How to Develop a Pix2Pix GAN for Image-to-Image Translation

by **Jason Brownlee** on August 2, 2019 in **Generative Adversarial Networks**

Last Updated on January 18, 2021

The Pix2Pix Generative Adversarial Network, or GAN, is an approach to training a deep convolutional neural network for image-to-image translation tasks.

The careful configuration of architecture as a type of image-conditional GAN allows for both the generation of large images compared to prior GAN models (e.g. such as 256×256 pixels) and the capability of performing well on a variety of different image-to-image translation tasks.

In this tutorial, you will discover how to develop a Pix2Pix generative adversarial network for image-to-image translation.

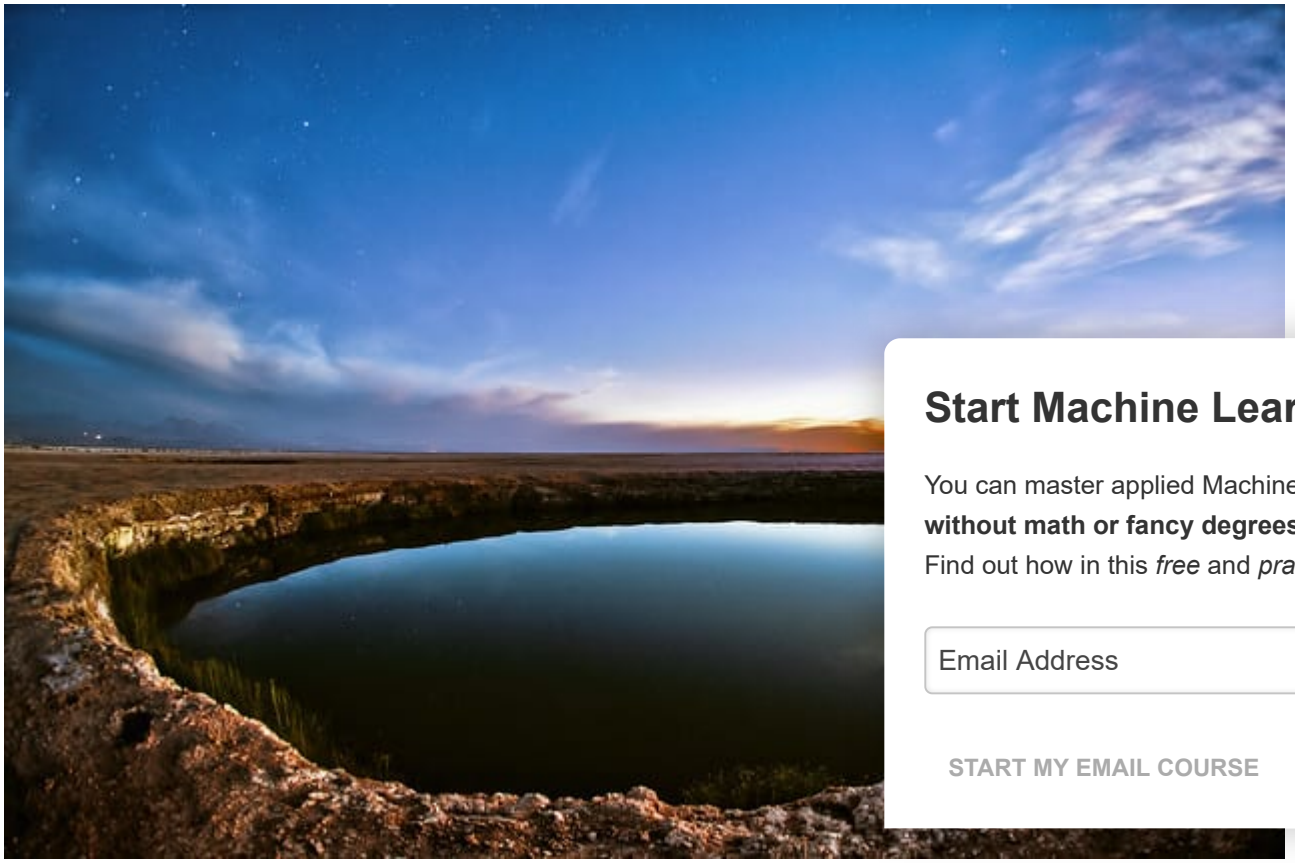
After completing this tutorial, you will know:

- How to load and prepare the satellite image to Google maps image-to-image translation dataset.
- How to develop a Pix2Pix model for translating satellite photographs to Google map images.
- How to use the final Pix2Pix generator model to translate ad hoc satellite images.

Kick-start your project with my new book Generative Adversarial Networks with Python, including *step-by-step tutorials* and the *Python source code* files for all examples.

Let's get started.

- **Updated Jan/2021**: Updated so layer freezing works with batch norm.



How to Develop a Pix2Pix Generative Adversarial Network for Image-to-Image Translation
Photo by European Southern Observatory, some rights reserved.

Tutorial Overview

This tutorial is divided into five parts; they are:

1. What Is the Pix2Pix GAN?
2. Satellite to Map Image Translation Dataset
3. How to Develop and Train a Pix2Pix Model

Start Machine Learning

4. How to Translate Images With a Pix2Pix Model
5. How to Translate Google Maps to Satellite Images

What Is the Pix2Pix GAN?

Pix2Pix is a Generative Adversarial Network, or GAN, model designed for general purpose image-to-image translation.

The approach was presented by Phillip Isola, et al. in their 2016 paper titled "Image-to-Image Translation with Conditional Adversarial Networks" and presented at CVPR in 2017.

The GAN architecture is comprised of a generator model for outputting new plausible synthetic images, and a discriminator model that classifies images as real (from the dataset) or fake (generated). The discriminator model is updated directly, whereas the generator model is updated via the discriminator model. As such, the two models are trained simultaneously i ks to better fool the discriminator and the discriminator seeks to better identify the counterfeit

The Pix2Pix model is a type of conditional GAN, or cGAN, where the generation of the out a source image. The discriminator is provided both with a source image and the target imag plausible transformation of the source image.

The generator is trained via adversarial loss, which encourages the generator to generate ator is also updated via L1 loss measured between the generated image and the expected outp generator model to create plausible translations of the source image.

The Pix2Pix GAN has been demonstrated on a range of image-to-image translation tasks such as converting maps to satellite photographs, black and white photographs to color, and sketches of products to product photographs.

Now that we are familiar with the Pix2Pix GAN, let's prepare a dataset that we can use with image-to-image translation.

Satellite to Map Image Translation Dataset

In this tutorial, we will use the so-called "*maps*" dataset used in the Pix2Pix paper.

This is a dataset comprised of satellite images of New York and their corresponding Google maps pages. The image translation problem involves converting satellite photos to Google maps format, or the reverse, Google maps images to Satellite photos.

The dataset is provided on the pix2pix website and can be downloaded as a 255-megabyt

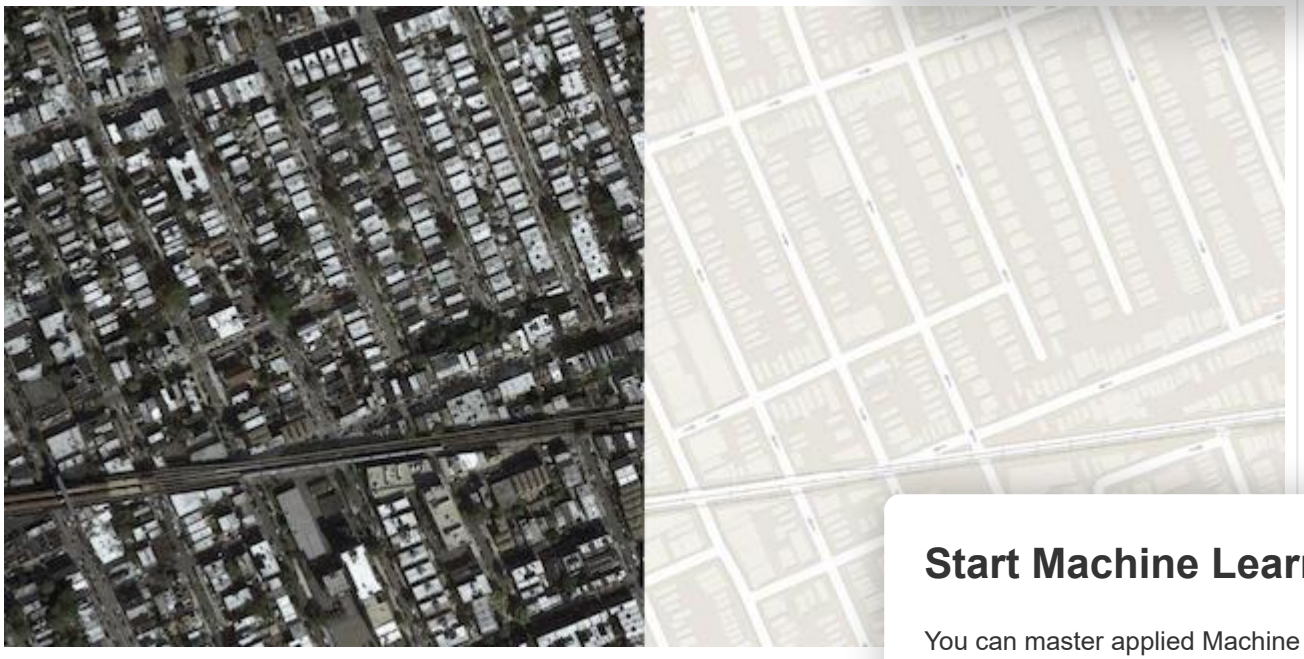
- Download Maps Dataset (maps.tar.gz)

Download the dataset and unzip it into your current working directory. This will create a dir

```
maps
├── train
└── val
```

The train folder contains 1,097 images, whereas the validation dataset contains 1,099 ima

Images have a digit filename and are in JPEG format. Each image is 1,200 pixels wide an ge on the left and the Google maps image on the right.



Sample Image From the Maps Dataset Including Both Satellite and

We can prepare this dataset for training a Pix2Pix GAN model in Keras. We will just work w e will be loaded, rescaled, and split into the satellite and Google map elements. The result w height of 256×256 pixels.

The *load_images()* function below implements this. It enumerates the list of images in a gi 256×512 pixels, splits each image into satellite and map elements and returns an array of

```
# load all images in a directory into memory
def load_images(path, size=(256,512)):
    src_list, tar_list = list(), list()
    # enumerate filenames in directory, assume all are images
    for filename in listdir(path):
        # load and resize the image
        pixels = load_img(path + filename, target_size=size)
        # convert to numpy array
        pixels = img_to_array(pixels)
        # split into satellite and map
        sat_img, map_img = pixels[:, :256], pixels[:, 256:]
        src_list.append(sat_img)
        tar_list.append(map_img)
    return [asarray(src_list), asarray(tar_list)]
```

We can call this function with the path to the training dataset. Once loaded, we can save the prepared arrays to a new file in compressed format for later use.

