ASLetter - Generating ASL Images From Letters

David Levit

Nir Ben Haim

July 3, 2023

Introduction

In the world as it is today, most people are aware of sign language, but do not really know the signs and cannot communicate using it. In our project we aimed to create a model that receives as an input English letters and produces an image, which contains the sign of this letter in ASL (American Sign Language). To do so, we trained a neural network on a well-known dataset called "Sign Language MNIST". Specifically, we used Generative Adversarial Networks (GAN).

Dataset

In order to actualize our idea, we needed a good, wide dataset. We found one which completely fits our needs - "Sign Language MNIST"[1]. This dataset includes about 34,000 photos, each with a size of 28x28, and in black and white. The images contain different hand positions, that describes different letters in ASL. The given data is also labelled, which means that for each image we are also given the letter that the image describes. The dataset was divided into two groups - train set and test set.

GAN

A Generative Adversarial Network (GAN) is a class of machine learning frameworks and a prominent framework for approaching generative AI. The concept was initially developed by Ian Goodfellow and his colleagues in June 2014.

In a GAN, two neural networks compete with each other in the form of a zero-sum game, where one agent's gain is another agent's loss. Given a training set, this technique learns how to generate new data (images, in our case) with the same statistics as the training set. For example, a GAN trained on photographs can generate new photographs that look at least superficially authentic to human observers, having many realistic characteristics.

Although originally proposed as a form of generative model for unsupervised learning, GANs have also proven useful for semi-supervised learning, fully supervised learning, and reinforcement learning. The core idea of a GAN is based on the "indirect" training through the discriminator, another neural network that can tell how "realistic" the input seems, which itself is also being updated dynamically. This means that the generator is not trained to minimize the distance to a specific image, but rather to fool the discriminator. This enables the model to learn in an unsupervised manner.

Mathematical Expression

The original GAN is defined as the following game: There are 2 players: generator and discriminator. The generator's strategy set is $\mathcal{P}(\Omega)$, the set of all probability measures μ_G on Ω. The discriminator's strategy set is the set of Markov kernels $\mu_D : \Omega \to \mathcal{P}[0,1]$, where $\mathcal{P}[0,1]$ is the set of probability measures on [0,1]. The GAN game is a zero-sum game, with objective function:

$$L(\mu_G, \mu_D) := \mathbb{E}_{x\sim\mu_{ref}, y\sim\mu_D}(x)[\ln(y)] + \mathbb{E}_{x\sim\mu_G, y\sim\mu_D}(x)[\ln(1-y)]$$

The generator aims to minimize the objective, and the discriminator aims to maximize the objective. The generator's task is to approach $\mu_G \approx \mu_{ref}$,that is, to match its own output distribution as closely as possible to the reference distribution. The discriminator's task is to output a value close to 1 when the input appears to be from the reference distribution, and to output a value close to 0 when the input looks like it came from the generator distribution.

WGAN and Wasserstein distance

The Wasserstein Generative Adversarial Network (WGAN) is a variant of Generative Adversarial Networks proposed in 2017 that aims to improve the stability of learning, get rid of problems like mode collapse, and provide meaningful learning curves useful for debugging and hyper parameter searches[2].

Compared with the original GAN discriminator, the Wasserstein GAN discriminator provides a better learning signal to the generator. This allows the training to be more stable when the generator is learning distributions in very high dimensional spaces.

In mathematics, the Wasserstein distance or Kantorovich–Rubinstein metric is a distance function defined between probability distributions on a given metric space M.

CGAN

A Conditional Generative Adversarial Network, or CGAN, is a type of GAN that includes the conditional generation of images by a generator model.

In CGANs, a conditional setting is applied, meaning that both the generator and discriminator are conditioned on some sort of auxiliary information (such as class labels or data) from other modalities. As a result, the ideal model can learn multi-modal mapping from inputs to outputs by being fed with different contextual information. By providing additional information, we get two benefits. First, convergence will be faster, even the random distribution that the fake images follow will have some pattern. Second, we can control the output of the generator at test time by giving the label for the image you want to generate.

