on those samples. We evaluate the performance of different acquisition functions against a random sampler (which chooses the same number of samples randomly from the dataset) on MNIST. Our CNN has the following architecture:

Input(-1, 1, 28 x 28) $\Rightarrow$ Layer 1[Conv2d(5x5, 16) - BatchNorm2d - ReLu - Dropout(0.2) - MaxPool2d(2x2)] $\Rightarrow$ Layer 2[Conv2d(5x5, 32) - BatchNorm2d - ReLu - Dropout(0.2) - MaxPool2d(2x2)] $\Rightarrow$ Fully Connected Layer[Dropout(0.2) - Dense(10)] $\Rightarrow$ Output[SoftMax(10)]

5.2 Generating Synthetic Images

We train a DC-GAN on MNIST and CIFAR-10 to generate synthetic images. The images need to labelled accurately, and manual labelling for such a huge dataset is not feasible. We considered four options - a) Training a classifier on the original dataset and using the classifier to label the synthetic dataset, b) Using K-Means Clustering to cluster the images into 10 different clusters and manually labelling the 10 classes, c) Training the DC-GAN for each category of the image separately and generating the synthetic images, d) Conditioning the training process of the DC-GAN on the image category (conditional GAN).

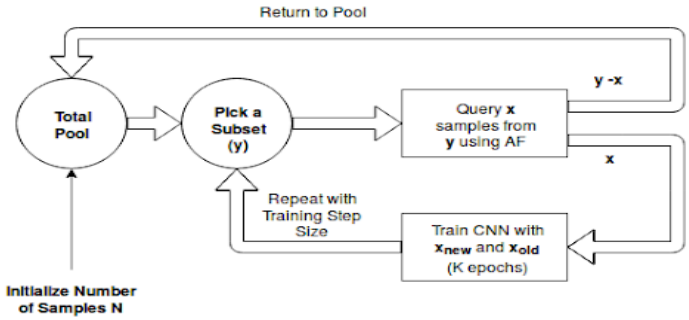

Figure 2: Flow Diagram of the Experiment Setup

Even though the first two methods are probably faster, we cannot be certain if the labels allocated to these images are going to be correct. Due to stability issues faced while training a conditional GAN, we opted for the third option and trained the DC-GAN separately on images of a single class.

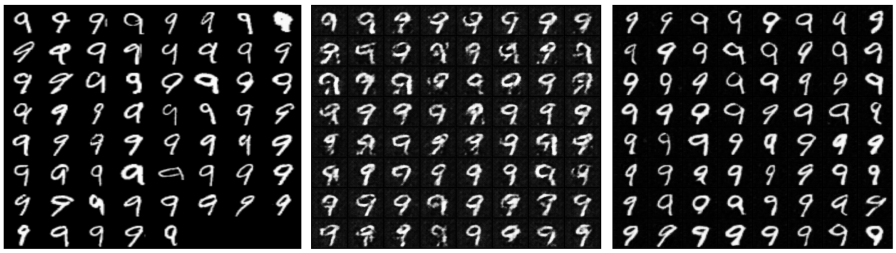

Figure 3: The figure on the (left) shows the actual MNIST data, the image generated by the GAN after training for 5 epochs (center) and 25 epochs (right)

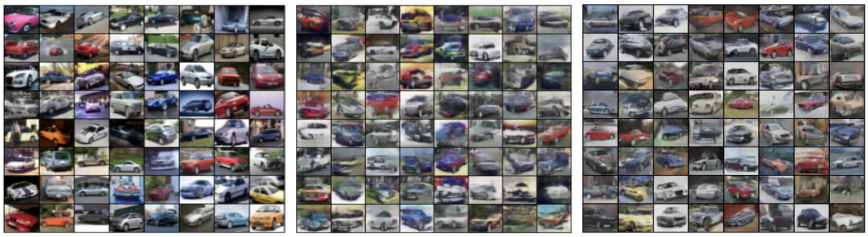

Figure 4: The figure on the (left) shows real CIFAR data for category automobile, the image generated by the GAN after training for 50 epochs (center) and 500 epochs (right)

5.3 Pool-based Active Learning with Real and Synthetic Images

As described above, we used the DC-GAN to obtain 60000 generated MNIST images (6000 for each digit). We synthesize a dataset using half of the generated and real MNIST data. We run the same experiment as described at the start of Section 6. In addition, we track the percentage of synthetic samples selected by our active learner.

6 RESULTS

The hyper-parameters used in all the experiments are the learning rate of the CNN, the sampling size (which is the size of the subset of data from which we want to select a few points based on the acquisition function - this value is ideally the size of the entire train dataset, but it was slightly reduced for faster experiments), the selection size (how many samples are chosen based on the acquisition function scores), the maximum number of training samples we want to train on, the number of epochs used in re-training the model after every selection step takes place, and whether we reset the model and train from scratch after every selection step.

