Gans (generative adversarial networks)

By eng.Ahmed Hisham

GANs

A generative adversarial network is a class of machine learning frameworks designed by Ian Goodfellow and his colleagues in June 2014. Two neural networks contest with each other in a game. Given a training set, this technique learns to generate new data with the same statistics as the training set.

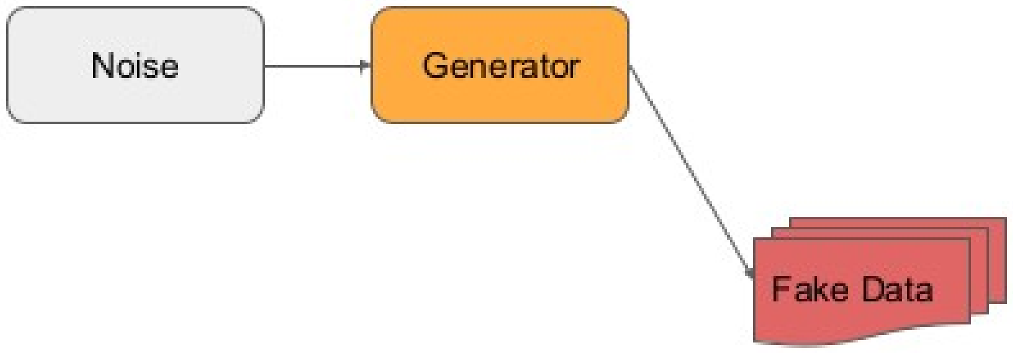
Gans Network

2 neural networks

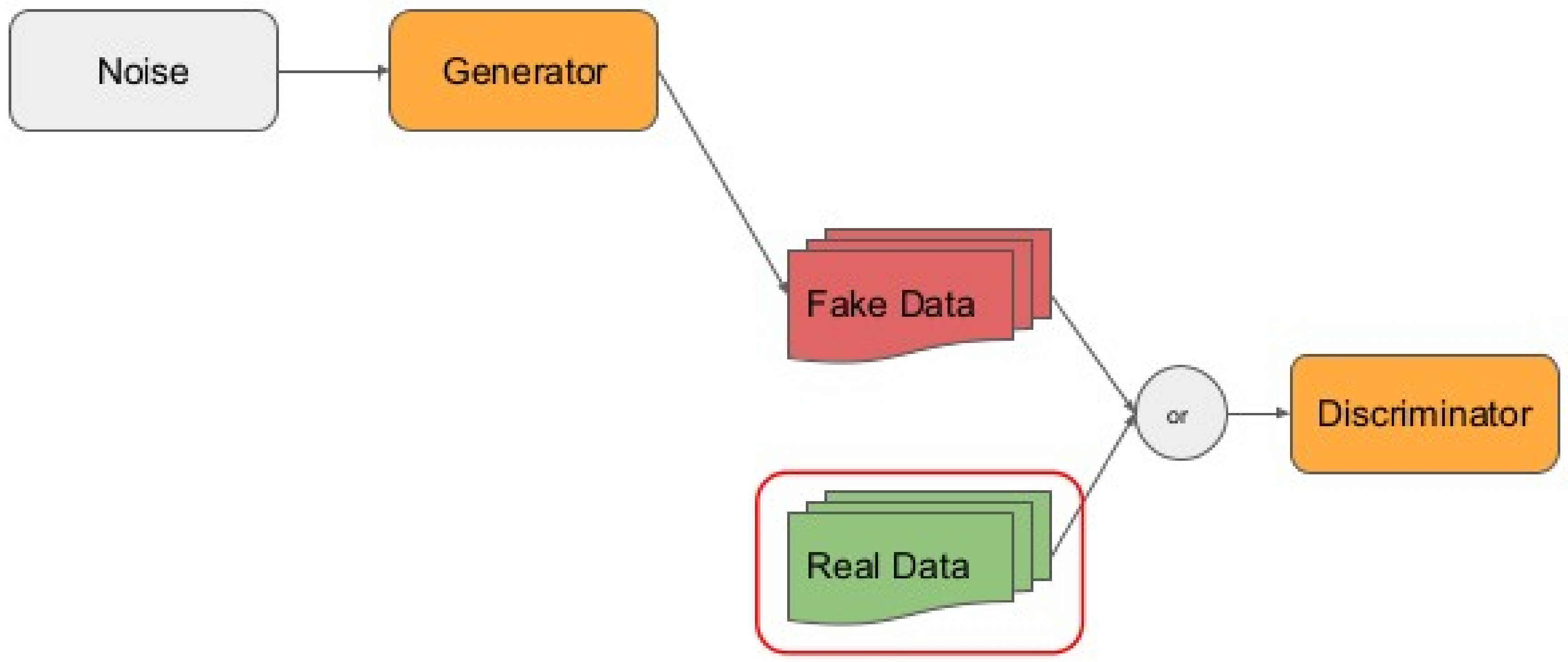
1st Generator used to generate Fake data