The complete example is listed below.

```
# load, split and scale the maps dataset ready for training
from os import listdir
from numpy import asarray
from numpy import vstack
from keras.preprocessing.image import img_to_array
from keras.preprocessing.image import load_img
from numpy import savez_compressed

# load all images in a directory into memory
def load_images(path, size=(256,512)):
    src_list, tar_list = list(), list()
    # enumerate filenames in directory, assume all are images
    for filename in listdir(path):
        # load and resize the image
        pixels = load_img(path + filename, target_size=size)
        # convert to numpy array
        pixels = img_to_array(pixels)
        # split into satellite and map
        sat_img, map_img = pixels[:, :256], pixels[:, 256:]
        src_list.append(sat_img)
        tar_list.append(map_img)
    return [asarray(src_list), asarray(tar_list)]

# dataset path
path = 'maps/train/'
# load dataset
[src_images, tar_images] = load_images(path)
print('Loaded: ', src_images.shape, tar_images.shape)
# save as compressed numpy array
filename = 'maps_256.npz'
savez_compressed(filename, src_images, tar_images)
print('Saved dataset: ', filename)
```


Running the example loads all images in the training dataset, summarizes their shape to ensure the images were loaded correctly, then saves the arrays to a new file called *maps_256.npz* in compressed NumPy array format.

```
Loaded:  (1096, 256, 256, 3) (1096, 256, 256, 3)
Saved dataset:  maps_256.npz
```

This file can be loaded later via the load() NumPy function and retrieving each array in turn

We can then plot some images pairs to confirm the data has been handled correctly.

```
# load the prepared dataset
from numpy import load
from matplotlib import pyplot
# load the dataset
data = load('maps_256.npz')
src_images, tar_images = data['arr_0'], data['arr_1']
print('Loaded: ', src_images.shape, tar_images.shape)
# plot source images
n_samples = 3
for i in range(n_samples):
    pyplot.subplot(2, n_samples, 1 + i)
    pyplot.axis('off')
    pyplot.imshow(src_images[i].astype('uint8'))
# plot target image
for i in range(n_samples):
    pyplot.subplot(2, n_samples, 1 + n_samples + i)
    pyplot.axis('off')
    pyplot.imshow(tar_images[i].astype('uint8'))
pyplot.show()
```

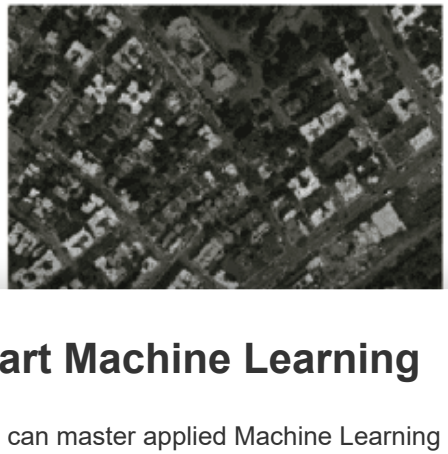
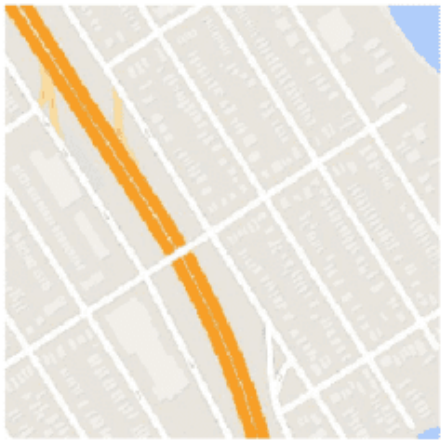
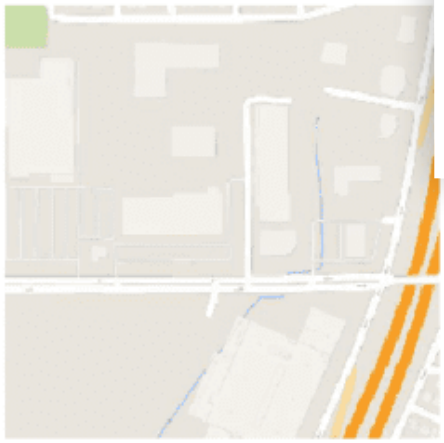
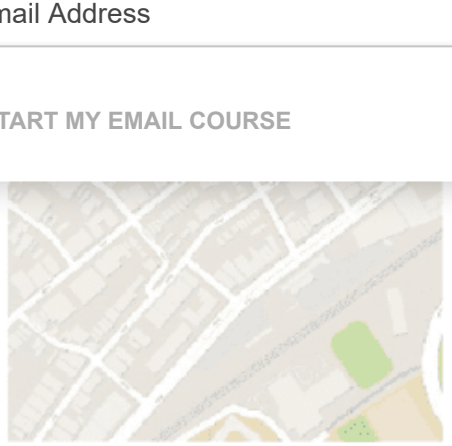
Running this example loads the prepared dataset and summarizes the shape of each arra
thousand 256×256 image pairs.

```
Loaded: (1096, 256, 256, 3) (1096, 256, 256, 3)
```

A plot of three image pairs is also created showing the satellite images on the top and Goo

We can see that satellite images are quite complex and that although the Google map ima
things like major roads, water, and parks.



Plot of Three Image Pairs Showing Satellite Images (top) and Google Map Images (bottom).

Now that we have prepared the dataset for image translation, we can develop our Pix2Pix GAN model.

How to Develop and Train a Pix2Pix Model

In this section, we will develop the Pix2Pix model for translating satellite photos to Google maps images.

The same model architecture and configuration described in the paper was used across a range of image translation tasks. This architecture is both described in the body of the paper, with additional detail in the appendix of the paper, and a fully working implementation provided as open source with the Torch deep learning framework.

The implementation in this section will use the Keras deep learning framework based direc
implemented in the author's code base, designed to take and generate color images with t

The architecture is comprised of two models: the discriminator and the generator.

The discriminator is a deep convolutional neural network that performs image classificatio
takes both the source image (e.g. satellite photo) and the target image (e.g. Google maps
whether target image is real or a fake translation of the source image.

The discriminator design is based on the effective receptive field of the model, which defin
to the number of pixels in the input image. This is called a PatchGAN model and is careful
model maps to a 70×70 square or patch of the input image. The benefit of this approach is
of different sizes, e.g. larger or smaller than 256×256 pixels.

The output of the model depends on the size of the input image but may be one value or a square activation map of values. Each value is a probability for the likelihood that a patch in the input image is real. These values can be averaged to give an overall likelihood or classification score if needed.

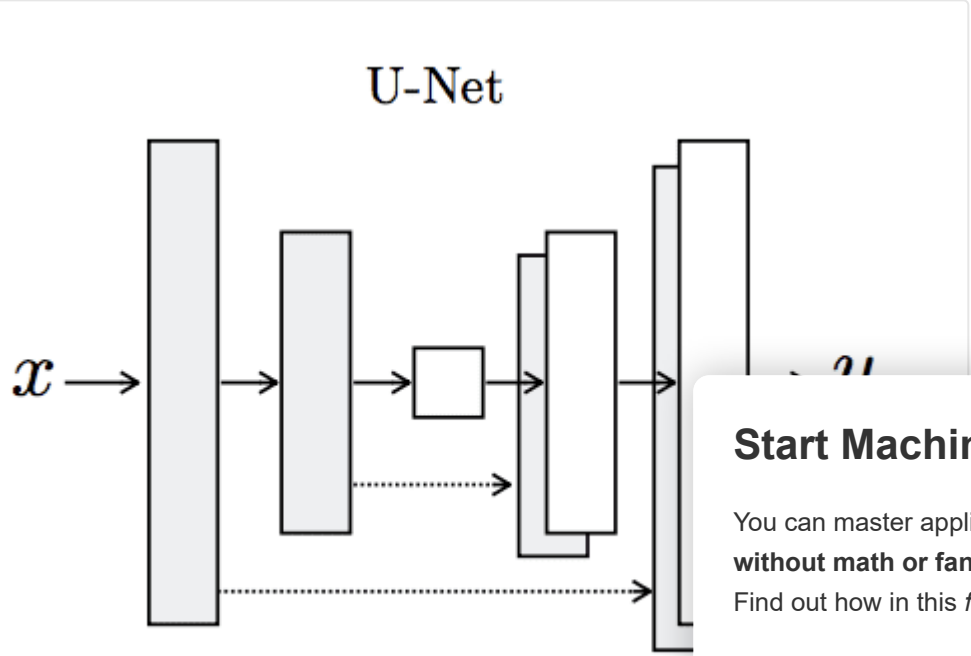
The *define_discriminator()* function below implements the 70×70 PatchGAN discriminator model as per the design of the model in the paper. The model takes two input images that are concatenated together and predicts a patch output of predictions. The model is optimized using binary cross entropy, and a weighting is used so that updates to the model have half (0.5) the usual effect. The authors of Pix2Pix recommend this weighting of model updates to slow down changes to the discriminator, relative to the

```python
# define the discriminator model
def define_discriminator(image_shape):
    # weight initialization
    init = RandomNormal(stddev=0.02)
    # source image input
    in_src_image = Input(shape=image_shape)
    # target image input
    in_target_image = Input(shape=image_shape)
    # concatenate images channel-wise
    merged = Concatenate()([in_src_image, in_target_image])
    # C64
    d = Conv2D(64, (4,4), strides=(2,2), padding='same', kernel_initializer=init)(merged)
    d = LeakyReLU(alpha=0.2)(d)
    # C128
    d = Conv2D(128, (4,4), strides=(2,2), padding='same', kernel_initializer=init)(d)
    d = BatchNormalization()(d)
    d = LeakyReLU(alpha=0.2)(d)
    # C256
    d = Conv2D(256, (4,4), strides=(2,2), padding='same', kernel_initializer=ir
    d = BatchNormalization()(d)
    d = LeakyReLU(alpha=0.2)(d)
    # C512
    d = Conv2D(512, (4,4), strides=(2,2), padding='same', kernel_initializer=ir
    d = BatchNormalization()(d)
    d = LeakyReLU(alpha=0.2)(d)
    # second last output layer
    d = Conv2D(512, (4,4), padding='same', kernel_initializer=init)(d)
    d = BatchNormalization()(d)
    d = LeakyReLU(alpha=0.2)(d)
    # patch output
    d = Conv2D(1, (4,4), padding='same', kernel_initializer=init)(d)
    patch_out = Activation('sigmoid')(d)
    # define model
    model = Model([in_src_image, in_target_image], patch_out)
    # compile model
    opt = Adam(lr=0.0002, beta_1=0.5)
    model.compile(loss='binary_crossentropy', optimizer=opt, loss_weights=[0.5])
    return model
```


The generator model is more complex than the discriminator model.

The generator is an encoder-decoder model using a U-Net architecture. The model takes a source image (e.g. satellite photo) and generates a target image (e.g. Google maps image). It does this by first downsampling or encoding the input image down to a bottleneck layer, then upsampling or decoding the bottleneck representation to the size of the output image. The U-Net architecture means that skip-connections are added between the encoding layers and the corresponding decoding layers, forming a U-shape.

The image below makes the skip-connections clear, showing how the first layer of the encoder is connected to the last layer of the decoder, and so on.



Architecture of the U-Net Generator Model
Taken from Image-to-Image Translation With Conditional Adve

The encoder and decoder of the generator are comprised of standardized blocks of convolutional, batch normalization, dropout, and activation layers. This standardization means that we can develop helper functions to create each block of layers and call it repeatedly to build-up the encoder and decoder parts of the model.

