$E_{\text{in-style}}$: Clothing Item Generation using GANs

Charlie Brayton (014559415)
Department of Software Engineering
San José State University
San José, California
charles.brayton@sjsu.edu

Mohit Patel (014501461)
Department of Software Engineering
San José State University
San José, California
mohit.patel@sjsu.edu

Andrew Selvia (014547273)
Department of Software Engineering
San José State University
San José, California
andrew.selvia@sjsu.edu

Dylan Zhang (013073437)
Department of Software Engineering
San José State University
San José, California
dylan.zhang@sjsu.edu

***Abstract*—The primary objective of this research is to classify and generate images of clothing items. It is a compilation of three distinct efforts: (1) to classify fashion-mnist [1] images and, inversely, add adversarial noise to test the classifier; (2) to generate novel images which emulate the true fashion-mnist images; (3) to infuse the grayscale images with life-like colors. Together, these endeavours pushed the team to explore topics at the forefront of modern machine learning, especially generative adversarial networks (GANs).**

***Index Terms*—machine learning, computer vision, neural networks, generative adversarial networks**

I. Introduction

The diverse branches of this project were made possible by the careful selection of a dataset. The fashion-mnist dataset was chosen specifically for its prevalence as a benchmark in recent research into neural networks. Its size, labels, and compatibility (with the long-established MNIST dataset) make it an ideal choice for evaluating deep neural networks and GANs. Rather than attempt to introduce a truly groundbreaking architecture given the authors' limited experience, this project aims for breadth.

Each subproject explores a unique research area with industrial applications which may answer the associated questions:

1) Classification: Can we quickly identify an item of clothing to automate sorting and retrieval? Supposing such an automated system existed, how susceptible might it be to failure?
2) Generation: Can the product development lifecycle for clothing be accelerated by avoiding physical prototyping?
3) Colorization: Can grayscale clothing imagery be augmented with colors to enable A/B testing of styles to increase customer satisfaction?

The implementations of the separate components are hosted on GitHub [2], [3].

II. Background

The papers which fuel this project center around the various GAN architectures that have been introduced in the past 6 years. subsection IV-B lists high-level synopses of the ten papers used in the generation component of the project. The other components describe their background in prose.

III. Data

The fashion-mnist dataset [1] is composed of 60,000 training images and 10,000 testing images. In keeping with the legacy of the traditional MNIST, fashion-mnist images are 28x28 grayscale pixels. Each image is labeled as one of the ten classes pictured in Figure 1 and enumerated below:

0) T-shirt/top
1) Trouser
2) Pullover
3) Dress
4) Coat
5) Sandal
6) Shirt
7) Sneaker
8) Bag
9) Ankle Boot

IV. Approach

A. Classification

Figure 2 demonstrates the classification task. In this example, the top row indicates the classifier has successfully classified the image into the 9^{th} class (Ankle Boot). However, the bottom image has been misclassified into the 5^{th} class (Sandal) rather than the 7^{th} class (Sneaker).

The classification task was able to leverage an existing implementation provided in the official TensorFlow documentation of Keras [4]. Training was performed using two neural network architectures; one used only dense layers and another used convolutional layers. The first architecture, shown in Figure 3, flattens the 28x28 pixel image into a 784-pixel vector which is then densely connected to a 128-node layer; finally, the 128-node layer is densely connected to a 10-node output layer.

The second architecture, shown in Figure 4, scans the image with a convolutional neural network using a 5x5 window and