In our project's context, this means adding 26 (as the number of English letters) additional channels to the image inputted to the discriminator, in which only channel has non-zero elements which corresponds to the correct label. For the generator, we instead add a 1-dimensional vector with 26 entries to the inputted feature vector, once again where only one element is non-zero which corresponds to the correct label.

The Model

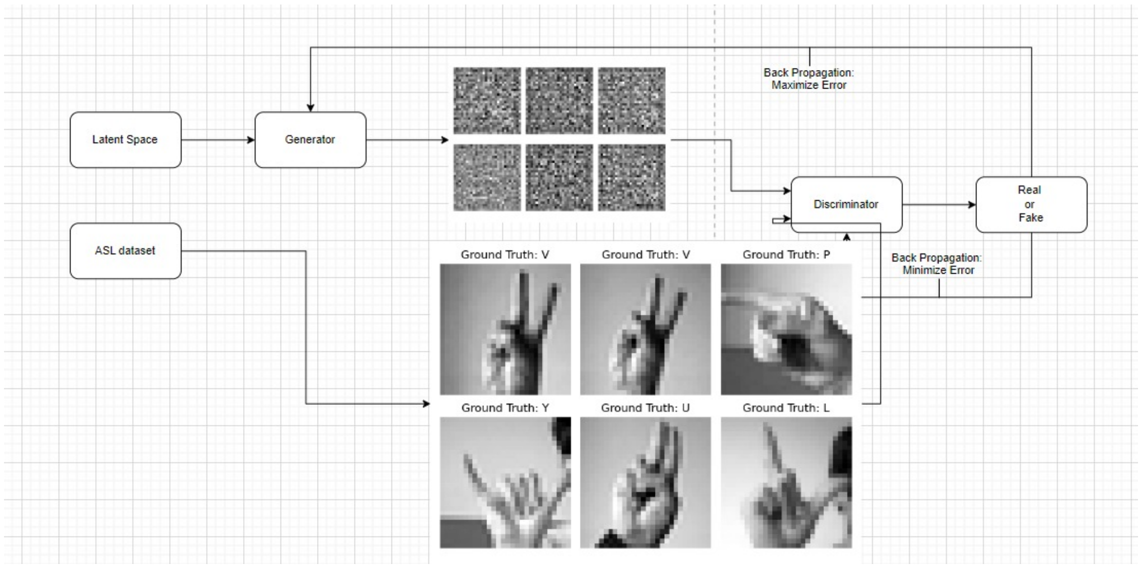


Figure 1: our model

Our model is divided into two parts, the generator is responsible for producing the images, while the discriminator is responsible for controlling the quality of the generated images. Each of these models are built from two parts, one of which is a fully connected (FC) network, while the other is a convolutional network.

Both models include the FC network, which consists of linear layers, dropout, batch normalization layers and LeakyReLU layers; and the convolutional network, which consists of 2D convolution layers, 2D batch normalization and LeakyReLU layers. The discriminator also includes layers of AvgPooling.

We added to our model different features and hyper parameters to improve stability and produce better images.

We tested several kinds of optimizers like ADAM, SGD and RMSprop and we found out that RMSprop improved the quality of the generated images. Also, We added a scheduler, that changes the value of the learning rate every epoch so the losses' rate won't get stuck on a certain value.

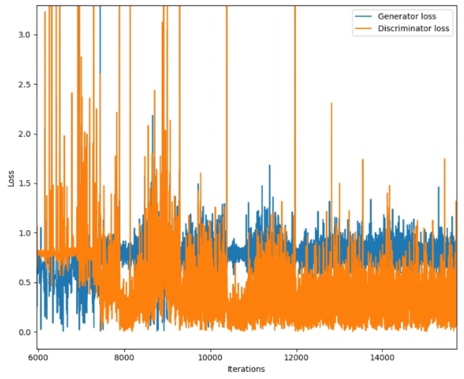
The sensitivity of the model lead to another important issue: parameter tuning. This includes the number of epochs, learning rates, batch size and latent space dimension.