The *define_generator()* function below implements the U-Net encoder-decoder generator model. It uses the *define_encoder_block()* helper function to create blocks of layers for the encoder and the *decoder_block()* function to create blocks of layers for the decoder. The tanh activation function is used in the output layer, meaning that pixel values in the generated image will be in the range [-1,1].

```
# define an encoder block
def define_encoder_block(layer_in, n_filters, batchnorm=True):
    # weight initialization
    init = RandomNormal(stddev=0.02)
    # add downsampling layer
```

```
    g = Conv2D(n_filters, (4,4), strides=(2,2), padding='same', kernel_initializer=init)(layer_in)
    # conditionally add batch normalization
    if batchnorm:
        g = BatchNormalization()(g, training=True)
    # leaky relu activation
    g = LeakyReLU(alpha=0.2)(g)
    return g

# define a decoder block
def decoder_block(layer_in, skip_in, n_filters, dropout=True):
    # weight initialization
    init = RandomNormal(stddev=0.02)
    # add upsampling layer
    g = Conv2DTranspose(n_filters, (4,4), strides=(2,2), padding='same', kernel_initializer=init)(layer_in)
    # add batch normalization
    g = BatchNormalization()(g, training=True)
    # conditionally add dropout
    if dropout:
        g = Dropout(0.5)(g, training=True)
    # merge with skip connection
    g = Concatenate()([g, skip_in])
    # relu activation
    g = Activation('relu')(g)
    return g

# define the standalone generator model
def define_generator(image_shape=(256,256,3)):
    # weight initialization
    init = RandomNormal(stddev=0.02)
    # image input
    in_image = Input(shape=image_shape)
    # encoder model
    e1 = define_encoder_block(in_image, 64, batchnorm=False)
    e2 = define_encoder_block(e1, 128)
    e3 = define_encoder_block(e2, 256)
    e4 = define_encoder_block(e3, 512)
    e5 = define_encoder_block(e4, 512)
    e6 = define_encoder_block(e5, 512)
    e7 = define_encoder_block(e6, 512)
    # bottleneck, no batch norm and relu
    b = Conv2D(512, (4,4), strides=(2,2), padding='same', kernel_initializer=init)(e7)
    b = Activation('relu')(b)
    # decoder model
    d1 = decoder_block(b, e7, 512)
    d2 = decoder_block(d1, e6, 512)
    d3 = decoder_block(d2, e5, 512)
    d4 = decoder_block(d3, e4, 512, dropout=False)
    d5 = decoder_block(d4, e3, 256, dropout=False)
    d6 = decoder_block(d5, e2, 128, dropout=False)
```

```
    d7 = decoder_block(d6, e1, 64, dropout=False)
    # output
    g = Conv2DTranspose(3, (4,4), strides=(2,2), padding='same', kernel_initializer=init)(d7)
    out_image = Activation('tanh')(g)
    # define model
    model = Model(in_image, out_image)
    return model
```

The discriminator model is trained directly on real and generated images, whereas the generator model is not.

Instead, the generator model is trained via the discriminator model. It is updated to minimize the loss predicted by the discriminator for generated images marked as "*real*." As such, it is encouraged to generate more real images. The generator is also updated to minimize the L1 loss or mean absolute error between the generated image and the target image.

The generator is updated via a weighted sum of both the adversarial loss and the L1 loss, w
weighting of 100 to 1 in favor of the L1 loss. This is to encourage the generator strongly to
image, and not just plausible images in the target domain.

This can be achieved by defining a new logical model comprised of the weights in the exis
This logical or composite model involves stacking the generator on top of the discriminator
generator and to the discriminator, although the output of the generator is connected to the
The discriminator then predicts the likelihood that the generator was a real translation of th

The discriminator is updated in a standalone manner, so the weights are reused in this cor
composite model is updated with two targets, one indicating that the generated images we
updates in the generator toward generating more realistic images, and the executed real t
the output of the generator model (L1 loss).

The *define_gan()* function below implements this, taking the already-defined generator and discriminator models as arguments and using the Keras functional API to connect them together into a composite model. Both loss functions are specified for the two outputs of the model and the weights used for each are specified in the *loss_weights* argument to the *compile()* function.

```
# define the combined generator and discriminator model, for updating the generator
def define_gan(g_model, d_model, image_shape):
    # make weights in the discriminator not trainable
    for layer in d_model.layers:
        if not isinstance(layer, BatchNormalization):
            layer.trainable = False
    # define the source image
```

```
    in_src = Input(shape=image_shape)
    # connect the source image to the generator input
    gen_out = g_model(in_src)
    # connect the source input and generator output to the discriminator input
    dis_out = d_model([in_src, gen_out])
    # src image as input, generated image and classification output
    model = Model(in_src, [dis_out, gen_out])
    # compile model
    opt = Adam(lr=0.0002, beta_1=0.5)
    model.compile(loss=['binary_crossentropy', 'mae'], optimizer=opt, loss_weights=[1,100])
    return model
```

Next, we can load our paired images dataset in compressed NumPy array format.

This will return a list of two NumPy arrays: the first for source images and the second for corresponding target images.

```
# load and prepare training images
def load_real_samples(filename):
    # load compressed arrays
    data = load(filename)
    # unpack arrays
    X1, X2 = data['arr_0'], data['arr_1']
    # scale from [0,255] to [-1,1]
    X1 = (X1 - 127.5) / 127.5
    X2 = (X2 - 127.5) / 127.5
    return [X1, X2]
```


Training the discriminator will require batches of real and fake images.

The *generate_real_samples()* function below will prepare a batch of random pairs of image g discriminator label of *class=1* to indicate they are real.

```
# select a batch of random samples, returns images and target
def generate_real_samples(dataset, n_samples, patch_shape):
    # unpack dataset
    trainA, trainB = dataset
    # choose random instances
    ix = randint(0, trainA.shape[0], n_samples)
    # retrieve selected images
    X1, X2 = trainA[ix], trainB[ix]
    # generate 'real' class labels (1)
    y = ones((n_samples, patch_shape, patch_shape, 1))
    return [X1, X2], y
```

The *generate_fake_samples()* function below uses the generator model and a batch of real source images to generate an equivalent batch of target images for the discriminator.

These are returned with the label class-0 to indicate to the discriminator that they are fake.

```
# generate a batch of images, returns images and targets
def generate_fake_samples(g_model, samples, patch_shape):
    # generate fake instance
    X = g_model.predict(samples)
    # create 'fake' class labels (0)
    y = zeros((len(X), patch_shape, patch_shape, 1))
    return X, y
```

Typically, GAN models do not converge; instead, an equilibrium is found between the generator and discriminator models. As such, we cannot easily judge when training should stop. Therefore, we can save the model and use it to ge
periodically during training, such as every 10 training epochs.

We can then review the generated images at the end of training and use the image quality

The *summarize_performance()* function implements this, taking the generator model at a p
number, in this case three, of translations of randomly selected images in the dataset. The
then plotted as three rows of images and the plot saved to file. Additionally, the model is sa
load later.

Both the image and model filenames include the training iteration number, allowing us to e

```
# generate samples and save as a plot and save the model
def summarize_performance(step, g_model, dataset, n_samples=3):
    # select a sample of input images
    [X_realA, X_realB], _ = generate_real_samples(dataset, n_samples, 1)
    # generate a batch of fake samples
    X_fakeB, _ = generate_fake_samples(g_model, X_realA, 1)
    # scale all pixels from [-1,1] to [0,1]
    X_realA = (X_realA + 1) / 2.0
    X_realB = (X_realB + 1) / 2.0
    X_fakeB = (X_fakeB + 1) / 2.0
    # plot real source images
    for i in range(n_samples):
        pyplot.subplot(3, n_samples, 1 + i)
        pyplot.axis('off')
        pyplot.imshow(X_realA[i])
    # plot generated target image
```

```
    for i in range(n_samples):
        pyplot.subplot(3, n_samples, 1 + n_samples + i)
        pyplot.axis('off')
        pyplot.imshow(X_fakeB[i])
    # plot real target image
    for i in range(n_samples):
        pyplot.subplot(3, n_samples, 1 + n_samples*2 + i)
        pyplot.axis('off')
        pyplot.imshow(X_realB[i])
    # save plot to file
    filename1 = 'plot_%06d.png' % (step+1)
    pyplot.savefig(filename1)
    pyplot.close()
    # save the generator model
    filename2 = 'model_%06d.h5' % (step+1)
    g_model.save(filename2)
    print('>Saved: %s and %s' % (filename1, filename2))
```

Finally, we can train the generator and discriminator models.

The *train()* function below implements this, taking the defined generator, discriminator, com
number of epochs is set at 100 to keep training times down, although 200 was used in the
in the paper.

Training involves a fixed number of training iterations. There are 1,097 images in the traini
number of examples, with a batch size of one means 1,097 training steps. The generator i
10,970 training steps, and the model will run for 100 epochs, or a total of 109,700 training

Each training step involves first selecting a batch of real examples, then using the generat
using the real source images. The discriminator is then updated with the batch of real images and then fake images.

Next, the generator model is updated providing the real source images as input and providing class labels of 1 (real) and the real target images as the expected outputs of the model required for calculating loss. The generator has two loss scores as well as the weighted sum score returned from the call to *train_on_batch()*. We are only interested in the weighted sum score (the first value returned) as it is used to update the model weights.

Finally, the loss for each update is reported to the console each training iteration and model performance is evaluated every 10 training epochs.

```
# train pix2pix model
def train(d_model, g_model, gan_model, dataset, n_epochs=100, n_batch=1):
    # determine the output square shape of the discriminator
```

```
    n_patch = d_model.output_shape[1]
    # unpack dataset
    trainA, trainB = dataset
    # calculate the number of batches per training epoch
    bat_per_epo = int(len(trainA) / n_batch)
    # calculate the number of training iterations
    n_steps = bat_per_epo * n_epochs
    # manually enumerate epochs
    for i in range(n_steps):
        # select a batch of real samples
        [X_realA, X_realB], y_real = generate_real_samples(dataset, n_batch, n_patch)
        # generate a batch of fake samples
        X_fakeB, y_fake = generate_fake_samples(g_model, X_realA, n_patch)
        # update discriminator for real samples
        d_loss1 = d_model.train_on_batch([X_realA, X_realB], y_real)
        # update discriminator for generated samples
        d_loss2 = d_model.train_on_batch([X_realA, X_fakeB], y_fake)
        # update the generator
        g_loss, _, _ = gan_model.train_on_batch(X_realA, [y_real, X_realB])
        # summarize performance
        print('>%d, d1[%.3f] d2[%.3f] g[%.3f]' % (i+1, d_loss1, d_loss2, g_loss
        # summarize model performance
        if (i+1) % (bat_per_epo * 10) == 0:
            summarize_performance(i, g_model, dataset)
```