2nd discriminator used to tell if the data is real or fake

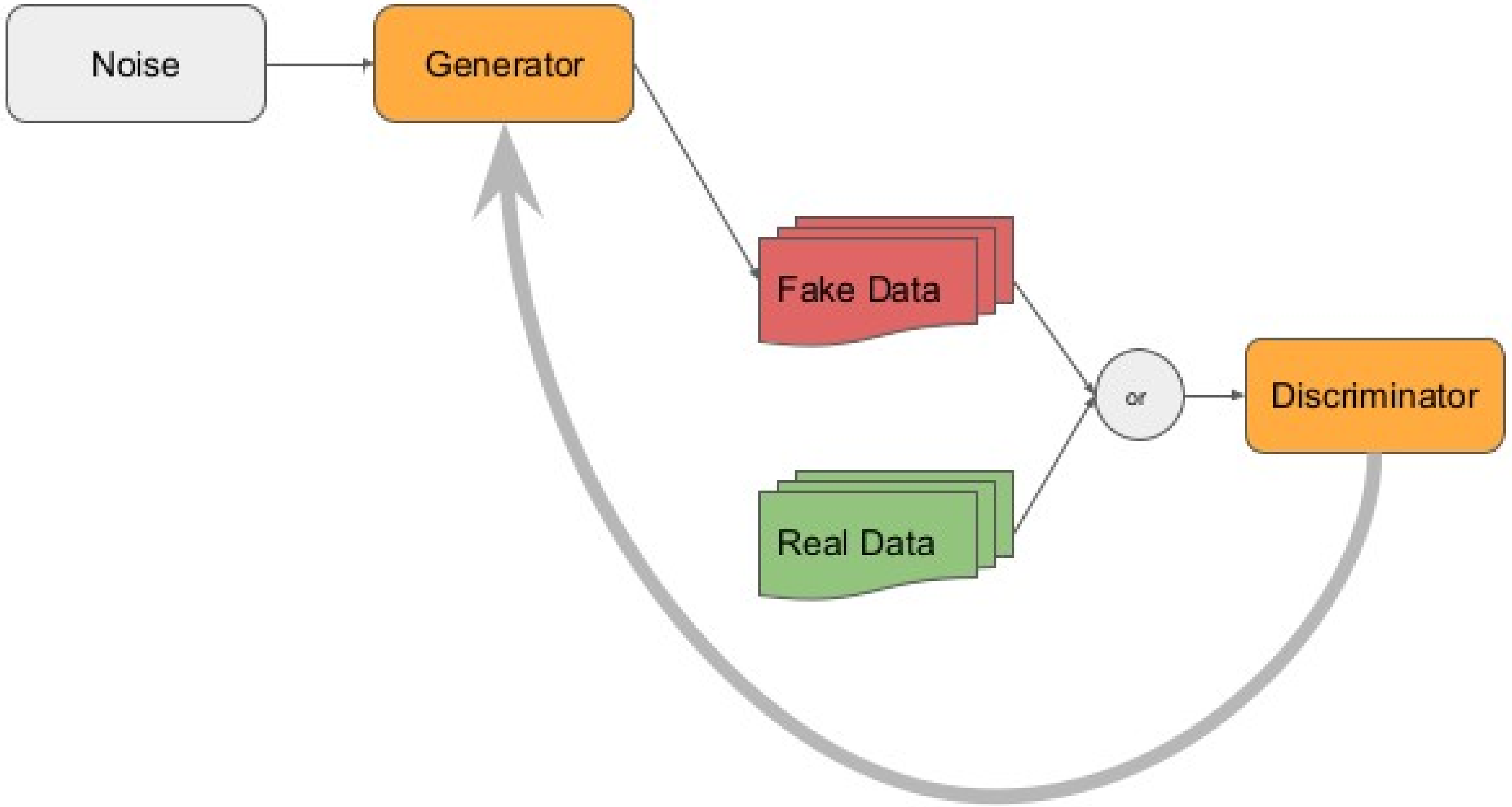
gans



Gans



Gans



Procedures

The generator takes in random noise and uses it to create fake images. For that, this model will take in the shape of the random noise and will output an image with the same dimensions of the MNIST dataset (i.e. 28 x 28).

Procedures

Discriminator