All the results shown below plot test accuracy vs. selection steps, using the model that is trained at each step against the held-out test dataset (10000 in MNIST). Note that at each step we had 250 new samples in this experiment and we take 20 such steps. The common settings in all experiments are: learning rate (0.001), sampling size (10000 - note that this is out of 60000 for MNIST), selection size (250) and maximum training samples (5000). We do not reset the model after every selection (reset mode = False). The number of epochs for each experiment differs and is mentioned in the corresponding plots. The code is implemented in PyTorch (Version '0.4.1.post2') with Cuda Version 9.1 on a system with 8GB RAM, Processor - i7-7700HQ CPU @ 2.80GHz 8, GPU - GeForce GTX 1050 Ti/PCIe/SSE2 (4GB).

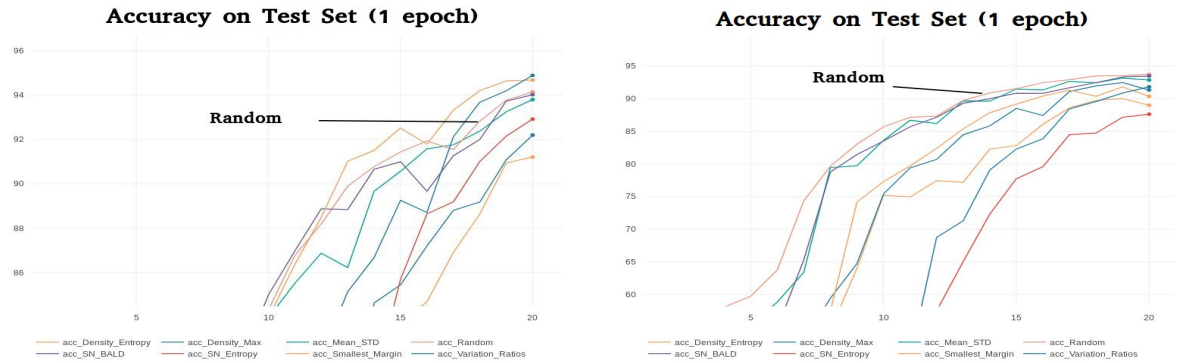

Figure 5: Test Accuracy on Real Data Alone (left) and Synthetic Data Alone (right) vs. Selection Step (trained on 1 epoch)

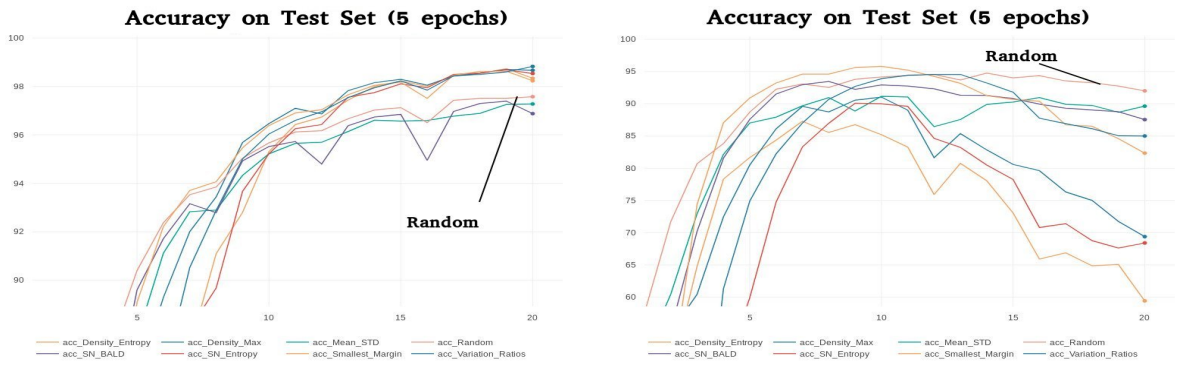

Figure 6: Test Accuracy on Real Data Alone (left) and Synthetic Data Alone (right) vs. Selection Step (trained on 5 epoch)

7 DISCUSSIONS

The results of the experiments for various epochs can be seen in Fig. 5, Fig. 6 and Fig. 7 for the cases where only real data or only synthetic data are used. For the real data, we can see that as the number