Tying all of this together, the complete code example of training a Pix2Pix GAN to translate
below.

```
# example of pix2pix gan for satellite to map image-to-image translation
from numpy import load
from numpy import zeros
from numpy import ones
from numpy.random import randint
from keras.optimizers import Adam
from keras.initializers import RandomNormal
from keras.models import Model
from keras.models import Input
from keras.layers import Conv2D
from keras.layers import Conv2DTranspose
from keras.layers import LeakyReLU
from keras.layers import Activation
from keras.layers import Concatenate
from keras.layers import Dropout
from keras.layers import BatchNormalization
from keras.layers import LeakyReLU
from matplotlib import pyplot

# define the discriminator model
```

```
def define_discriminator(image_shape):
    # weight initialization
    init = RandomNormal(stddev=0.02)
    # source image input
    in_src_image = Input(shape=image_shape)
    # target image input
    in_target_image = Input(shape=image_shape)
    # concatenate images channel-wise
    merged = Concatenate()([in_src_image, in_target_image])
    # C64
    d = Conv2D(64, (4,4), strides=(2,2), padding='same', kernel_initializer=init)(merged)
    d = LeakyReLU(alpha=0.2)(d)
    # C128
    d = Conv2D(128, (4,4), strides=(2,2), padding='same', kernel_initializer=init)(d)
    d = BatchNormalization()(d)
    d = LeakyReLU(alpha=0.2)(d)
    # C256
    d = Conv2D(256, (4,4), strides=(2,2), padding='same', kernel_initializer=i
    d = BatchNormalization()(d)
    d = LeakyReLU(alpha=0.2)(d)
    # C512
    d = Conv2D(512, (4,4), strides=(2,2), padding='same', kernel_initializer=i
    d = BatchNormalization()(d)
    d = LeakyReLU(alpha=0.2)(d)
    # second last output layer
    d = Conv2D(512, (4,4), padding='same', kernel_initializer=init)(d)
    d = BatchNormalization()(d)
    d = LeakyReLU(alpha=0.2)(d)
    # patch output
    d = Conv2D(1, (4,4), padding='same', kernel_initializer=init)(d)
    patch_out = Activation('sigmoid')(d)
    # define model
    model = Model([in_src_image, in_target_image], patch_out)
    # compile model
    opt = Adam(lr=0.0002, beta_1=0.5)
    model.compile(loss='binary_crossentropy', optimizer=opt, loss_weights=[0.5])
    return model

# define an encoder block
def define_encoder_block(layer_in, n_filters, batchnorm=True):
    # weight initialization
    init = RandomNormal(stddev=0.02)
    # add downsampling layer
    g = Conv2D(n_filters, (4,4), strides=(2,2), padding='same', kernel_initializer=init)(layer_in)
    # conditionally add batch normalization
    if batchnorm:
        g = BatchNormalization()(g, training=True)
    # leaky relu activation
    g = LeakyReLU(alpha=0.2)(g)
```

```
    return g

# define a decoder block
def decoder_block(layer_in, skip_in, n_filters, dropout=True):
    # weight initialization
    init = RandomNormal(stddev=0.02)
    # add upsampling layer
    g = Conv2DTranspose(n_filters, (4,4), strides=(2,2), padding='same', kernel_initializer=init)(layer_in)
    # add batch normalization
    g = BatchNormalization()(g, training=True)
    # conditionally add dropout
    if dropout:
        g = Dropout(0.5)(g, training=True)
    # merge with skip connection
    g = Concatenate()([g, skip_in])
    # relu activation
    g = Activation('relu')(g)
    return g

# define the standalone generator model
def define_generator(image_shape=(256,256,3)):
    # weight initialization
    init = RandomNormal(stddev=0.02)
    # image input
    in_image = Input(shape=image_shape)
    # encoder model
    e1 = define_encoder_block(in_image, 64, batchnorm=False)
    e2 = define_encoder_block(e1, 128)
    e3 = define_encoder_block(e2, 256)
    e4 = define_encoder_block(e3, 512)
    e5 = define_encoder_block(e4, 512)
    e6 = define_encoder_block(e5, 512)
    e7 = define_encoder_block(e6, 512)
    # bottleneck, no batch norm and relu
    b = Conv2D(512, (4,4), strides=(2,2), padding='same', kernel_initializer=init)(e7)
    b = Activation('relu')(b)
    # decoder model
    d1 = decoder_block(b, e7, 512)
    d2 = decoder_block(d1, e6, 512)
    d3 = decoder_block(d2, e5, 512)
    d4 = decoder_block(d3, e4, 512, dropout=False)
    d5 = decoder_block(d4, e3, 256, dropout=False)
    d6 = decoder_block(d5, e2, 128, dropout=False)
    d7 = decoder_block(d6, e1, 64, dropout=False)
    # output
    g = Conv2DTranspose(3, (4,4), strides=(2,2), padding='same', kernel_initializer=init)(d7)
    out_image = Activation('tanh')(g)
    # define model
    model = Model(in_image, out_image)
```

```
    return model

# define the combined generator and discriminator model, for updating the generator
def define_gan(g_model, d_model, image_shape):
    # make weights in the discriminator not trainable
    for layer in d_model.layers:
        if not isinstance(layer, BatchNormalization):
            layer.trainable = False
    # define the source image
    in_src = Input(shape=image_shape)
    # connect the source image to the generator input
    gen_out = g_model(in_src)
    # connect the source input and generator output to the discriminator input
    dis_out = d_model([in_src, gen_out])
    # src image as input, generated image and classification output
    model = Model(in_src, [dis_out, gen_out])
    # compile model
    opt = Adam(lr=0.0002, beta_1=0.5)
    model.compile(loss=['binary_crossentropy', 'mae'], optimizer=opt, loss_wei
    return model

# load and prepare training images
def load_real_samples(filename):
    # load compressed arrays
    data = load(filename)
    # unpack arrays
    X1, X2 = data['arr_0'], data['arr_1']
    # scale from [0,255] to [-1,1]
    X1 = (X1 - 127.5) / 127.5
    X2 = (X2 - 127.5) / 127.5
    return [X1, X2]

# select a batch of random samples, returns images and target
def generate_real_samples(dataset, n_samples, patch_shape):
    # unpack dataset
    trainA, trainB = dataset
    # choose random instances
    ix = randint(0, trainA.shape[0], n_samples)
    # retrieve selected images
    X1, X2 = trainA[ix], trainB[ix]
    # generate 'real' class labels (1)
    y = ones((n_samples, patch_shape, patch_shape, 1))
    return [X1, X2], y

# generate a batch of images, returns images and targets
def generate_fake_samples(g_model, samples, patch_shape):
    # generate fake instance
    X = g_model.predict(samples)
    # create 'fake' class labels (0)
```

```
    y = zeros((len(X), patch_shape, patch_shape, 1))
    return X, y

# generate samples and save as a plot and save the model
def summarize_performance(step, g_model, dataset, n_samples=3):
    # select a sample of input images
    [X_realA, X_realB], _ = generate_real_samples(dataset, n_samples, 1)
    # generate a batch of fake samples
    X_fakeB, _ = generate_fake_samples(g_model, X_realA, 1)
    # scale all pixels from [-1,1] to [0,1]
    X_realA = (X_realA + 1) / 2.0
    X_realB = (X_realB + 1) / 2.0
    X_fakeB = (X_fakeB + 1) / 2.0
    # plot real source images
    for i in range(n_samples):
        pyplot.subplot(3, n_samples, 1 + i)
        pyplot.axis('off')
        pyplot.imshow(X_realA[i])
    # plot generated target image
    for i in range(n_samples):
        pyplot.subplot(3, n_samples, 1 + n_samples + i)
        pyplot.axis('off')
        pyplot.imshow(X_fakeB[i])
    # plot real target image
    for i in range(n_samples):
        pyplot.subplot(3, n_samples, 1 + n_samples*2 + i)
        pyplot.axis('off')
        pyplot.imshow(X_realB[i])
    # save plot to file
    filename1 = 'plot_%06d.png' % (step+1)
    pyplot.savefig(filename1)
    pyplot.close()
    # save the generator model
    filename2 = 'model_%06d.h5' % (step+1)
    g_model.save(filename2)
    print('>Saved: %s and %s' % (filename1, filename2))

# train pix2pix models
def train(d_model, g_model, gan_model, dataset, n_epochs=100, n_batch=1):
    # determine the output square shape of the discriminator
    n_patch = d_model.output_shape[1]
    # unpack dataset
    trainA, trainB = dataset
    # calculate the number of batches per training epoch
    bat_per_epo = int(len(trainA) / n_batch)
    # calculate the number of training iterations
    n_steps = bat_per_epo * n_epochs
    # manually enumerate epochs
    for i in range(n_steps):
```

```
        # select a batch of real samples
        [X_realA, X_realB], y_real = generate_real_samples(dataset, n_batch, n_patch)
        # generate a batch of fake samples
        X_fakeB, y_fake = generate_fake_samples(g_model, X_realA, n_patch)
        # update discriminator for real samples
        d_loss1 = d_model.train_on_batch([X_realA, X_realB], y_real)
        # update discriminator for generated samples
        d_loss2 = d_model.train_on_batch([X_realA, X_fakeB], y_fake)
        # update the generator
        g_loss, _, _ = gan_model.train_on_batch(X_realA, [y_real, X_realB])
        # summarize performance
        print('>%d, d1[%.3f] d2[%.3f] g[%.3f]' % (i+1, d_loss1, d_loss2, g_loss))
        # summarize model performance
        if (i+1) % (bat_per_epo * 10) == 0:
            summarize_performance(i, g_model, dataset)

# load image data
dataset = load_real_samples('maps_256.npz')
print('Loaded', dataset[0].shape, dataset[1].shape)
# define input shape based on the loaded dataset
image_shape = dataset[0].shape[1:]
# define the models
d_model = define_discriminator(image_shape)
g_model = define_generator(image_shape)
# define the composite model
gan_model = define_gan(g_model, d_model, image_shape)
# train model
train(d_model, g_model, gan_model, dataset)
```


The example can be run on CPU hardware, although GPU hardware is recommended.

The example might take about two hours to run on modern GPU hardware.

Note: Your results may vary given the stochastic nature of the algorithm or evaluation procedure, or differences in numerical precision. Consider running the example a few times and compare the average outcome.

The loss is reported each training iteration, including the discriminator loss on real examples (d1), discriminator loss on generated or fake examples (d2), and generator loss, which is a weighted average of adversarial and L1 loss (g).

If loss for the discriminator goes to zero and stays there for a long time, consider re-starting the training run as it is an example of a training failure.

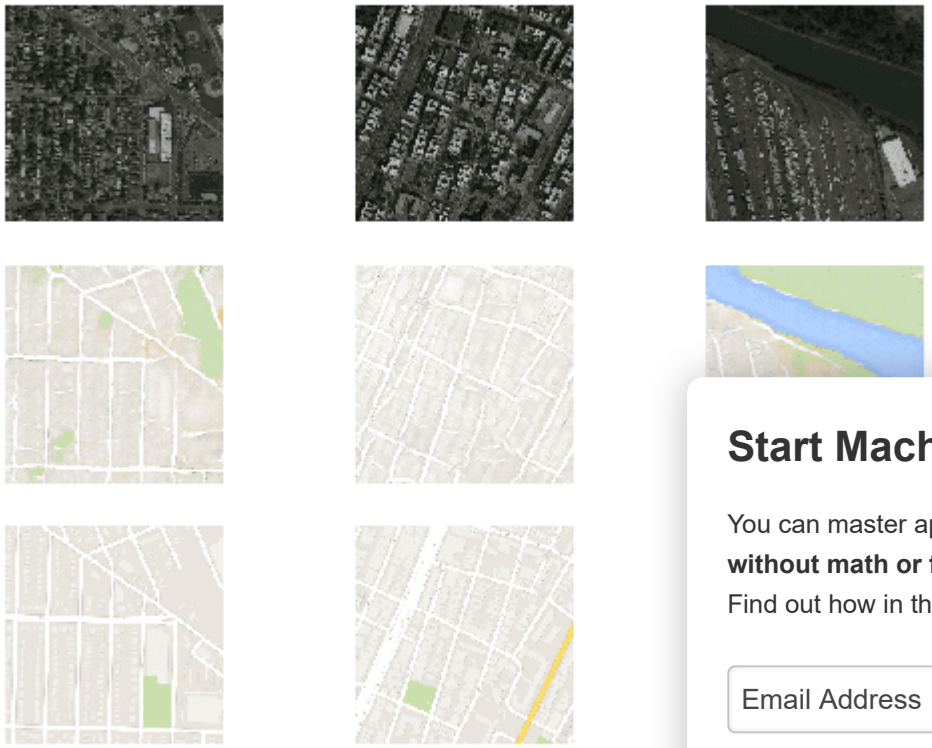
```
>1, d1[0.566] d2[0.520] g[82.266]
>2, d1[0.469] d2[0.484] g[66.813]
```

```
>3, d1[0.428] d2[0.477] g[79.520]
>4, d1[0.362] d2[0.405] g[78.143]
>5, d1[0.416] d2[0.406] g[72.452]
...
>109596, d1[0.303] d2[0.006] g[5.792]
>109597, d1[0.001] d2[1.127] g[14.343]
>109598, d1[0.000] d2[0.381] g[11.851]
>109599, d1[1.289] d2[0.547] g[6.901]
>109600, d1[0.437] d2[0.005] g[10.460]
>Saved: plot_109600.png and model_109600.h5
```

Models are saved every 10 epochs and saved to a file with the training iteration number. Additionally, images are generated every 10 epochs and compared to the expected target images. These plots can be assessed at the end of the run and used to select a final generator model based on generated image quality.

