Synthetic Art Using GANs

Team: Amiel Orbach (519, Shubham's cohort), Brian Perriello (519, Halley's cohort), Nicholas Pilotti (419, Hanwen's cohort)

Project Mentor TA: Shubham

Abstract

We attempt to make a "creative" ML algorithm by implementing a GAN that is able to produce images imitating several external datasets. We use two neural networks training against each other, with a generator attempting to create images and a discriminator that tries to distinguish between the images from the dataset and images created by the generator. The discriminator trains by taking in images produced by the generator, along with real images, labeled as true or false, while the generator only receives noise as input (to ensure that it doesn't only produce one image once it is trained). We evaluate the generator by whether it looks like the images to the human eye. We were able to generate images that imitated several different grayscale datasets very successfully, but had less success with RGB images.

Github Link to Code: https://github.com/aorbach/CIS-519-Final-Project.git

Introduction

We chose artwork as the domain of this project because we have outside interests in art history and creating art. Art has historically been thought of as something that only humans can create. We find this project interesting because it pushes against this perspective and touches on philosophical questions about what should be considered proper art. Building a machine learning model that can mimic human creativity in this way is interesting to us. This will show that machines can produce original artwork from noise once it has been trained on real images. The generator model doesn't even see the dataset it is imitating, so it isn't performing any image manipulation; it is only receiving feedback about what art it produces is "good". This is like humans learning by trial and error. Finally, we wanted to learn more about GANs and gain more technical experience working with them, since they are a fascinating branch of machine learning. Yann LeCun, who helped develop the modern deep neural net, said that "GANs are the most interesting idea in the last ten years in Machine Learning."

Related Prior Work

"Generative Adversarial Networks" [1] is the paper by Ian Goodfellow and his colleagues at the Université de Montréal which first posited the idea of a GAN in 2014. It proposes a generative model G and a discriminative model D who are engaged in a game of sorts. G tries to generate samples that imitate a certain training data distribution while D attempts to distinguish whether a sample came from the training data or was generated by G. They describe it as a competition between the police and a team of counterfeiters. Their paper uses multilayer perceptrons, which work well, since they can be trained using backpropagation. An important innovation is that G is able to imitate images that it has never seen, since only D is given the "real" dataset. Radford et al. [2] were the first to use convolutional neural nets (CNNs) in the context of GANs. Within a year of Goodfellow proposing the initial idea, they created a GAN where both the discriminator and the generator used convolutional layers to produce images. The generator used convolutional layers, batch-normalization, and ReLU and the discriminator used convolutional layers, batch-normalization, and leaky ReLU. Since that publication, there has been a particular emphasis on using GANs to synthesize images. Huang et al. [3] focus on different methods used to generate images and discuss the strengths and weaknesses of the various options. They compare the standard deep convolutional GAN (DCGAN) to other architectures, such as using multiple generators and/or multiple discriminators. The Hierarchical Method uses two of each, where each algorithm focuses on a different component of the image, such as "styles and structure" or "foreground and background". The Iterative Method uses more than two of each model, and they work to generate an image from coarse to fine details. Arjovsky et al. [5] established an improved way of training GANs to avoid "mode collapse", the phenomenon in GANs where different noise inputs will produce the same output due to vanishing

gradients and failing to train properly. It advises not using an output activation function and using the raw score of the discriminator as the loss function for both the discriminator and the generator, as well as using RMSprop as the optimizer.

Formal Problem Setup (T, E, P)

T (Task): We propose to make a generative adversarial network (GAN) that takes in a dataset of artwork/images along with random noise and is able to generate synthetic images that imitate the pictures provided.

E (Experience): We used several grayscale datasets of images from Keras and SKLearn to train small-scale GANs. We used the MNIST, Fashion MNIST, Eigenfaces, and Labeled Faces in the Wild datasets. We also tried using RGB images of abstract art from Kaggle. Those are the "real" images. These allowed us to produce several different models, each trained on a different type of art, depending on the size of the images. Once we trained these models, we can tell the generator to output images that imitate the dataset it was trained on. The only other input is random noise, which we can generate using PyTorch functions.