As the GAN requires some random vector (known as noise) as input in order to generate an image, we had some freedom in the initialization of this vector, choosing to sample it from a Gaussian distribution. Additionally, we used "softer" fake/real values; instead of binary values of 0 and 1 we chose 0.1 and 0.9 respectively.

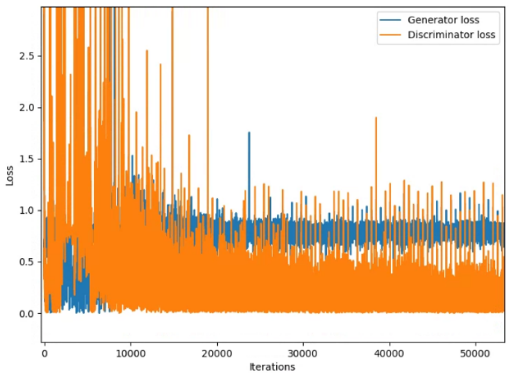
The last feature we added is updating the WGAN to a CGAN model, so that the training also includes the labels. Due to this change, now we were able to add the option for the user to select an English letter so the model will provide images that correspond to the requested letter.

This can all be seen demonstrated in figure 1.

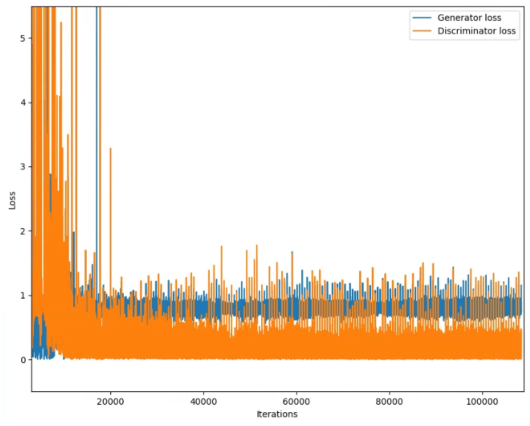
Training and Losses



(a) losses of generator and discriminator, 30 epochs



(b) losses of generator and discriminator, 100 epochs



(c) losses of generator and discriminator, 200 epochs

Figure 2: effects of different number of epochs on the losses

As we already mentioned, the model is very sensitive to changes in hyper parameters as well as to the number of epochs on which the model is trained. At the start, as we increase the number of epochs the results improve and we get better quality images. From a certain threshold of epochs, we may start overfitting on the dataset, which will cause the model to generate a set of images that look similar or even identical to one another. This is a parameter that must be tuned for its optimal value. In our model, it is hard to find an optimal value, as we will see later in the fact that different letters require a different number of epochs. That is because some of them are easier to learn, simpler shapes, while others are much more complicated.

Figure 2 shows the change in the generator and discriminator losses as a function of the number of epochs. We want the discriminator losses to be 0.5 as this means it cannot differentiate between real images and generated ones. Likewise, we want the generator's loss to be minimal, as its loss is calculated purely by how much it manages to fool the discriminator.

Data Augmentation

In order to improve the training, we thought that more information is needed, so we augmented the images. The images of sign language have a directional meaning, therefore it has limits to which augmentation can be performed. We chose to rotate at small angles (random rotate, up to 20 degrees) as well as cropping the images.

Adding the augmentation did not really improve the model because the augmented data was too similar to the original images, which caused overfitting. Eventually, we did not include the data augmentation. In order to increase the data, we trained the model on the train set and the test set (seeing as in GANs the test set is not used).

Results

After our model finished training, we compared the generated images to the real ones.

The original images can be seen in figure 3, while the generated images can be seen in figures 4, 5, 6, representing 30 epochs, 100 epochs, and 200 epochs respectively.