At the end of the run, will you will have 10 saved model files and 10 plots of generated ima

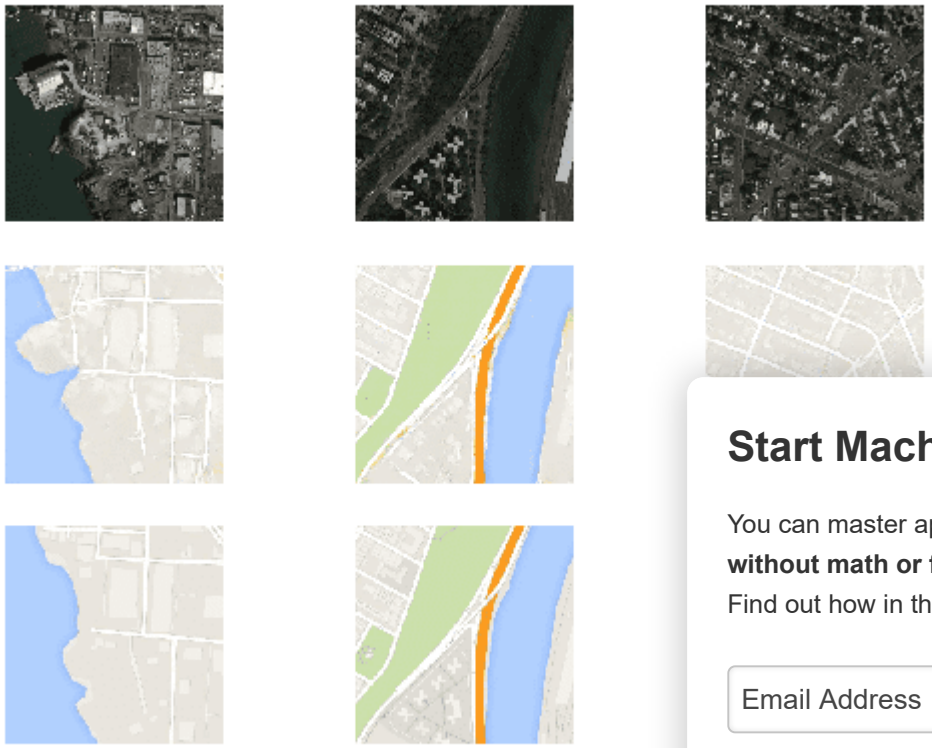
After the first 10 epochs, map images are generated that look plausible, although the lines contain some blurring. Nevertheless, large structures are in the right places with mostly the



Plot of Satellite to Google Map Translated Images Using Pix2Pix After 10 Training Epochs

Generated images after about 50 training epochs begin to look very realistic, at least to mean, and quality appears to remain good for the remainder of the training process.

Note the first generated image example below (right column, middle row) that includes more useful detail than the real Google map image.



Plot of Satellite to Google Map Translated Images Using Pix2Pix After 100 Training Epochs

Now that we have developed and trained the Pix2Pix model, we can explore how they can be used in a standalone manner.

How to Translate Images With a Pix2Pix Model

Training the Pix2Pix model results in many saved models and samples of generated images for each.

More training epochs does not necessarily mean a better quality model. Therefore, we can choose a model based on the quality of the generated images and use it to perform ad hoc image-to-image translation.

In this case, we will use the model saved at the end of the run, e.g. after 100 epochs or 109,600 training iterations.

A good starting point is to load the model and use it to make ad hoc translations of source images in the training dataset.

First, we can load the training dataset. We can use the same function named *load_real_samples()* for loading the dataset as was used when training the model.

```
# load and prepare training images
def load_real_samples(filename):
    # load compressed ararys
    data = load(filename)
    # unpack arrays
    X1, X2 = data['arr_0'], data['arr_1']
    # scale from [0,255] to [-1,1]
    X1 = (X1 - 127.5) / 127.5
    X2 = (X2 - 127.5) / 127.5
    return [X1, X2]
```

This function can be called as follows:

```
...
# load dataset
[X1, X2] = load_real_samples('maps_256.npz')
print('Loaded', X1.shape, X2.shape)
```

Next, we can load the saved Keras model.

```
...
# load model
model = load_model('model_109600.h5')
```

Next, we can choose a random image pair from the training dataset to use as an example.

```
...
# select random example
ix = randint(0, len(X1), 1)
src_image, tar_image = X1[ix], X2[ix]
```

We can provide the source satellite image as input to the model and use it to predict a Google map image.

```
...
# generate image from source
gen_image = model.predict(src_image)
```

Finally, we can plot the source, generated image, and the expected target image.

The *plot_images()* function below implements this, providing a nice title above each image.

```
# plot source, generated and target images
def plot_images(src_img, gen_img, tar_img):
    images = vstack((src_img, gen_img, tar_img))
    # scale from [-1,1] to [0,1]
    images = (images + 1) / 2.0
    titles = ['Source', 'Generated', 'Expected']
    # plot images row by row
    for i in range(len(images)):
        # define subplot
        pyplot.subplot(1, 3, 1 + i)
        # turn off axis
        pyplot.axis('off')
        # plot raw pixel data
        pyplot.imshow(images[i])
        # show title
        pyplot.title(titles[i])
    pyplot.show()
```

This function can be called with each of our source, generated, and target images.

```
...
# plot all three images
plot_images(src_image, gen_image, tar_image)
```

Tying all of this together, the complete example of performing an ad hoc image-to-image tr et is listed below.

```
# example of loading a pix2pix model and using it for image to image translation
from keras.models import load_model
from numpy import load
from numpy import vstack
from matplotlib import pyplot
from numpy.random import randint

# load and prepare training images
def load_real_samples(filename):
    # load compressed arrays
    data = load(filename)
    # unpack arrays
    X1, X2 = data['arr_0'], data['arr_1']
    # scale from [0,255] to [-1,1]
    X1 = (X1 - 127.5) / 127.5
```

```
    X2 = (X2 - 127.5) / 127.5
    return [X1, X2]

# plot source, generated and target images
def plot_images(src_img, gen_img, tar_img):
    images = vstack((src_img, gen_img, tar_img))
    # scale from [-1,1] to [0,1]
    images = (images + 1) / 2.0
    titles = ['Source', 'Generated', 'Expected']
    # plot images row by row
    for i in range(len(images)):
        # define subplot
        pyplot.subplot(1, 3, 1 + i)
        # turn off axis
        pyplot.axis('off')
        # plot raw pixel data
        pyplot.imshow(images[i])
        # show title
        pyplot.title(titles[i])
    pyplot.show()

# load dataset
[X1, X2] = load_real_samples('maps_256.npz')
print('Loaded', X1.shape, X2.shape)
# load model
model = load_model('model_109600.h5')
# select random example
ix = randint(0, len(X1), 1)
src_image, tar_image = X1[ix], X2[ix]
# generate image from source
gen_image = model.predict(src_image)
# plot all three images
plot_images(src_image, gen_image, tar_image)
```


Running the example will select a random image from the training dataset, translate it to a Google map, and plot the result compared to the expected image.

Note: Your results may vary given the stochastic nature of the algorithm or evaluation procedure, or differences in numerical precision. Consider running the example a few times and compare the average outcome.

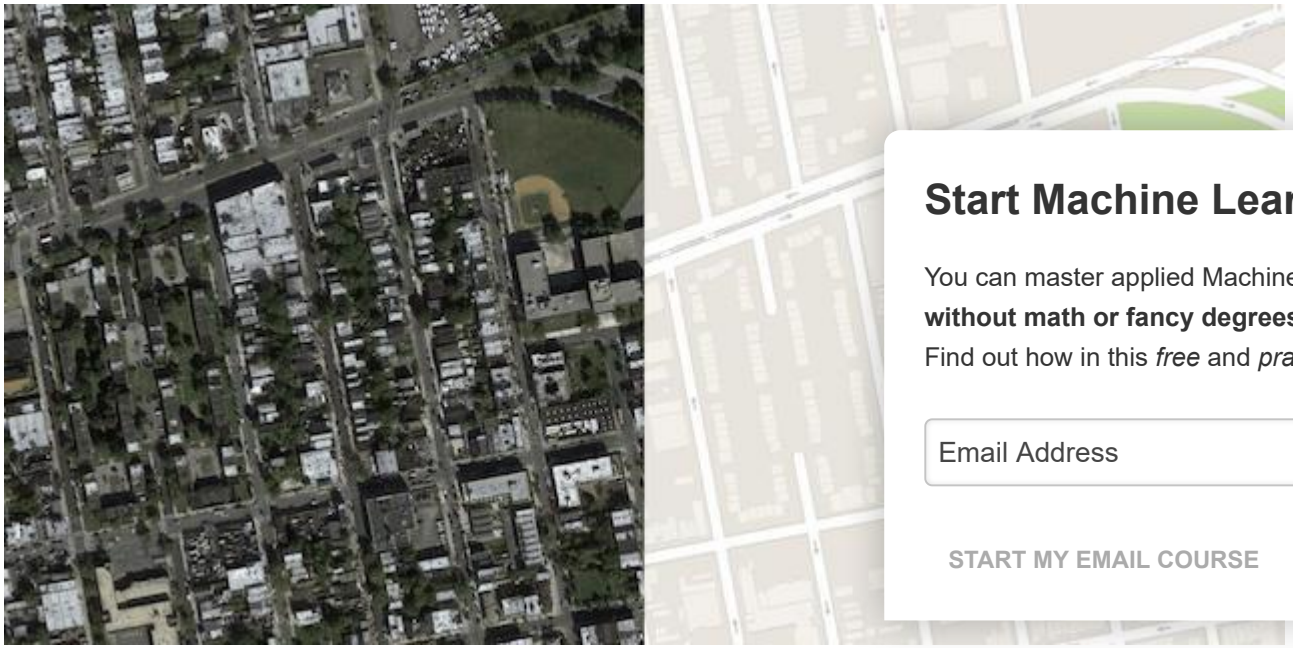
In this case, we can see that the generated image captures large roads with orange and yellow as well as green park areas. The generated image is not perfect but is very close to the expected image.