The discriminator takes in the input (fake or real) images and determines if it is fake or not. Thus, the input shape will be that of the training images. This will be flattened so it can be fed to the dense networks and the final output is a value between 0 (fake) and 1 (real).

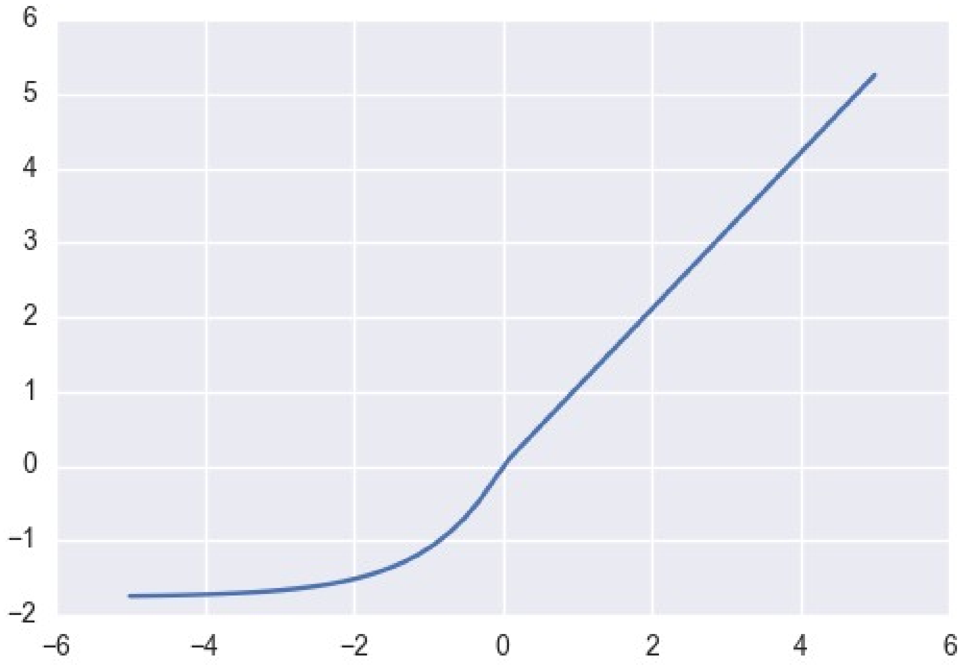
network

We can append these two models to build the GAN.

```
gan = keras.models.Sequential([generator,
discriminator])
```