Fig. 1: fashion-mnist Samples

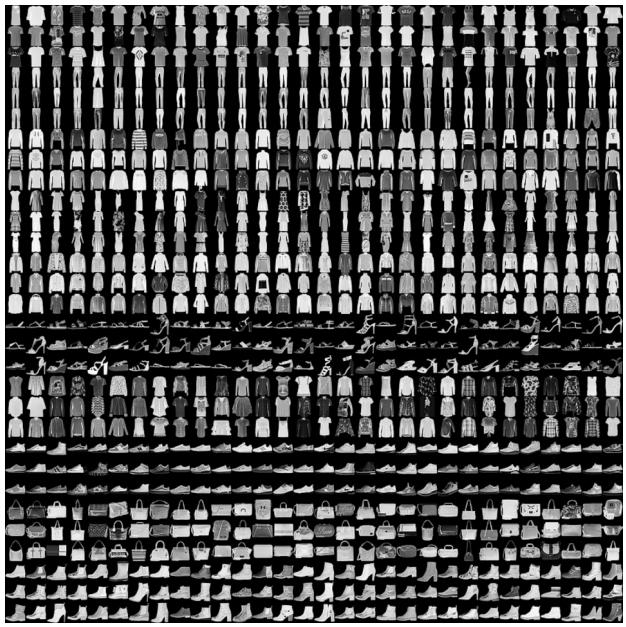


Fig. 2: Classified fashion-mnist Testing Images

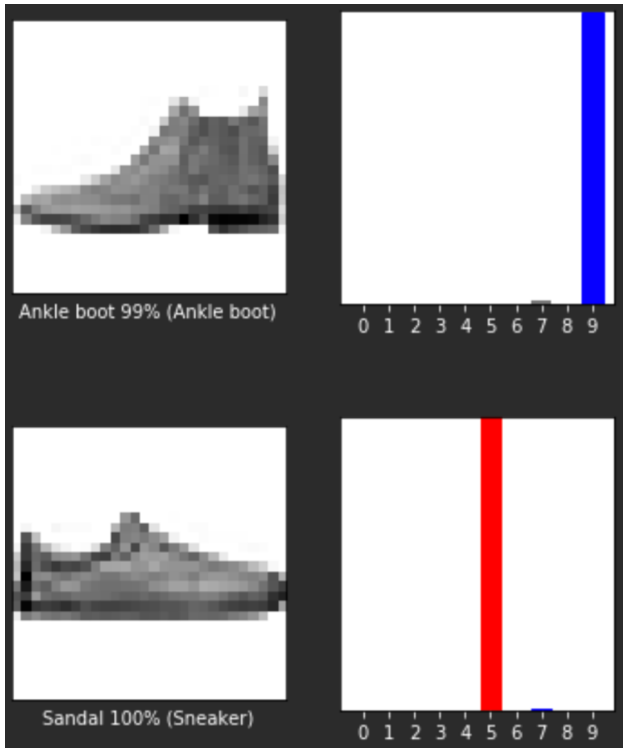


Fig. 3: First Classifier Architecture

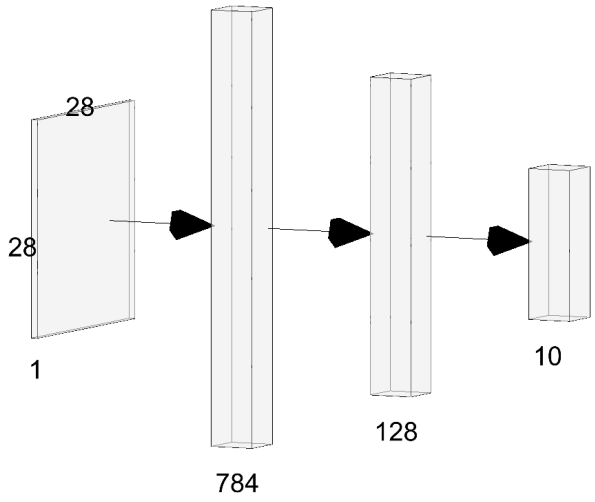
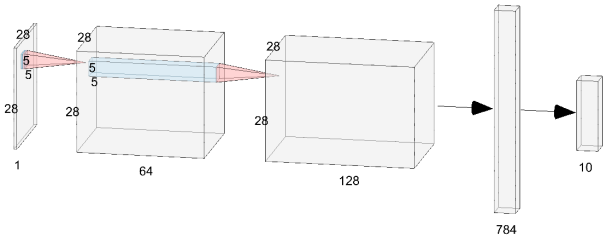


Fig. 4: Second Classifier Architecture



64 layers; the results of the first convolutional layer are then scanned by another convolutional layer with a 5x5 window and 128 layers. The final step is to flatten the layer and densely connect it to the 10-node output layer.