P (Performance Metrics): A GAN uses two networks, each with their own loss function. The generator's loss increases when the discriminator isn't fooled and shrinks when it is able to fool the discriminator. The discriminator's loss function is the negative of the generator's, as well as an additional function that increases when it misclassifies the training data and goes down when the "true" images are classified correctly. To evaluate the performance of the GAN as a whole, we can use two performance metrics, one numeric and one subjective. One is whether the generator is able to consistently fool the discriminator. This means that the discriminator has an accuracy that is no better than random guessing. This can be calculated after training, but is not particularly relevant to the quality of the generator, as the discriminator is only used to train the generator and we don't particularly care about how good it is, given that the generator works as intended (Shmelkov, et al.). The discriminator could be good or bad after training, as long as the generator outputs recognizable images. The more important metric we can use is subjective. We can tell the generator network to output images and we can see whether they look like recognizable pictures that resemble the images in the dataset we are using to train the discriminator.

Methods

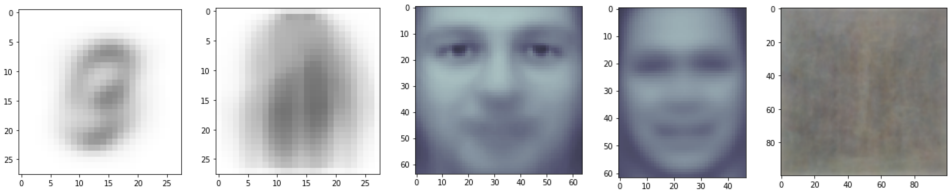
We procured several datasets by importing them from Python libraries (Keras and SKLearn). As grayscale images, they were a two dimensional grid of pixels, each of which had an integer value between 0 and 255 (inclusive). We converted them into PyTorch tensors and standardized them so that the pixel values were between -1 and 1, with a mean of 0. Although this warped some of the images, it greatly improved performance. Standardizing images helps speed up convergence to optimal values and increases computation speeds for neural networks. We then converted them into dataloaders for ease of training using the Dataset and DataLoader classes in PyTorch. This allows us to use mini-batch learning, which is faster than batch gradient descent and not as random as stochastic gradient descent. We tried several different batch sizes, but a size of 10 seemed to work the best. We made a joint training loop that alternates between improving the discriminator and the generator. It first trains the discriminator against a batch of images from the dataloader. Then, the generator produces a batch of images, which are classified by the discriminator. The loss is then propagated backward to the discriminator, but not the generator. The generator then outputs a new batch which is classified by the discriminator. This time, the gradient is propagated backward through the generator, not the discriminator. We also employed several tricks during training that together make up a variant of the classical GAN called a Wasserstein GAN. A Wasserstein GAN uses the trick of training the discriminator several times more than the generator. Otherwise, the generator is too good and results in the discriminator classifying all of the images as either real or fake. We accomplished this by having the generator only train every five batches. We repeat this loop for many epochs until the images being output are visually satisfactory (usually between 20 and 50). We also noticed that the discriminator is often able to achieve high accuracy classifying the real images but has lower accuracy on the fake images. We were able to remedy this by training twice on the generated images for every time it trains on the real images. This helped balance the accuracy of the discriminator on both sets of images. Along with the other tricks employed by the Wasserstein GAN (in implementation details), we were able to imitate several different datasets successfully. Once we were able to produce grayscale images, we attempted it again with RGB images. We downloaded a dataset of

abstract art from Kaggle (https://www.kaggle.com/bryanb/abstract-art-gallery), resized them all so that the images would be of uniform shape, and created a new GAN. This one needed many more parameters, since RGB images have triple the number of features as grayscale images. These pictures were also larger, further increasing the number of features. We ran the Wasserstein GAN again on these, but had less success, due to the increased complexity associated with having three channels instead of one.