Plot of Satellite to Google Map Image Translation With Final Pix2Pix GAN Model

We may also want to use the model to translate a given standalone image.

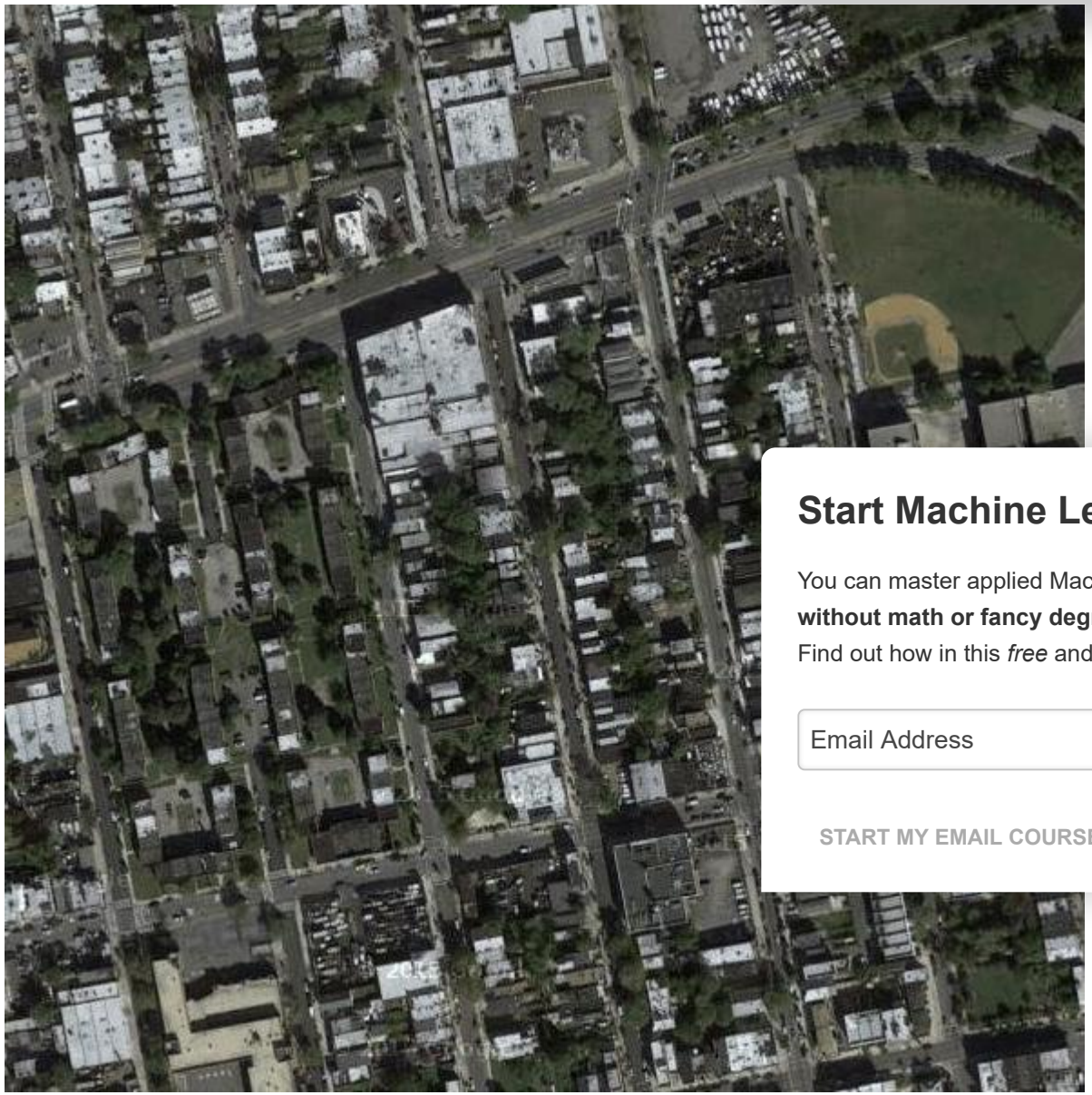
We can select an image from the validation dataset under maps/val and crop the satellite element of the image. This can then be saved and used as input to the model.

In this case, we will use "*maps/val/1.jpg*".



Example Image From the Validation Part of the Maps Dataset

We can use an image program to create a rough crop of the satellite element of this image to use as input and save the file as satellite.jpg in the current working directory.



Example of a Cropped Satellite Image to Use as Input to the Pix2Pix Model.

We must load the image as a NumPy array of pixels with the size of 256×256, rescale the pixel values to the range [-1,1], and then expand the single image dimensions to represent one input sample.

The *load_image()* function below implements this, returning image pixels that can be provi

Start Machine Learning

```
# load an image
def load_image(filename, size=(256,256)):
    # load image with the preferred size
    pixels = load_img(filename, target_size=size)
    # convert to numpy array
    pixels = img_to_array(pixels)
    # scale from [0,255] to [-1,1]
    pixels = (pixels - 127.5) / 127.5
    # reshape to 1 sample
    pixels = expand_dims(pixels, 0)
    return pixels
```

We can then load our cropped satellite image.

```
...
# load source image
src_image = load_image('satellite.jpg')
print('Loaded', src_image.shape)
```

As before, we can load our saved Pix2Pix generator model and generate a translation of th

```
...
# load model
model = load_model('model_109600.h5')
# generate image from source
gen_image = model.predict(src_image)
```

Finally, we can scale the pixel values back to the range [0,1] and plot the result.

```
...
# scale from [-1,1] to [0,1]
gen_image = (gen_image + 1) / 2.0
# plot the image
pyplot.imshow(gen_image[0])
pyplot.axis('off')
pyplot.show()
```

Tying this all together, the complete example of performing an ad hoc image translation with a single image file is listed below.

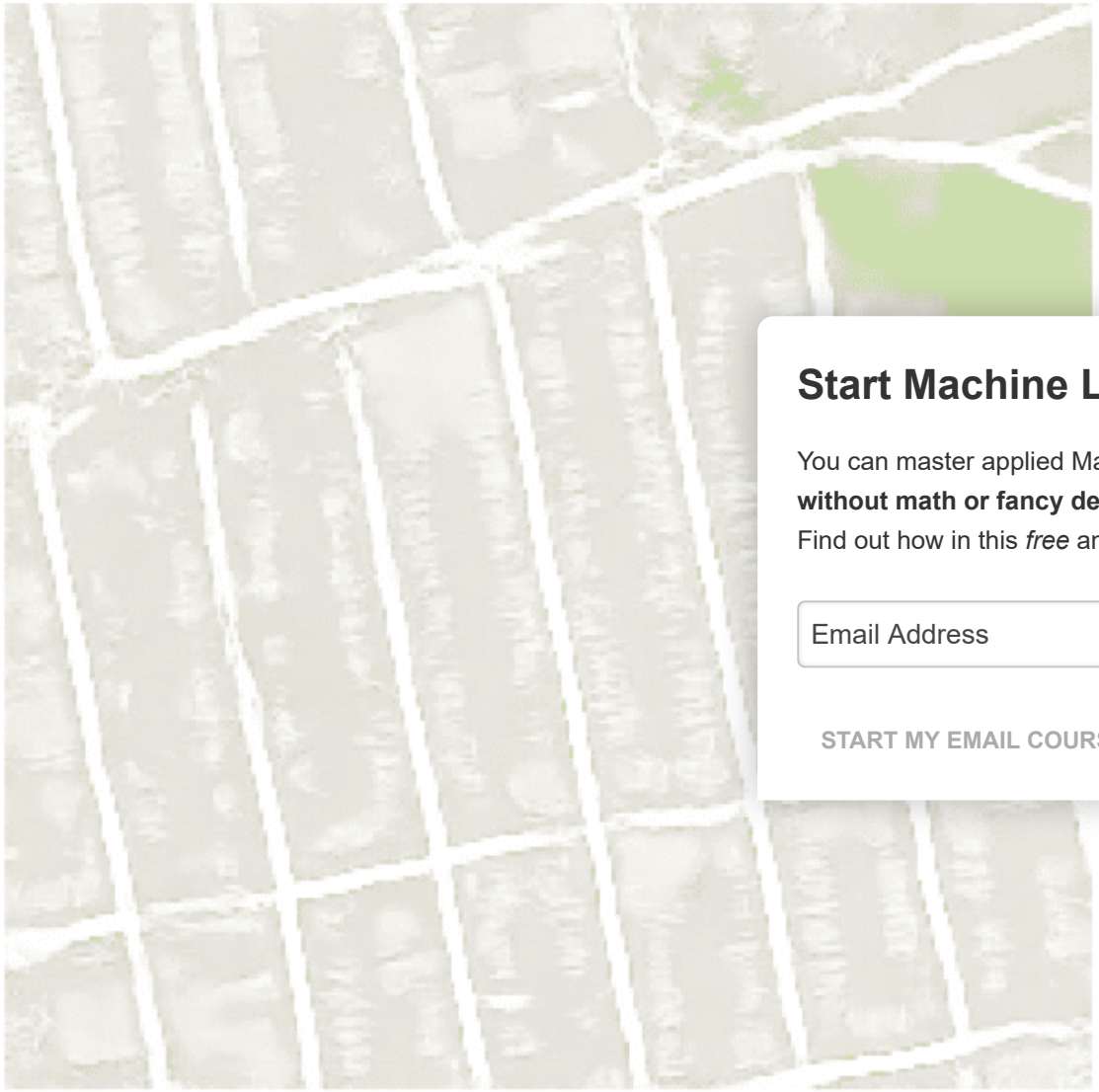
```
# example of loading a pix2pix model and using it for one-off image translation
from keras.models import load_model
from keras.preprocessing.image import img_to_array
from keras.preprocessing.image import load_img
from numpy import load
from numpy import expand_dims
from matplotlib import pyplot
```

```
 8
 9  # load an image
10  def load_image(filename, size=(256,256)):
11      # load image with the preferred size
12      pixels = load_img(filename, target_size=size)
13      # convert to numpy array
14      pixels = img_to_array(pixels)
15      # scale from [0,255] to [-1,1]
16      pixels = (pixels - 127.5) / 127.5
17      # reshape to 1 sample
18      pixels = expand_dims(pixels, 0)
19      return pixels
20
21  # load source image
22  src_image = load_image('satellite.jpg')
23  print('Loaded', src_image.shape)
24  # load model
25  model = load_model('model_109600.h5')
26  # generate image from source
27  gen_image = model.predict(src_image)
28  # scale from [-1,1] to [0,1]
29  gen_image = (gen_image + 1) / 2.0
30  # plot the image
31  pyplot.imshow(gen_image[0])
32  pyplot.axis('off')
33  pyplot.show()
```

Running the example loads the image from file, creates a translation of it, and plots the res

The generated image appears to be a reasonable translation of the source image.

The streets do not appear to be straight lines and the detail of the buildings is a bit lacking. Perhaps with further training or choice of a different model, higher-quality images could be generated.



Plot of Satellite Image Translated to Google Maps With Final Pix2Pix GAN Model

How to Translate Google Maps to Satellite Images

Now that we are familiar with how to develop and use a Pix2Pix model for translating satellite images to Google maps, we can also explore the reverse.