The output layer of both architectures indicates the network's confidence that the image belongs to each of the classes, so a large value in the 0^{th} and 6^{th} nodes would indicate that the network is uncertain whether an image should be classified as a T-shirt/top or just a Shirt. The neural networks described were constructed using Keras and TensorFlow in Jupyter notebooks.

The robustness of the classifier was tested by adding adversarial signals to the input images. The adversarial signal was calculated by determining the gradient of the loss function based on the input image used. This allows the adversarial signal to skew the image to the closest incorrect classification. Once the adversarial signal was determined, it was scaled by an ϵ value (between 0 and 1) and added to the image. To add the noise effectively, the image is considered to consist entirely of pixels that have a brightness value greater than 0. Limiting the noise to non-zero values means the classification isn't being skewed due to noise added in the black background portions of the image. Robustness was determined by comparing the accuracy of our classifiers against ever-increasing ϵ

values.

B. Generation

A key goal from the onset of this project was to use GANs to generate images which are believable reproductions of those in fashion-mnist. During the planning stage, various projects which achieved the goal were identified as references. Two stood out above the rest: tensorflow-generative-model-collections [5] and pytorch-generative-model-collections [6]. Both take the same approach of training ten different GANs on various datasets, including fashion-mnist. The PyTorch implementation proved easier to execute on the SJSU HPC due to its more modern dependency stack, therefore it was chosen as the foundation of the generative component of this project.

The ten GANs explored for this project were:

1) Generative Adversarial Network (GAN; June 2014) [7]: The original GAN paper by Goodfellow, et al. introduced the world to the idea of training competing deep networks to improve generative models.
2) Conditional GAN (CGAN; November 2014) [8]: Followed an idea posed at the end of the original GAN paper to see how GANs might be used to fulfill conditional constraints such that the generated samples targeted a specific output class.
3) Information Maximizing GAN (infoGAN; June 2016) [9]: Maximizes mutual information during training to learn features without supervision which can be tuned to alter characteristics of the generated models (i.e. shifting the angle of MNIST–like digits).
4) Energy-based GAN (EBGAN; March 2017) [10]: An energy-based approach to GANs that yielded promising results for high-resolution images.
5) Auxiliary Classifier GAN (ACGAN; July 2017) [11]: Trains the generator to generate a classification for each sample it produces, eventually proving GANs can learn over numerous diverse classes with high-resolution images. It achieves especially impressive results on ImageNet.
6) Least Squares GAN (LSGAN; April 2017) [12]: Swaps out the sigmoid cross entropy loss function for the least squares loss function to avoid vanishing gradients, thereby forcing the generator to produce higher quality images.
7) Wasserstein GAN (WGAN; January 2017) [13]: A theoretically-dense paper which introduces the critic into the standard game between a generator and discriminator to achieve more durable training and avoid mode collapse.
8) Boundary Equilibrium GAN (BEGAN; May 2017) [14]: Stabilizes training via an equilibrium, but more interestingly, explores the inverse relationship between diversity of generated images (modality) and image quality.
9) Wasserstein GAN - Gradient Penalty (WGAN_GP; December 2017) [15]: Enables training deeper, more complicated networks than previously possible by removing critic weight clipping from WGAN in favor of gradient penalties.
10) Deep Regret Analytic GAN (DRAGAN; December 2017) [16]: Calls into question the assumption that modality and quality are inversely correlated.

Though this project does not dwell too long on the mathematical foundations of GANs, it is beneficial to have a high-level peek at the fundamental equation governing them. (1) shows the minimax game being played by the generator and discriminator.