Baseline approaches we compare against:

It is hard to come up with a baseline for generating an image. What we can use is look at the average image from each dataset, made by adding up the pixel values and dividing by the number of images. This isn't what the GAN is doing, but it would be a comparison of what a deterministic algorithm could output to imitate a dataset. GANs don't have a widely accepted evaluation metric other than visual inspection, so we can't objectively classify how well it does (Shmelkov, et al.). The other goal we have is to be able to use similar architectures for the discriminator and generator for different datasets instead of using wildly different architectures for each occasion, which is what many GANs are forced to do. We could evaluate this part by how similar our architectures are to each other (not counting adjusting for the size of the images). The averages of the datasets we used are as follows:

Left to Right: Average of MNIST, Fashion MNIST, Eigenfaces, Labeled Faced in the Wild, Abstract Art



Implementation Details:

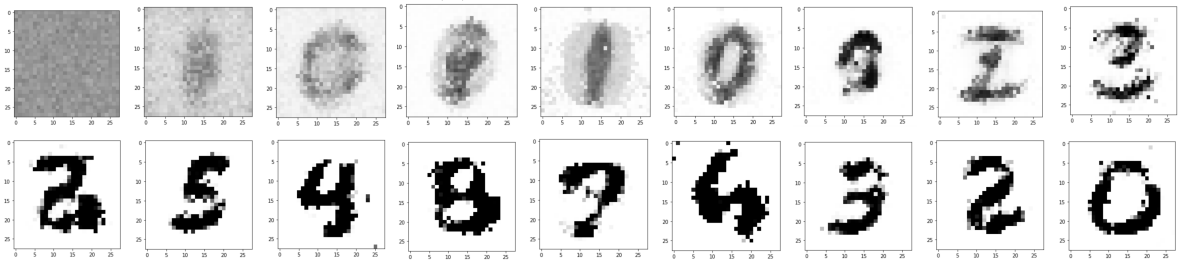
We first tried making a DCGAN to imitate the MNIST dataset (handwritten digits), which used a combination of convolutional layers and fully connected layers, with ReLU activation functions and a sigmoid output. This was able to generate vaguely formed blobs in the center of the image, so it was able to recognize that the digits were in the center of the image, but couldn't recreate the digits. Using only fully connected layers performed slightly better, but still couldn't produce realistic looking pictures. We decided to standardize the images before inputting them into the model, which warped the images a little but led to better results. We also decided to incorporate dropout layers in the discriminator model, which further improved performance and made it more robust to deviations in specific pixels. The final trick we used was to change the discriminator output and loss function to create a Wasserstein GAN. Instead of the discriminator output being passed through a sigmoid function and using a cross entropy loss function, it has the discriminator output the raw score for each image. It then receives a loss equal to the score of the image if the image was fake and the negative score if the image was real. The generator receives a loss of the negative of the discriminator's score. This linear loss function helps prevent convergence issues such as the vanishing gradient problem, known as mode collapse in a GAN. The Wasserstein GAN also uses RMSprop as an optimizer, instead of other methods such as Adam. This is because GANs often have trouble converging, and using momentum can knock it out of a precarious optimum. Normally this is a good thing, but GANs often want to be at such a place in the gradient space. It also provides further help with the vanishing gradient problem. Finally, we clipped the weights with every iteration of training, which helped stop it from converging too quickly to an adverse location. The combination of these tricks greatly improved our model and allowed it to imitate the various datasets successfully.

Experimental Results:

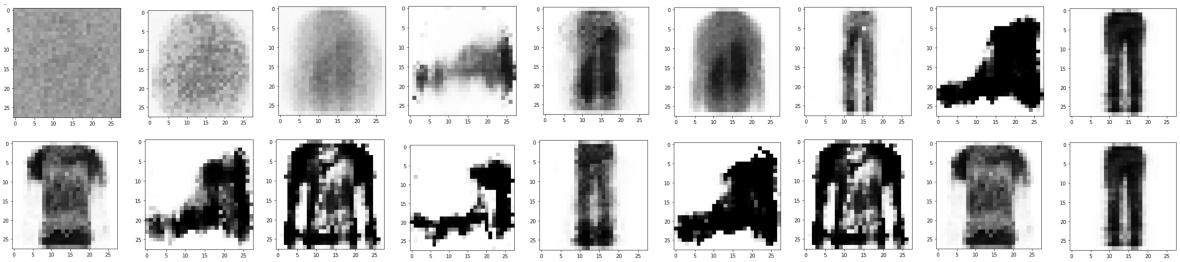
When the models first start training, the generator outputs static noise. After a few batches, it is able to create vaguely shaped blobs in the center of the image. This is good because it recognizes that the pixels of concern are in the center, while the ones at the edges should all be white. This blob slowly starts to develop features imitating the dataset until it looks like a dark shape with some blur around it. After several more epochs, the shapes resolve into recognizable shapes that successfully imitate the dataset. This can be seen in the following ordered series of images from the various grayscale datasets we used. With the abstract art, the GAN was able to generate what looks like more vividly colored