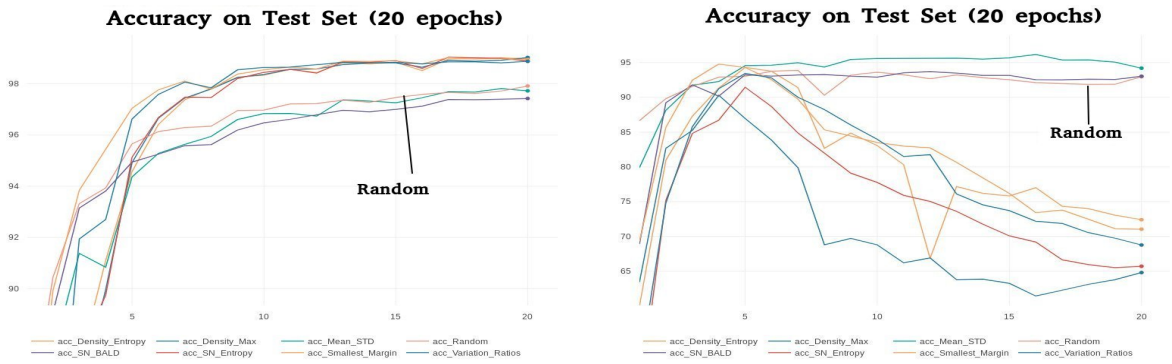

Figure 7: Test Accuracy on Real Data Alone (left) and Synthetic Data Alone (right) vs. Selection Step (trained on 20 epoch)

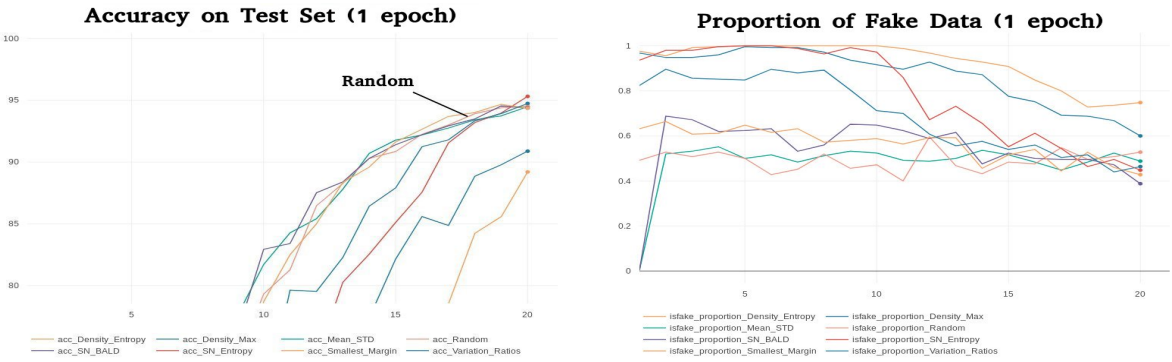

Figure 8: Real and Synthetic Data (1 epoch) Test Accuracy (left) and Proportion of Synthetic Data (right) vs. Selection Step

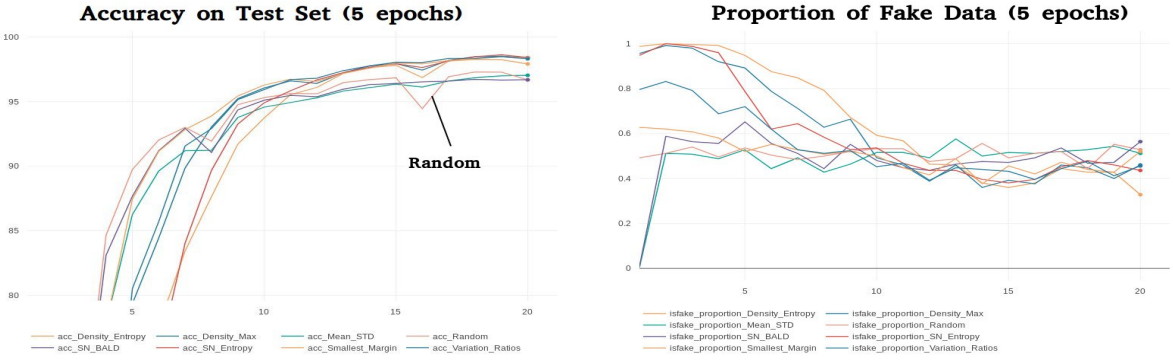

Figure 9: Real and Synthetic Data (5 epoch) Test Accuracy (left) and Proportion of Synthetic Data (right) vs. Selection Step

of epochs increases, the separation between the random and other acquisition functions increases. The other functions get better than random as larger epochs are used. We attribute this effect to the fact that additional training will allow the acquisition functions to give more reliable uncertainty estimates. Since models trained with higher epochs generally have much better performance, the uncertainty estimates from such models will have a much more reliable uncertainty estimate. For the case of the synthetic data, training additional epochs actually drops the performance of the acquisition function and increases the separation between the random and acquisition functions. We believe this is happening because the acquisition functions are operating on the entire synthetic