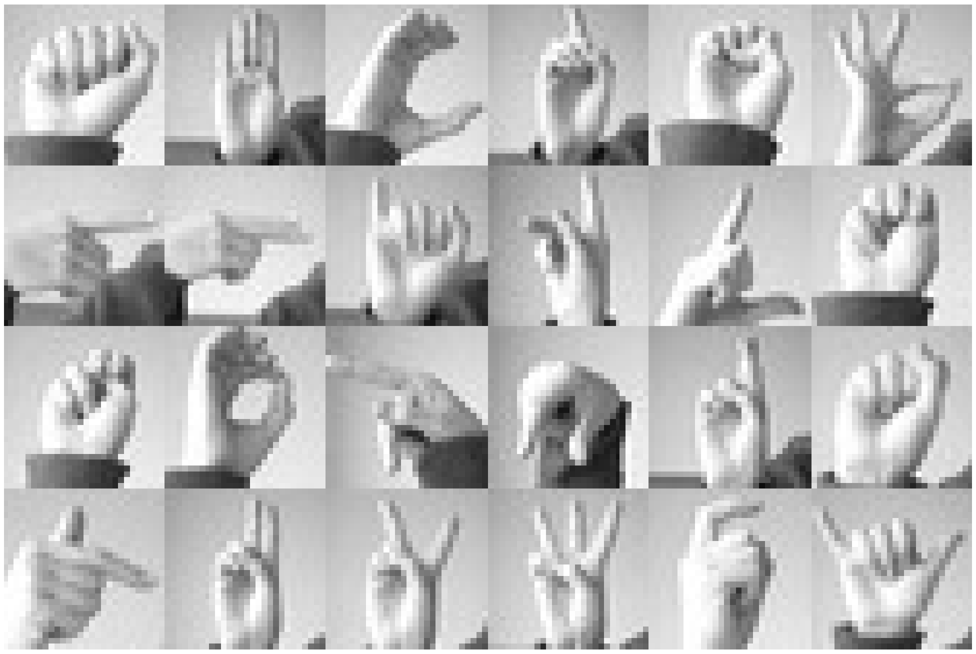


Figure 3: original data

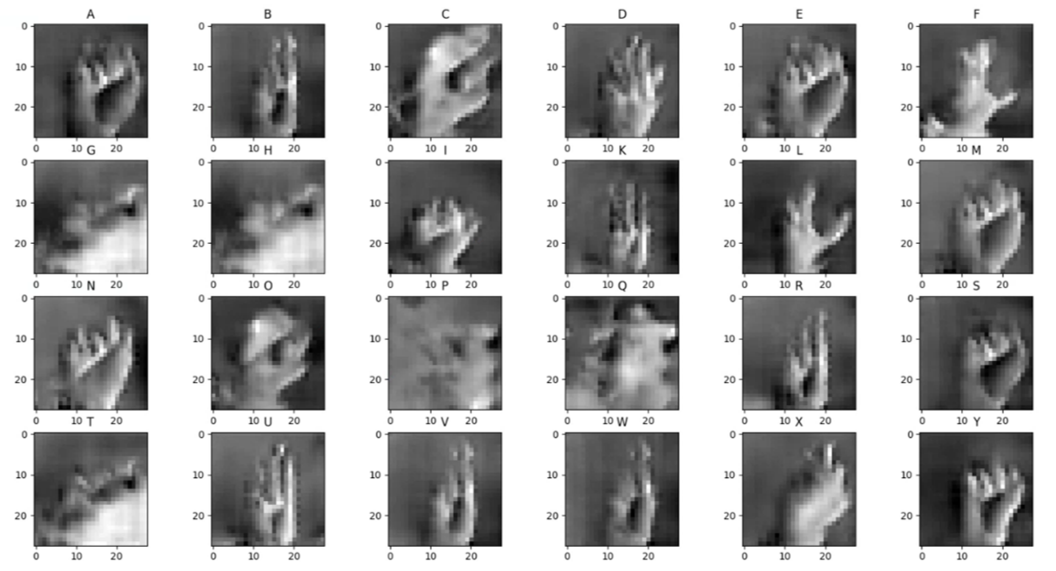


Figure 4: generated images after 30 epochs