versions of the average image of the dataset. This looks better than the average, since the average looks like a dull blur, but the GAN was unable to generate images that looked very different from the average. This was due to several factors. The first one is that we had to use a very small dataset, because the images contained much more information (having three channels instead of one) and were all different sizes. Having to load the images, resize them, and copy them into arrays would often slow down and crash the program, limiting the size of the dataset we could use. Below are a selection of images generated by our GAN models. Notice that they first start out by producing static, which resolves itself to look like the images in the respective datasets.

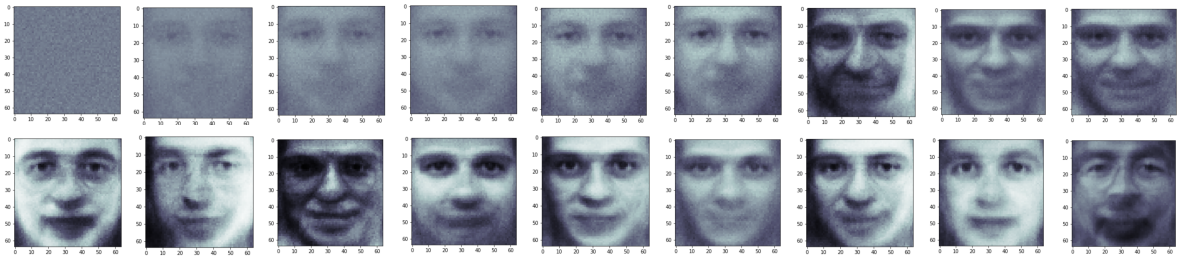
MNIST Generated Images



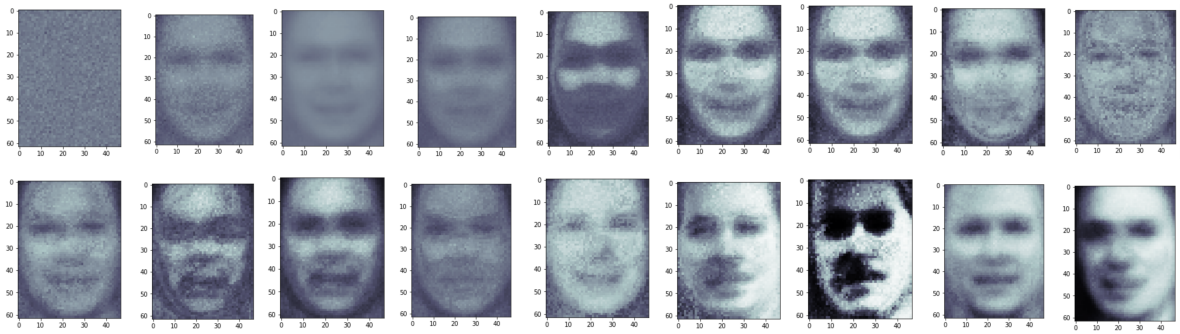
Fashion MNIST Generated Images



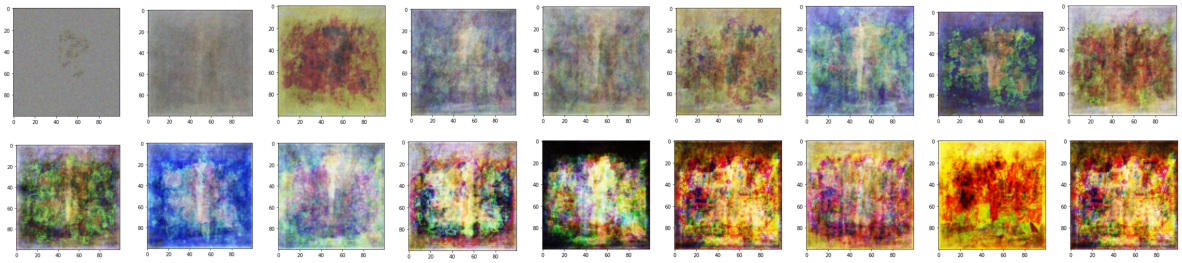
Eigenfaces Generated Images



Labeled Faces in the Wild Generated Images



Abstract Art Generated Images



Once it converged, it was able to imitate a large variety of images and didn't experience mode collapse where it only produced one image, regardless of the noise input. The outputs of the generator are visually recognizable as members of the dataset. The larger the dataset and the more varied the images, the longer the models took to be able to generate successfully. The eigenfaces converged quickly, since the faces share many features. The Fashion MNIST dataset, on the other hand, consists of several different images, such as shirts, pants, and shoes. The output of the corresponding generator stayed fuzzy for longer and sometimes stayed a little blurrier than the other datasets. The Labeled Faces in the Wild dataset was the least successful of the grayscale images, since there were many different