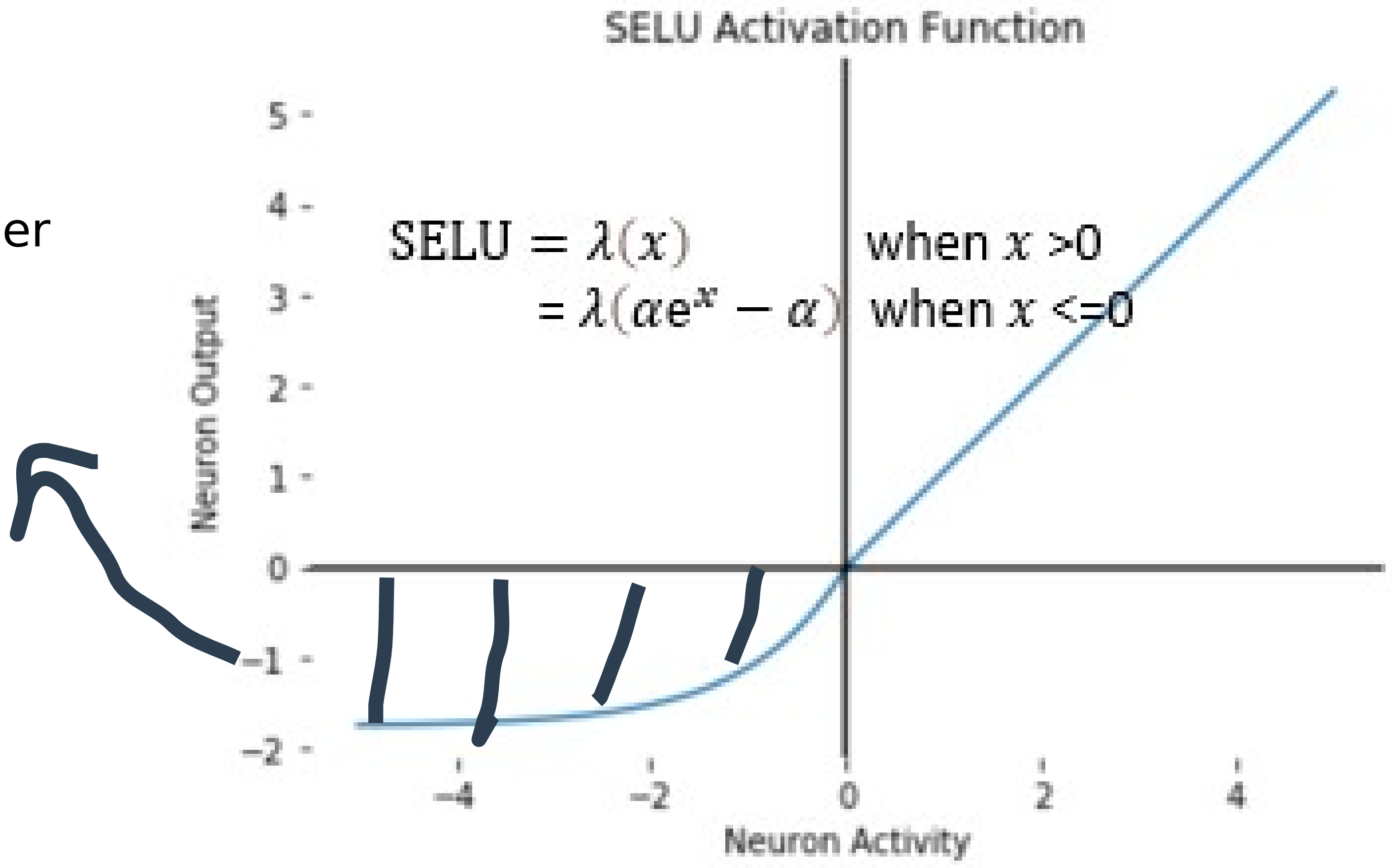
Important note

GAN is usually trained with activation function SELU not RELU



SElu

Remember we need noise



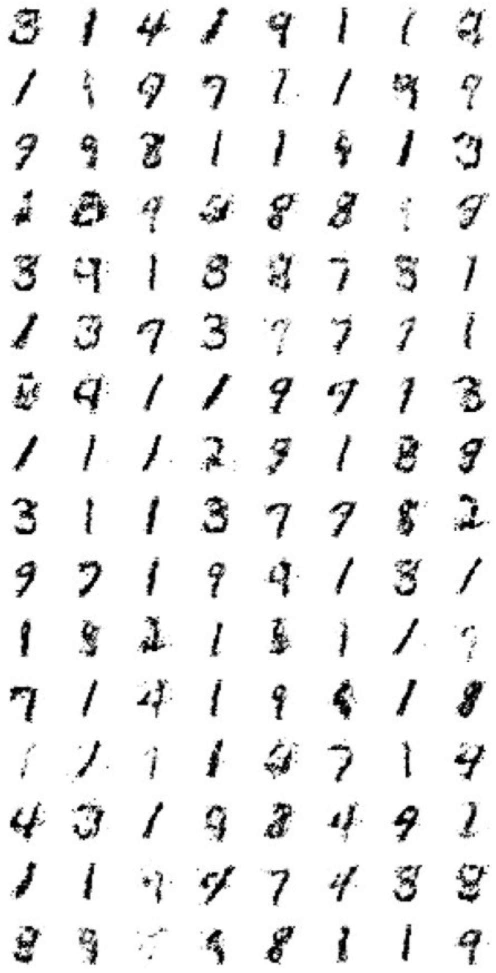
$$\mathrm{SELU} = \lambda(x) \quad \text{when } x > 0$$
$$= \lambda(\alpha e^{x} - \alpha) \quad \text{when } x <= 0$$