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z)))] \quad (1)$$

Each GAN architecture was trained independently on the fashion-mnist dataset for fifty epochs. The loss metric was tracked for each training session. To communicate the results of the training process more coherently, a snapshot of images produced by the generator is saved at the end of each epoch as demonstrated in Figure 8. Upon completion, each set of fifty images was compiled into a GIF to aid comprehension.

The code associated with the generation component of this project is hosted on GitHub [3]. Pay special attention to the instructions for executing the application on the SJSU HPC.

C. Colorization

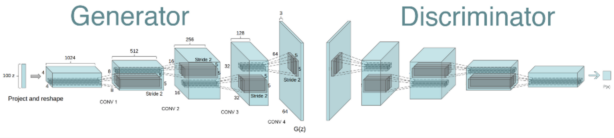
The images in the fashion-mnist dataset are all grayscale images. We decided to build a machine learning model that would fill colors into these grayscale images intelligently. There exist several techniques available to fill static colors into them; however, our goal is to have the model learn to color the images intelligently. We decided to use GANs [17] to perform this task.

To build a model capable of intelligently colorizing grayscale images, we had to first build a dataset for training our GAN model. We combined the training and testing sets of the fashion-mnist dataset. Next, we filled colors into these 70,000 grayscale images as seen in Figure 5. Initially, the shape of each image was (28,28,1). After colorizing the images, their shape changed to (28,28,3). The additional rows hold the red, green, and blue pixel values. Each label of the fashion-mnist dataset was assigned different colors. For instance, ankle boots were colored brown and dresses were colored red. Now, all 70,000 data points in fashion-mnist have been transformed into (28,28,3) matrices to hold colorized representations [18].

Once the training samples were created, experiments with GANs commenced with inspiration from [19]. The original grayscale images were input to the generator. The generator then tried to fill colors into these grayscale images using some randomly set weights. The generated images were then passed on to the discriminator. Along with these images, the discriminator also received the statically-colored images from the training sample. The discriminator tried to identify which image was created from the generator and which was drawn from the training sample. Based on the decision made on the

Fig. 5: Training Samples for Colorization

Fig. 6: DCGAN Architecture



classification task by the discriminator, the loss was computed for both the generator and discriminator.

The discriminator loss is propagated back through to the discriminator to update its weights. A similar procedure also occurs for the generator. This process of updating the weights and learning was carried out iteratively for 50 epochs. After training, the network proved to be capable of colorizing the items in the fashion-mnist dataset intelligently.

DCGAN [20], [21] was chosen to model this problem. Its architecture is displayed in Figure 6. Both the generator and discriminator networks are visualized.

DCGANs use a random noise vector as input. The input is then scaled up into two-dimensional data similarly to CNNs but in the opposite structure. For non-linear layers, we recommend using LeakyReLU for all layers of the discriminator. For the generator, we found that the BatchNormalize layer was the optimal choice. Using BatchNormalize allows the CNN version of the generator to learn better, but it is impossible to use BatchNormalize for all layers. The output layer of the generator and input layer of the discriminator do not require a BatchNormalize layer.

The parameters used during training are listed below:

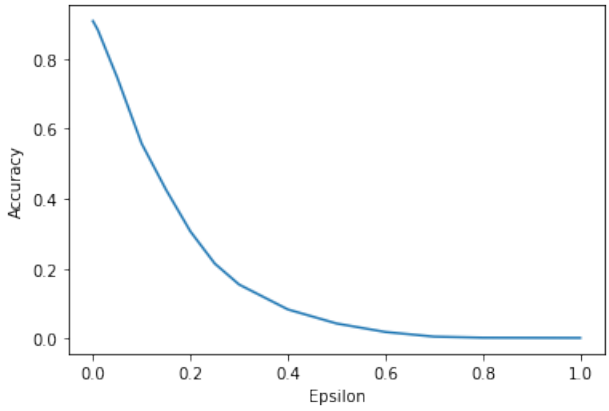
- Batch size: 64
- Optimizer: Adam
- Iterations: 20000
- Learning rate: 0.0001
- BatchNormalization momentum: 0.9

V. Tooling

Each of the components utilizes its own tooling.