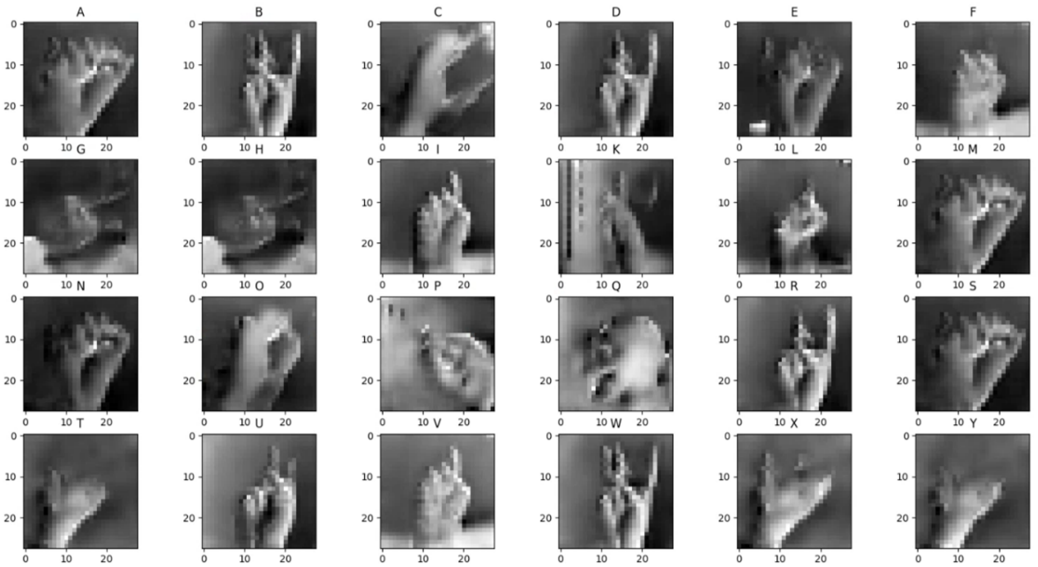


Figure 5: perfect generated images after 100 epochs

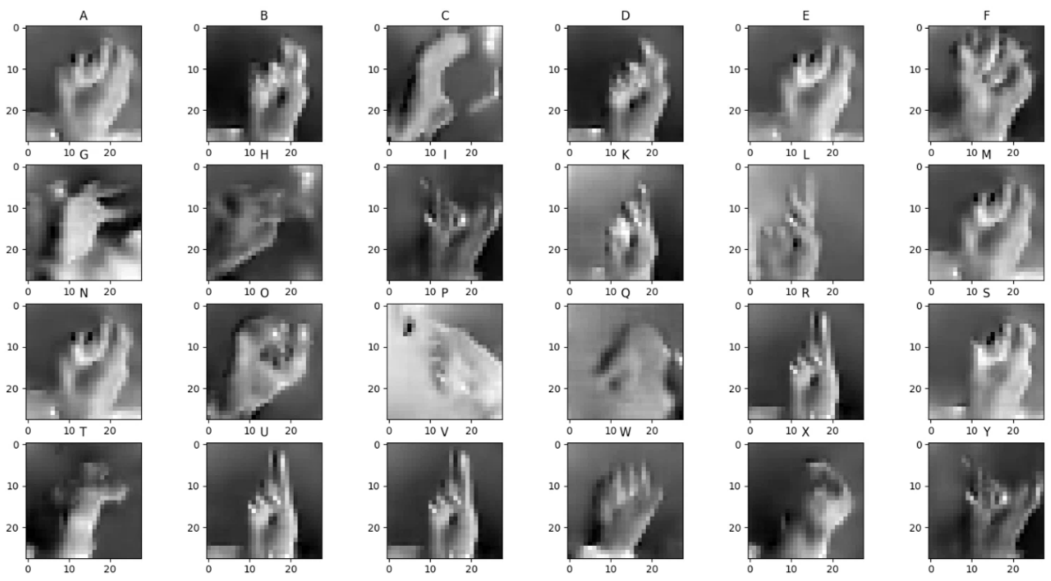
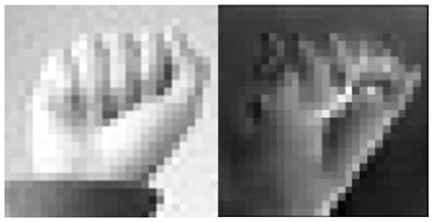