Selu mathematical formula

Scaled Exponential Linear Units, or SELUs, are activation functions that induce self-normalizing properties. The SELU activation function is given by. f (x) = λ x if x ≥ 0 f (x) = λ α (exp with α ≈ 1.6733 and λ ≈ 1.0507

https://paperswithcode.com/method/selu

Gan output



DCGAN

DCGAN: stands for Deep convolutional Generative adversarial Network

DCGANS

Generator

For the generator, we take in random noise and eventually transform it to the shape of the Fashion MNIST images. The general steps are:

*** Feed the input noise to a dense layer.**

*** Reshape the output to have three dimensions. This stands for the (length, width, number of filters).**

*** Perform a deconvolution (with [Conv2DTranspose](https://www.tensorflow.org/api_docs/python/tf/keras/layers/Conv2DTranspose)), reducing the number of filters by half and using a stride of `2`.**

*** The final layer upsamples the features to the size of the training images. In this case 28 x 28 x 1.**

Notice that batch normalization is performed except for the final deconvolution layer. As best practice, `selu` is the activation used for the intermediate deconvolution while `tanh` is for the output. We printed the model summary so you can see the shapes at each layer.

DCGANS

Discriminator

The discriminator will use strided convolutions to reduce the dimensionality of the input images. As best practice, these are activated by LeakyRELU. The output features will be flattened and fed to a 1-unit dense layer activated by sigmoid.

Notes to be considered

Generator part :