Classification was performed by all team members either locally on their computers (PC and Mac) or in Google CoLab. The provided implementation leverages Keras APIs run through a Jupyter notebook. No changes were required to use it. However, the adversarial noise experiments did require

Fig. 7: Accuracy with Adversarial Noise



customization as can be seen in "Color Generator, Classifier, and Adversarial.ipynb" on GitHub [2].

Generation was initially performed by Andrew locally as a Jupyter notebook, but long training times due to the lack of a GPU spurred him to get the code running elsewhere. His first experiment proved the code could work on a Google Cloud Platform node with a GPU, drastically reducing training times. After additional modifications, the app became capable of running on the SJSU HPC which offered fast training times at no cost. The PyTorch implementation and documentation for running the app on the HPC can be browsed on GitHub [3].

Colorization was performed locally and in Google CoLab. The same notebook used for classification was used to experiment with colorization. It uses Keras to construct the DCGAN.

VI. Results

A. Classification

Both classifiers had a training accuracy of 91% and a test accuracy around 90%. Specifically, the dense network achieved 88% accuracy and the convolutional network achieved 90%. This indicates some slight overfitting, but overall both classifiers performed similarly on test data as they did on training data. The classifiers also seem to be fairly robust, since with an ϵ value of 0.05 we still saw a classification accuracy of 75%. Figure 7 shows the accuracy of our classifiers as ϵ is increased.

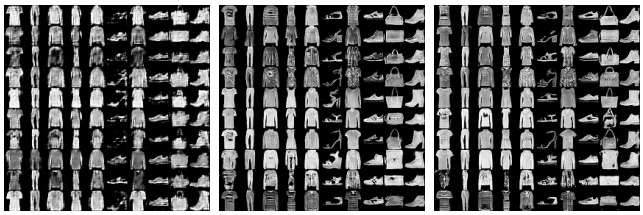
The adversarial noise can be visually noticed with an ϵ of 0.1, so maintaining a 75% accuracy rate when the visual noise is largely unnoticeable to human observers is fairly robust. Based on the work of Goodfellow, Schlens, and Szegedy [22], the ϵ values at which adversarial noise affects a dataset is rather small. Their classifier for the traditional MNIST dataset produces an error rate of 89.4% with an ϵ of 0.25 and an error rate of 87% with an ϵ of 0.1. The difference in our accuracy compared to theirs is probably due to filtering the adversarial noise to only be on the image rather than in the black background as well. The main issues for the classifiers seem to be the similarity between some classes. The classifier has lower confidence in distinguishing between t-shirt and

Fig. 8: GAN Generations

(a) Epoch 1 (b) Epoch 25 (c) Epoch 50

Fig. 9: CGAN Generations

(a) Epoch 1 (b) Epoch 25 (c) Epoch 50



shirt. It also struggles to disambiguate between some sandals and ankle boots. It could be interesting to see how the classifier changes if some of these groups are joined together.

B. Generation

The results of the original implementation were fully replicated for all ten GANs. A select few are highlighted in this paper, but the associated presentation contains the rest.

After reading the papers in which these architectures are described, their unique properties became apparent in the results. For instance, a common problem articulated by many of the papers is the challenge of training stability. They describe the fragility of the balance that exists between the discriminator and generator. Recent advances focus on ways to synchronize their learning, so the discriminator doesn't outpace the generator too early. The plots all tell that story; the discriminator's loss always descends faster than the generator's. Recent research has largely focused on training stability. For instance, WGAN keeps the discriminator and generator losses in balance during training as can be seen in Figure 13. In contrast, the losses for the original GAN shown in Figure 11 and CGAN shown in Figure 12 clearly diverge.