That is, we can develop a Pix2Pix model to translate Google map images to plausible satellite images. This requires that the model invent or hallucinate plausible buildings, roads, parks, and more.

We can use the same code to train the model with one small difference. We can change the order of the datasets returned from the *load_real_samples()* function; for example:

```
# load and prepare training images
def load_real_samples(filename):
    # load compressed arrays
    data = load(filename)
    # unpack arrays
    X1, X2 = data['arr_0'], data['arr_1']
    # scale from [0,255] to [-1,1]
    X1 = (X1 - 127.5) / 127.5
    X2 = (X2 - 127.5) / 127.5
    # return in reverse order
    return [X2, X1]
```


Note: the order of X1 and X2 is reversed.

This means that the model will take Google map images as input and learn to generate satellite images.

Run the example as before.

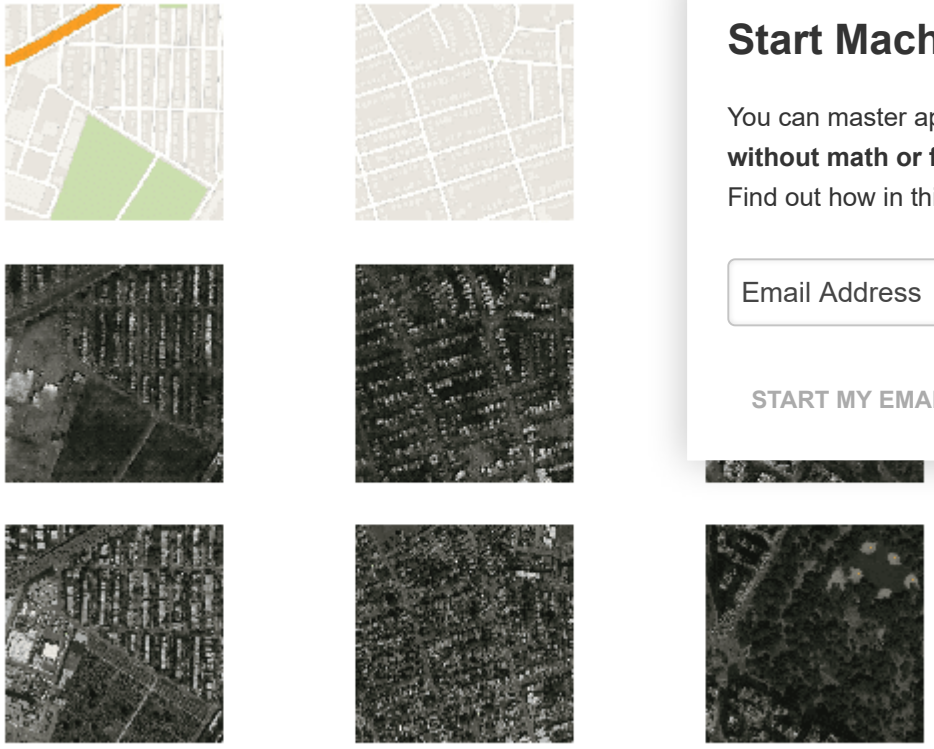
Note: Your results may vary given the stochastic nature of the algorithm or evaluation procedure, or differences in numerical precision. Consider running the example a few times and compare the average outcome.

As before, the loss of the model is reported each training iteration. If loss for the discriminator goes to zero and stays there for a long time, consider re-starting the training run as it is an example of a training failure.

```
>1, d1[0.442] d2[0.650] g[49.790]
>2, d1[0.317] d2[0.478] g[56.476]
```

```
>3, d1[0.376] d2[0.450] g[48.114]
>4, d1[0.396] d2[0.406] g[62.903]
>5, d1[0.496] d2[0.460] g[40.650]
...
>109596, d1[0.311] d2[0.057] g[25.376]
>109597, d1[0.028] d2[0.070] g[16.618]
>109598, d1[0.007] d2[0.208] g[18.139]
>109599, d1[0.358] d2[0.076] g[22.494]
>109600, d1[0.279] d2[0.049] g[9.941]
>Saved: plot_109600.png and model_109600.h5
```

It is harder to judge the quality of generated satellite images, nevertheless, plausible images are generated after just 10 epochs.

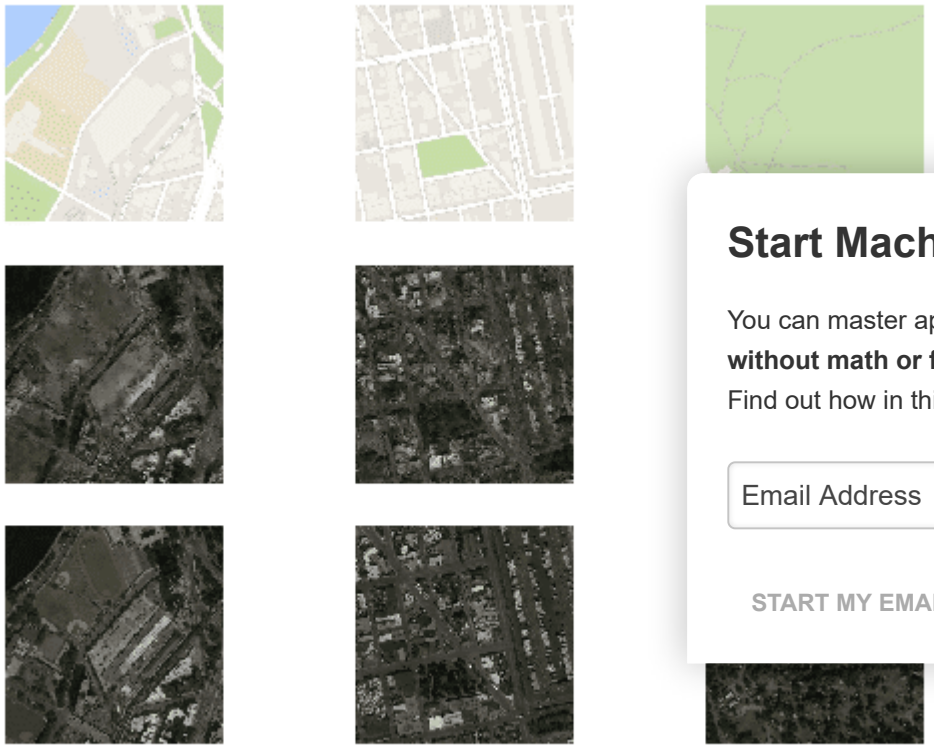


Plot of Google Map to Satellite Translated Images Using Pix2Pix Aft

Start Machine Learning

As before, image quality will improve and will continue to vary over the training process. A final model can be chosen based on generated image quality, not total training epochs.

The model appears to have little difficulty in generating reasonable water, parks, roads, and more.



Plot of Google Map to Satellite Translated Images Using Pix2Pix After 90 Training Epochs

Extensions

This section lists some ideas for extending the tutorial that you may wish to explore.

Start Machine Learning

- **Standalone Satellite**. Develop an example of translating standalone Google map images to satellite images, as we did for satellite to Google map images.
- **New Image**. Locate a satellite image for an entirely new location and translate it to a Google map and consider the result compared to the actual image in Google maps.
- **More Training**. Continue training the model for another 100 epochs and evaluate whether the additional training results in further improvements in image quality.
- **Image Augmentation**. Use some minor image augmentation during training as described in the Pix2Pix paper and evaluate whether it results in better quality generated images.

If you explore any of these extensions, I'd love to know.
Post your findings in the comments below.

Further Reading

This section provides more resources on the topic if you are looking to go deeper.

Official

- Image-to-Image Translation with Conditional Adversarial Networks, 2016.
- Image-to-Image Translation with Conditional Adversarial Nets, Homepage.
- Image-to-image translation with conditional adversarial nets, GitHub.
- pytorch-CycleGAN-and-pix2pix, GitHub.
- Interactive Image-to-Image Demo, 2017.
- Pix2Pix Datasets

API

- Keras Datasets API.
- Keras Sequential Model API
- Keras Convolutional Layers API
- How can I "freeze" Keras layers?

Summary

In this tutorial, you discovered how to develop a Pix2Pix generative adversarial network for image-to-image translation.

Specifically, you learned:

- How to load and prepare the satellite image to Google maps image-to-image translation dataset.
- How to develop a Pix2Pix model for translating satellite photographs to Google map images.
- How to use the final Pix2Pix generator model to translate ad hoc satellite images.

Do you have any questions?
Ask your questions in the comments below and I will do my best to answer.

About Jason Brownlee

Jason Brownlee, PhD is a machine learning specialist who teaches developers how to get results with modern machine learning methods via hands-on tutorials.

View all posts by Jason Brownlee →

‹ How to Implement Pix2Pix GAN Models From Scratch With Keras

A Gentle Introduction to CycleGAN for Image Translation ›

258 Responses to *How to Develop a Pix2Pix GAN for Image-to-Ima*

Deepanshu SIngh August 2, 2019 at 7:33 am #

Amazing tutorial. Detailed and clear explanation of concepts as well as the codes.

Thanks & Regards

REPLY

Jason Brownlee August 2, 2019 at 2:34 pm #

Thanks!

REPLY

Sean O'Connor August 3, 2019 at 10:38 am #

From a digital signal processing viewpoint a weighted sum is an adjustable filter.
Each layer in a conventional artificial neural network has n of those filters and the total compute is a brutal n squared fused multiply accumulates.
A fast Fourier transform is a fixed (nonadjustable) bank of filters, where each filter picks out frequency/phase.
There are other transforms that act as filter banks too, such as the fast Walsh Hadamard transform and these often require far less compute (eg. nlog(n)) than a filter bank of weighed sums.
The question then is why not use an efficient transform based filter bank and adjust the nonlinear functions in a neural network by individually parameterizing them?
Ie. change what you adjust:
https://github.com/S6Regen/Fixed-Filter-Bank-Neural-Networks
https://discourse.numenta.org/t/fixed-filter-bank-neural-networks/6392
https://discourse.numenta.org/t/distributed-representations-1984-by-hinton/6378/10

Jason Brownlee August 4, 2019 at 6:21 am #

Perhaps test your alternate approach and compare the results Sean?

Villem Lassmann August 6, 2019 at 7:41 pm #

It seems to me that the discriminator is not a 70×70 PatchGAN, since the 4th layer s
discirminator is a 142×142 PatchGAN. Please correct me if I am mistaken.

Jason Brownlee August 7, 2019 at 7:50 am #

REPLY

I believe you are mistaken.

You can learn more about the 70×70 patch gan in greater detail in this post:
https://machinelearningmastery.com/how-to-implement-pix2pix-gan-models-from-scratch-with-keras/

Villem Lassmann August 7, 2019 at 8:17 pm #

REPLY

That example has the same structure, 6 layers of Conv2D (including the last one). But when looking at the beginning of the post where You are calculating the receptive field with 5 layers of Conv layers. The calculation also states that there are only 3 layers of Conv2D with a stride of 2. I believe that the layer named C512 should be the second to last layer.

Jason Brownlee August 8, 2019 at 6:40 am #

REPLY

I believe the implementation matches the official implementation described here:
https://github.com/phillipi/pix2pix/blob/master/models.lua#L180

Vasudevakrishna August 20, 2019 at 3:50 pm #

REPLY

Thanks for the tutorial.
My question is in original paper they are giving the direction as configurable parameter.
But in your implementation I am unable to see that one.
How can do that for both direction.
Please explain

Jason Brownlee August 21, 2019 at 6:34 am #

REPLY

I show how to translate images in both directions in the above tutorial.