Uses Conv2DTranspose followed by batch normalization

All activations should be SELU except for the last layer which is TanH

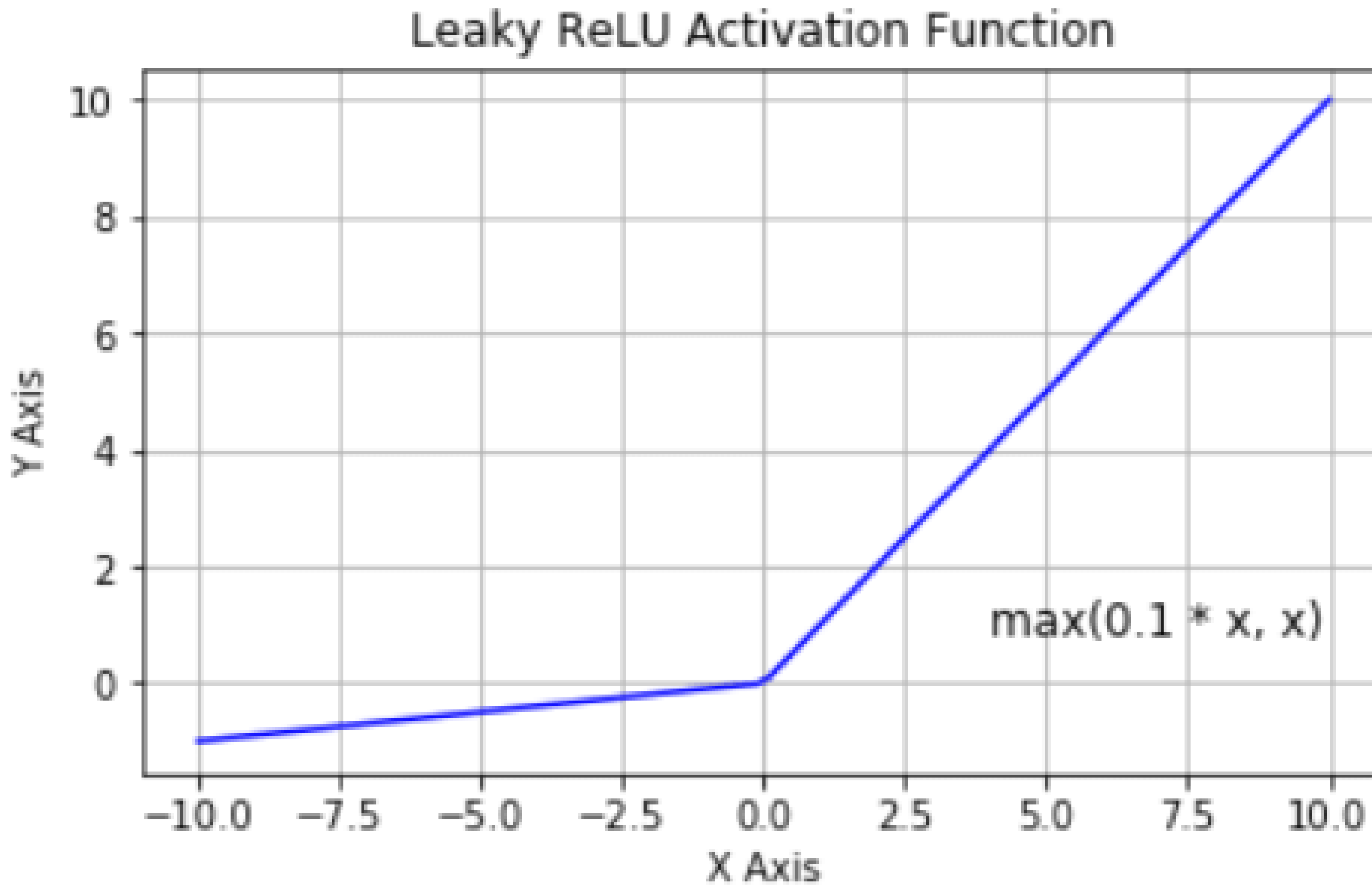
Notes to be considered

Discriminator

No hidden layers as Dense layers all are conv2d except for the output

Activation function is leakyRelu

Remember leakyRelu



Case study handsigns generation

Get the training data

Preprocess the images

Build the generator

Generator configuartion

Dense: number of units should equal 7 * 7 * 128, input_shape takes in a list containing the random normal dimensions

Case study handsigns generation

random_normal_dimensions is a hyperparameter that defines how many random numbers in a vector you'll want to feed into the generator as a starting point for generating images.

Reshape: reshape the vector to a 7 x 7 x 128 tensor.

BatchNormalization

Conv2DTranspose: takes 64 units, kernel size is 5, strides is 2, padding is SAME, activation is selu.

BatchNormalization

Conv2DTranspose: 1 unit, kernel size is 5, strides is 2, padding is SAME, and activation is tanh

Configurations

Build the discriminator

Conv2D: 64 units, kernel size of 5, strides of 2, padding is SAME, activation is a leaky relu with alpha of 0.2, input shape is 28 x 28 x 1

Dropout: rate is 0.4 (fraction of input units to drop)

Conv2D: 128 units, kernel size of 5, strides of 2, padding is SAME, activation is LeakyRelu with alpha of 0.2

Dropout: rate is 0.4.

Flatten

Dense: with 1 unit and a sigmoid activation

Train the GAN (train discriminator)

Phase 1

real_batch_size: Get the batch size of the input batch (it's the zero-th dimension of the tensor)

noise: Generate the noise using tf.random.normal. The shape is batch size x random_normal_dimension

fake images: Use the generator that you just created. Pass in the noise and produce fake images.

mixed_images: concatenate the fake images with the real images.

Set the axis to 0.

discriminator_labels: Set to 0. for real images and 1. for fake images.

Set the discriminator as trainable.

Use the discriminator's train_on_batch() method to train on the mixed images and the discriminator labels.

Train phase 2 (generator training)

Phase 2

noise: generate random normal values with dimensions batch_size x random_normal_dimensions

Use real_batch_size.

Generator_labels: Set to 1. to mark the fake images as real

The generator will generate fake images that are labeled as real images and attempt to fool the discriminator.

Set the discriminator to NOT be trainable.

Train the GAN on the noise and the generator labels.