Another interesting observation manifest in the results is that GANs can learn to model specific classes. The original GAN does not constrain its generator to a particular output class, therefore it can be observed to shift its output based on the likelihood of fooling the discriminator. In Figure 8, look specifically at the image directly to the right of the top-left corner (i.e. row 0, column 1) to see the generator switch from producing a Shirt to a Bag. In contrast, CGAN permits learning only on specified classes. Each column of Figure 9 holds predictions in a distinct output class.

Between Figure 8, Figure 9, and Figure 10, it seems the most recent (WGAN) actually produces the lowest quality predictions. Why? We suspect that its attempts to avoid mode collapse (the tendency for a GAN's predictions to pool increasingly around a few high-confidence output classes) come at the cost of granularity. Of course, the DRAGAN authors would disagree. Ultimately, we need to perform additional research to understand this property of GANs better.

C. Colorization

The colorized images produced by DCGAN can be seen in Figure 14.

Fig. 10: WGAN Generations

(a) Epoch 1 (b) Epoch 25 (c) Epoch 50

Fig. 11: GAN Loss Plot

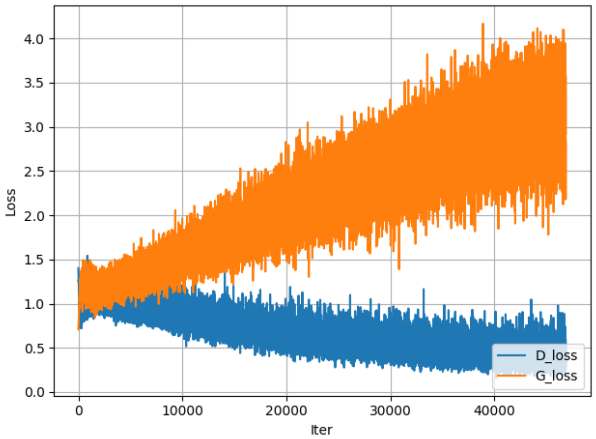


Fig. 12: CGAN Loss Plot

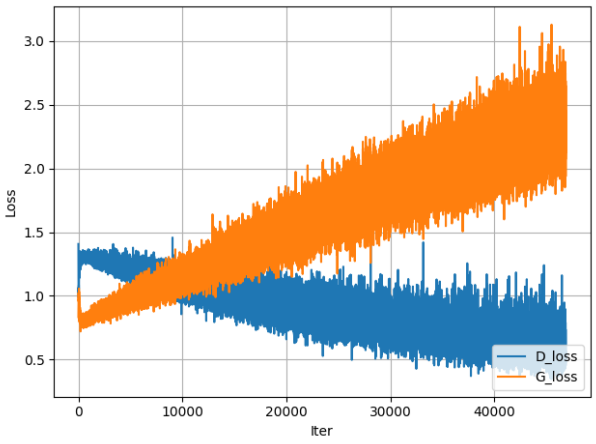


Fig. 13: WGAN Loss Plot

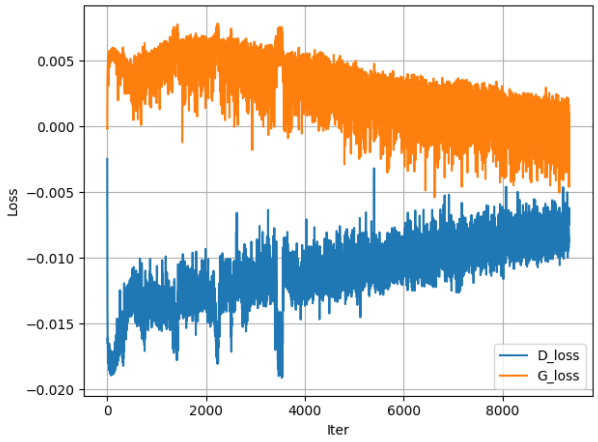
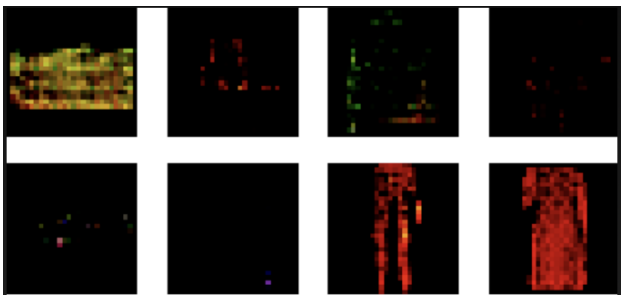