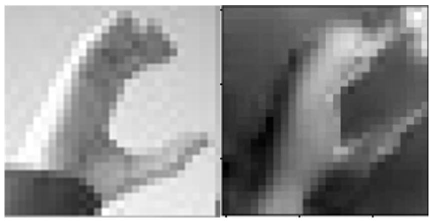
Figure 6: generated images after 200 epochs

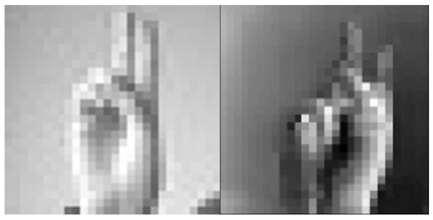
Looking at figure 5, we can clearly see that there are general characteristics of hands, like fingers. Some of the images are very realistic, similar to the original data and manage to capture the hand gestures.

Figure 7 shows examples of generated images which match their original counterparts. In these examples we can see both the fine and rough features accurately conveyed in the fake images.

On the other hand, figure 8 shows examples of generated images which still need improvement. In these examples we cannot tell what the letter is supposed to be as the model only managed to capture either fine (a) or rough (b) features, but not both.


(a) letter A


(b) letter C


(c) letter U

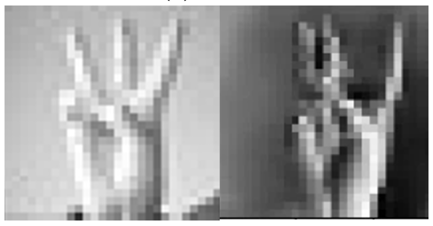
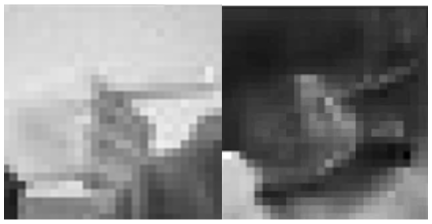

(d) letter W

Figure 7: original images (left) and accurate generated images (right)


(a) letter G

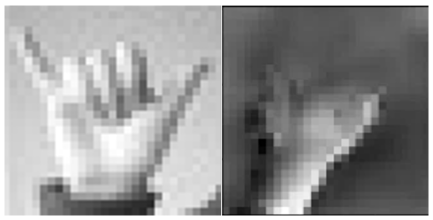

(b) letter Y

Figure 8: original images (left) inaccurate generated images (right)

Going Forward

We feel that this project can be improved and not just in the sense of better accuracy. Further research and experiments can focus in data augmentation techniques and trying different architectures, such as stylegan2-ada, which heavily uses data augmentation. We think that more data would help to train more complex letters. Also, we're very enthusiastic about trying and implementing a self-supervised GAN.

Links

- Git repository

References

[1] https://www.kaggle.com/datasets/datamunge/sign-language-mnist

[2] Arjovsky, Martin; Chintala, Soumith; Bottou, Léon (2017-07-17). "Wasserstein Generative Adversarial Networks". International Conference on Machine Learning. PMLR: 214–223