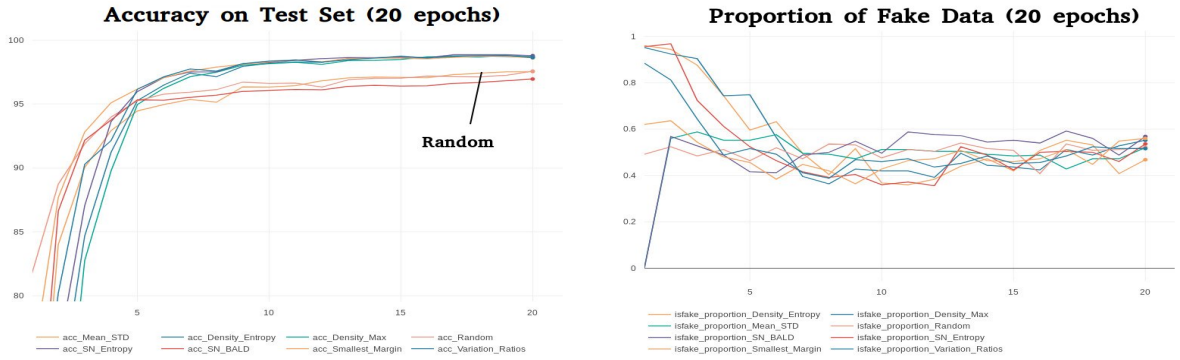

Figure 10: Real and Synthetic Data (20 epoch) Test Accuracy (left) and Proportion of Synthetic Data (right) vs. Selection Step

training dataset and they give us the most informative samples in context to the synthetic dataset. In can be viewed as over-fitting over the synthetic dataset, so that its performance drops on the generalized real test dataset. In this sense, using random acquisition acts as a form of regularization over the synthetic dataset and so it does not over-fit even if the training size and number of epochs is increased.

The experiments with real and fixed data mixed together can be seen in Fig. 8, Fig. 9 and Fig. 10. With the previous discussion in mind, we find that the learning curves of the real data rise much faster than the learning curves of the mixed data (although they look similar, this can be easily seen by comparing the y-axis scales between the 2 experiments). This makes sense, since we should reasonably expect to see a kind of hybrid between the results on real and synthetic data. What is particularly interesting to see in this case is the nature of the synthetic proportion graphs as the number of epochs increases. As the selection steps increase (x-axis, which is same as that of the accuracy graph), the proportion graphs go to the half proportion point (0.5) faster in the case when higher epochs are used in training. This means that the acquisition functions are preferring to take equally from both the real and synthetic data. It is still unclear why it tends to settle around the 0.5 mark (reasonably it could settle at any value on the y-axis), therefore further investigation is required.

Thus it is clear that active learning does help considerably, we can see in the graphs (trained with higher epochs) that better performance is achieved with a much smaller number of training samples. We provided a justification as to why this does not happen for the case of synthetic data alone, and further analysis is required to tackle this problem.

8 Future Work

In our next step, we plan to tweak the generators cost function in order to generate synthetic samples which are going to be the most efficient for training the model. In this setting, we plan to use the uncertainty estimates to identify latent variable values that produce samples that are the most informative. Further experiments and research are still required to identify the best approach. We also plan to perform the same experiments on CIFAR-10 (we already have the synthetic images generated for this dataset) and ISIC 2016 melanoma diagnosis dataset to verify if the observations can be generalized across different datasets.

9 ACKNOWLEDGMENT

We would like to sincerely thank our instructors Mr. Riashat Islam (McGill University, MILA) and Dr. Audrey Durand (McGill University, MILA) for guiding us throughout the project from the formulation of the idea to its final delivery.

REFERENCES

Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pp. 1050–1059, 2016.

Yarin Gal, Riashat Islam, and Zoubin Ghahramani. Deep bayesian active learning with image data. *arXiv preprint arXiv:1703.02910*, 2017.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pp. 2672–2680, 2014.

Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 (canadian institute for advanced research). URL `http://www.cs.toronto.edu/~kriz/cifar.html`.

Yann LeCun. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.

Christoph Mayer and Radu Timofte. Adversarial sampling for active learning. *arXiv preprint arXiv:1808.06671*, 2018.

Arash Mehrjou, Mehran Khodabandeh, and Greg Mori. Distribution aware active learning. *arXiv preprint arXiv:1805.08916*, 2018.

Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

Burr Settles. Active learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 6 (1):1–114, 2012.

Jia-Jie Zhu and José Bento. Generative adversarial active learning. *arXiv preprint arXiv:1702.07956*, 2017.