Fig. 14: Colorized Clothing Images



It can be clearly seen that the generated images are not as good as the grayscale examples due to the difficulty in setting up the training set. We tried replacing too many background images and ended up using this new dataset. The generator must generate clothing images and have a valid background (retrieved from real images). With more training, additional training samples, or some special techniques, one might be able to improve the generated images.

For our project, we have used DCGAN to generate colorful clothing images. One can try to change the dataset to generate images that suit their personal needs. Alternatively, modifying the neural network architecture or parameters may lead to more success in achieving realistic colorization.

VII. Conclusion

As evidenced by the diverse branches this project explored, neural networks offer a wide surface area of applications. We successfully leveraged them to perform multi-class classification and generative modeling over fashion-mnist. Furthermore, we explored a breadth of GAN architectures. These provided broad exposure to modern research trends and problems GANs still face. No breakthrough theoretical discoveries were made; nonetheless, our experiments and results lent us critical insight into the forces governing these techniques.

VIII. Contributions

For more information regarding a specific portion of the project, direct communication to the responsible individuals:

1) Classification: Charlie Brayton, Andrew Selvia
2) Adversarial Noise: Charlie Brayton
3) Generation: Andrew Selvia
4) Colorization: Mohit Patel, Dylan Zhang, Charlie Brayton

References

[1] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," 2017.

[2] C. Brayton, M. Patel, A. Selvia, and D. Zhang, "e-in-style," 2020. https://github.com/AndrewSelviaSJSU/e-in-style.

[3] C. Brayton, M. Patel, A. Selvia, and D. Zhang, "Andrewselviasjsu/pytorch-generative-model-collections," 2020. https://github.com/AndrewSelviaSJSU/pytorch-generative-model-collections.

[4] T. Documentation, "Basic classification: Classify images of clothing," 2020. https://www.tensorflow.org/tutorials/keras/classification.

[5] hwalsuklee, "tensorflow-generative-model-collections," 2017. https://github.com/hwalsuklee/tensorflow-generative-model-collections.

[6] H. Kang, "pytorch-generative-model-collections," 2018. https://github.com/znxlwm/pytorch-generative-model-collections.

[7] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," 2014.

[8] M. Mirza and S. Osindero, "Conditional generative adversarial nets," 2014.

[9] X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, and P. Abbeel, "Infogan: Interpretable representation learning by information maximizing generative adversarial nets," 2016.

[10] J. Zhao, M. Mathieu, and Y. LeCun, "Energy-based generative adversarial network," 2017.

[11] A. Odena, C. Olah, and J. Shlens, "Conditional image synthesis with auxiliary classifier gans," 2017.

[12] X. Mao, Q. Li, H. Xie, R. Y. K. Lau, Z. Wang, and S. P. Smolley, "Least squares generative adversarial networks," 2017.

[13] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein gan," 2017.

[14] D. Berthelot, T. Schumm, and L. Metz, "Began: Boundary equilibrium generative adversarial networks," 2017.

[15] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville, "Improved training of wasserstein gans," 2017.

[16] N. Kodali, J. Abernethy, J. Hays, and Z. Kira, "On convergence and stability of gans," 2017.

[17] Q. Fu, W.-T. Hsu, and M. heng Yang, "Colorization using convnet and gan," 2017.

[18] Stwind, "Exploring fashion mnist."

[19] W. Bulten, "Getting started with generative adversarial networks (gan)," 2017.

[20] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," 2016.

[21] "Deep convolutional generative adversarial networks."

[22] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," 2014.