faces, sharing fewer features in common than the eigenfaces dataset. This resulted in blurrier images, even after the generator was able to output shapes that were recognizable as human faces. The other success we had is using the same architecture for each of the datasets. We adjusted some of the layers due to the images being different sizes, but the basic order of the layers is the same. This avoided the problem of having to come up with a different architecture for each dataset and made it more robust to different image sets.

Conclusions and Future Work

We were able to successfully generate grayscale images, which only had as many features as it has pixels. RGB images have triple that amount, which makes creating a GAN with them much harder. We were able to create what looked like abstract art, but it resembled a colored version of the average image of the dataset rather than any individual picture. We could explore this avenue further by trying different architectures of discriminators and generators, both using convolutional networks and fully connected networks. We could also attempt to use smaller images with fewer features, which should be easier to imitate. We tried this using the CIFAR-10 dataset, which is 32x32x3. While the GAN was able to generate differently colored blobs and hazy shapes, it still could not produce recognizable images. We could try to produce colored images using different architecture such as inverse convolution/transpose convolution layers. Another direction we could explore is a CycleGAN, which takes in an image and attempts to modify it in a specified way, such as changing the season in which the image was taken. This involves mapping the images to a latent feature space, altering it there, and then changing it back to pixel space to produce the final output.

Ethical Considerations and Broader Impacts:

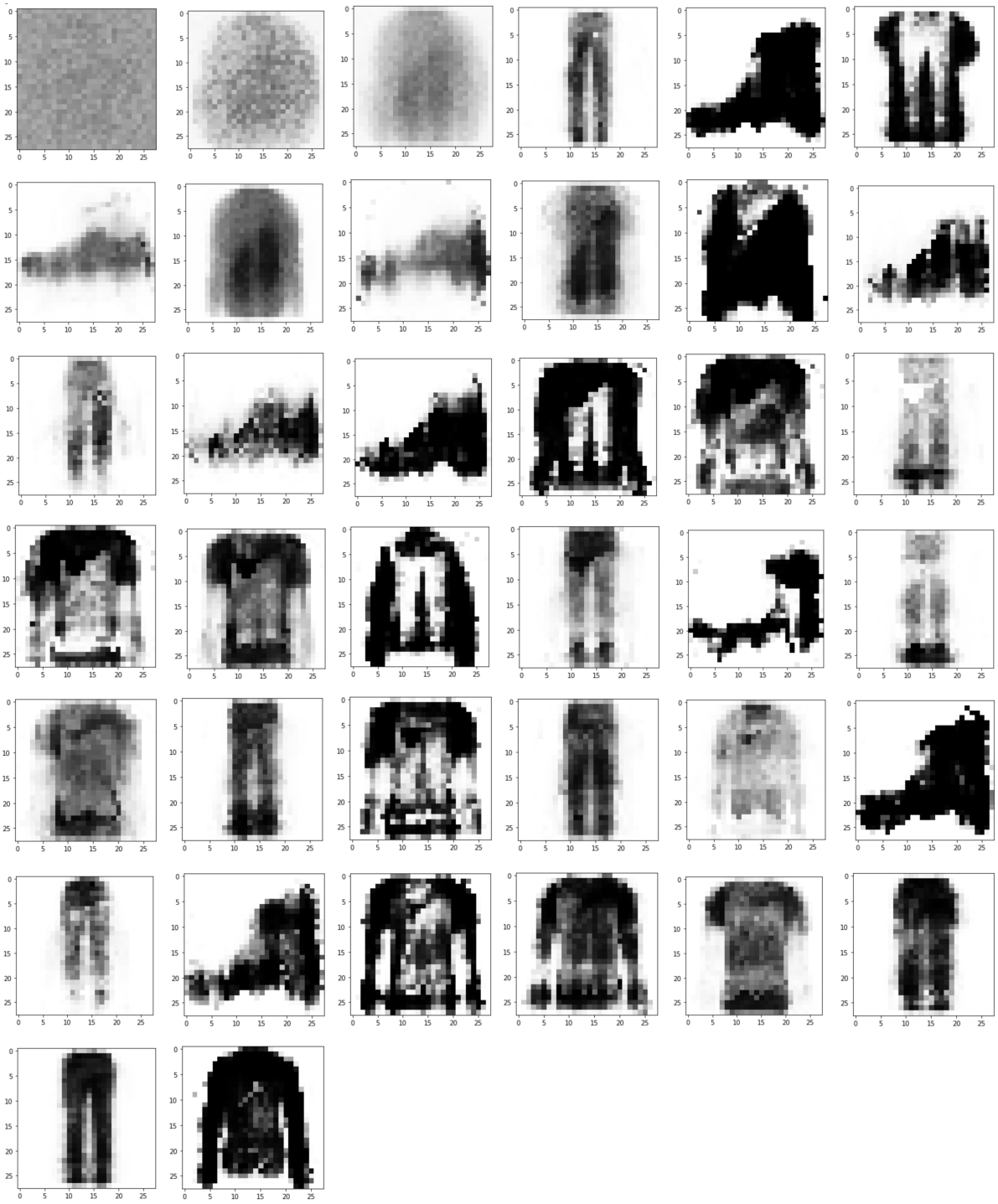
Using machine learning to generate creative output leads to several ethical questions. First, would we want to live in a world where machines can reliably create "better" art than humans? Would this render human artists obsolete? Would the benefits of being able to enjoy the superior AI art outweigh the potential costs imposed on human artists? Perhaps in the future there will still be a market and a special appreciation for human-created art, even if it isn't as aesthetically pleasing as the art made by machines.

Prior Work / References:

1. Goodfellow I, Courville A, Warde-Farley D, Bengio Y, et al. “Generative Adversarial Networks”. *Advances in Neural Information Processing Systems.* June 2014.
2. Radford A, Metz L, Chintala S. “Unsupervised Representation Learning with Deep convolutional Generative Adversarial Networks”. *International Conference on Learning Representations.* 2015.
3. Huang H, Yu P, Wang C. “An Introduction to Image Synthesis with Generative Adversarial Nets”. *Arxiv.org*. November 2018.
4. Brownlee, J. “How to Evaluate Generative Adversarial Networks”. *Machine Learning Mastery.* August 2019. https://machinelearningmastery.com/how-to-evaluate-generative-adversarial-networks/
5. Arjovsky M, Chintala S, Bottou L. “Wasserstein GAN”. *Arxiv.org.* December 2017.
6. Shmelkov K, Schmid C, Alahari K. “How Good Is My GAN?”. *European Conference on Computer Vision.* September 2018.

Supplementary material:

Fashion MNIST Generated Images



Eigenfaces Generated Images

Labeled Faces in the Wild Generated Images

Abstract Art Generated Images