Alex August 29, 2019 at 11:52 pm #

REPLY

Many thanks for this amazing tutorial!

PS “There are 1,097 images”… and then there are saves every 10970 steps, and 109700 ste

Start Machine Learning

Jason Brownlee August 30, 2019 at 6:24 am #

REPLY

Thanks.

Fixed.

Salman Sajd September 1, 2019 at 7:13 am #

REPLY

Thanks for an amazing tutorial
How we use GAN for motion transfer or which type of GAN will best for Motion Transfer?

Jason Brownlee September 2, 2019 at 5:24 am #

I don't know off hand sorry, perhaps try a search on scholar.google.com

Lin September 6, 2019 at 9:52 pm #

Hello, thanks for the great article.
I have one question, but why you scale the image to [-1, 1] instead of [0, 1]?
Does this make the model behave differently?

Jason Brownlee September 7, 2019 at 5:29 am #

REPLY

Because the generator generates pixels in that range, and the discriminator must "see" all images pixels in the same range.

The choice of -1,1 for pixels is a gan hack:
https://machinelearningmastery.com/how-to-code-generative-adversarial-network-hacks/

Dang Tuan Hoang September 12, 2019 at 1:45 pm #

REPLY

Hi sir, is it possible to train this model with inputs and output of different sizes?
For example, I have 3 image a,b,c with size 50x50x3. I want the model to generate c from a,b. First I append a and b to get d with size 50x100x3, then use d as input, c as output

Jason Brownlee September 12, 2019 at 1:49 pm #

Yes, you can use different sized input and output, although the u-net will

Dang Tuan Hoang September 16, 2019 at 5:09 pm #

Could you give me some more details about how do I need to modify
texture

Jason Brownlee September 17, 2019 at 6:24 am #

Sorry, I don't have the capacity to prepare custom code for you.

Perhaps experiment with adding/subtracting groups of layers to the decoder part of the model and see the effect on the image size?

Dang Tuan Hoang September 17, 2019 at 12:37 pm #

I know you are very busy so I didn't ask for custom code, I just need something to start with. Thank for the suggestion sir !

Jason Brownlee September 17, 2019 at 2:34 pm #

Perhaps start with just the function that defines the model and try playing around with it.

REPLY

David September 27, 2019 at 9:17 am #

Hi Jason,

First of all, thank you very much for posting this tutorial, I learned a lot from it.

I have a question.

Do u think if I leave the picture resolution as it is rather than compressing them.
The performance is gonna be better? As my pictures between translation is very minor.

Thank you!

David

REPLY

Jason Brownlee September 27, 2019 at 1:15 pm #

Thanks, I'm happy that it helped.

Interesting idea. You mean likely working with TIFF images or other loess-less formats?

Probably not, but perhaps test it to confirm.

REPLY

Alice October 16, 2019 at 6:16 pm #

Hello,

How can I increase speed of training? It uses very small portion of gpu memory.

Jason Brownlee October 17, 2019 at 6:26 am #

REPLY

Some ideas:

Use less data.
Use a smaller model.
Use a faster machine.

Alice October 17, 2019 at 8:33 am #

I am using a machine with 8 gpus (8 X p4000) ☺
I mean, for example, while training on darknet, changing batch size directly affects gp
each gpu. And batch size doesn't affect it. So I need an adjustment just like on darkne
Thanks

Jason Brownlee October 17, 2019 at 1:50 pm #

I see, I'm not sure I can help sorry.

Samuel October 17, 2019 at 8:30 am #

REPLY

Swapping out the training data for the SEN1-2 dataset had amazing results. I can now translate Sentinel 1 images to RGB Sentinel 2! Many thanks for such a thorough tutorial.

Start Machine Learning

Jason Brownlee October 17, 2019 at 1:49 pm #

REPLY

Well done!

I would love to see an example of a translated image.

Mohammad October 20, 2019 at 12:03 pm #

REPLY

Hi Jason,

Thank you so much for your great website, it is fantastic.

I was wondering what your opinion is about the future research direction for this area of resea

Thanks

Jason Brownlee October 21, 2019 at 6:14 am #

Thanks.

Sorry, I don't have thoughts on research directions – I try to stay focused on the industrial

Michael October 28, 2019 at 9:08 am #

REPLY

Awesome tutorial on Pix2Pix. Your other GAN articles were great too and very helpful. After reading your tutorials, I was able to implement my own Pix2Pix project. All the code is on my GItHub. https://github.com/michaelnation26/pix2pix-edges-with-color

Jason Brownlee October 28, 2019 at 1:18 pm #

REPLY

Thanks.

Well done, that is very impressive!

REPLY

SSS November 17, 2019 at 10:43 pm #

– python version: 3.6.7
– tensorflow-gpu version: 2.0.0
– keras version: 2.3.1
– cuDNN version:10.0
– CUDA version:10.0

mnist_mlp.py (https://github.com/keras-team/keras/blob/master/examples/mnist_mlp.py) works perfectly but code which is given below gives me this error:

```
tensorflow.python.framework.errors_impl.UnknownError: Failed to get convolu
cuDNN failed to initialize, so try looking to see if a warning log message
[[node conv2d_7/convolution (defined at C:\Users\ACSECKIN\Anaconda3\envs\te
packages\tensorflow_core\python\framework\ops.py:1751) ]] [Op:__inference_k

Function call stack:
keras_scratch_graph
```


REPLY

Jason Brownlee November 18, 2019 at 6:45 am #

Perhaps try running on the CPU first to confirm your environment is working?

REPLY

SSS November 18, 2019 at 6:04 pm #

Code works with CPU. But I want to run in the GPU to complete in less time. CPU time is about 16 hours. I am open to any alternative to decrease the training time.

Jason Brownlee November 19, 2019 at 7:38 am #

REPLY

That is odd. I ran the examples on GPU without incident, specifically on EC2.

Perhaps try that?

Jack November 27, 2019 at 6:12 pm #

Based on my experience, I receive this error message when my GPU does not have enough memory to handle the process. Maybe try reducing the computational workload by using a smaller image? If not, use CPU if you are fine with it.

Jason Brownlee November 28, 2019 at 6:33 am #

Great suggestions!

Ivan November 23, 2019 at 9:05 pm #

Hi guys,

It takes 8 hours to train the model on GPU (Floydhub).
But several different models were saved in the process.

Can you explain why?

Jason Brownlee November 24, 2019 at 9:17 am #

REPLY

Perhaps try training on a large ec2 instance, it is much faster.

Models are saved periodically during training, we cannot know what a good model will be

Start Machine Learning

Ivan November 25, 2019 at 6:52 pm #

REPLY

Thanks for the tip!

Thought that more epochs give better results, since GAN`s cannot converge… so by theory no over fitting.. but I am new in the field, so I will look more into it ☺

Jason Brownlee November 26, 2019 at 6:00 am #

REPLY

Not with GANs. Perhaps start here:
https://machinelearningmastery.com/start-here/#gans

Jack November 26, 2019 at 2:07 pm #

Thank you for your awesome post, it is really detailed and helpful! I have a question : images are single channel and the maximum and minimum pixel values vary for each image, should I go about normalizing the images? Should I take the maximum of the whole batch of s for each sample and normalize each individually?

Also, when translating new images, what would be the values of image? Would it be between -1 to 1? If yes, how should I “denormalize” the values to the original? Thank you for your help!

Jason Brownlee November 26, 2019 at 4:10 pm #

REPLY

Thanks.

I recommend selecting a min and max that you expect to see for all time and use that to scale the data. If not known or it cannot be known, use the limits of the domain (e.g. 0 and 255).

Jack November 27, 2019 at 6:11 pm #

REPLY

Thank you for your suggestions! How would you suggest I “de-normalize” the data during testing? Should I use the same range (I am taking the range from the training data) and reverse the process on the test data?

Jason Brownlee November 28, 2019 at 6:32 am #

REPLY

Yes.

Syd Rawat December 1, 2019 at 2:50 pm #

Hi Jason,

Thank you very much for sharing such an in-depth analysis of Pix2Pix GAN. It is really helpful a CS background. I thought of applying this fro solving and inverse problems in Digital Holograp preliminary results I have got. As you know, the output of the model is a translated image, her I am looking for an image quality metric such as SSIM. Do you have any suggestions?

Thank You,

PS: As this post helped me enormously, I would like to cite your works on GANs in the future.

Jason Brownlee December 2, 2019 at 5:58 am #

REPLY

You’re welcome.

That sounds very cool! Perhaps one of the metrics here would be helpful:
https://machinelearningmastery.com/how-to-evaluate-generative-adversarial-networks/

Start Machine Learning

Anthony December 15, 2019 at 1:24 am #

REPLY

Hi! Currently implementing this with images with shape (256, 512, 3) and keep running to an error as follows:

“ValueError: A target array with shape (1, 16, 16, 1) was passed for an output of shape (None, 16, 32, 1) while using as loss `binary_crossentropy`. This loss expects targets to have the same shape as the output.”

I assume that this is due to the downsampling? Any help would be appreciated

Jason Brownlee December 15, 2019 at 6:08 am #

REPLY

Perhaps start with square images, get it working, then try changing to rectangula

Anthony Mäkelä December 15, 2019 at 6:21 am #

Hmm, alright! Could you explain why you use target_size=(256, 512) instead

Jason Brownlee December 16, 2019 at 6:03 am #

The images are 256×512 – as they contain the input and output images together.

We load them and split them into two separate square images 256×256 when working with the GAN.

MK December 16, 2019 at 11:45 pm #

REPLY

The discriminator error seems to be going to zero pretty quick, any tips to avoid this?

Jason Brownlee December 17, 2019 at 6:36 am #

REPLY

Perhaps try running the example a few times and continue running – it may recover.

Avdudia December 17, 2019 at 4:58 am #

REPLY

Thank you for this tutorial and simple code. I used it to perform image-to-image translation from Köppen–Geiger climate classification map (https://en.wikipedia.org/wiki/K%C3%B6ppen_climate_classification) to real satellite data, with truly amazing results, but I have a question.

In my strategy I create near one thousand pairs of 256×256 tiles from the Köppen–Geiger map (present in the Wikipedia article above), and a high-resolution satellite map of the Earth. In order to minimize deformation on tiles pairs near poles
pairs of image for GAN training (see https://photos.app.goo.gl/eGvpXghUtCB9kqkX6).

I trained the GAN until the end (n_epochs=100) with amazing results. Using training data give
(https://photos.app.goo.gl/a4EV6Gh15hAnYokm7). Even with hand-painted or with source ima
Geiger colormap, results are very nice (https://photos.app.goo.gl/eGbFmTH7YqYi4xfu5).

However I noticed that the result lacked of “relief” effect. Moreover, on large landmasses whe
noticeably affects the satellite view (e.g Tibetan Plateau or the Grand Canyon), the model res

As the climate map is composed of only 29 different indexed colors (plus the one I added for
used, instead of using a full RGB climate image as input.

So my idea was to store a heightmap of the earth on the first channel of the input image, and
channel. The third channel is kept unused. It results in a Red-Green image where the Red channel is the heightmap and the Green channel is normalized climate index (see https://photos.app.goo.gl/cN1cmCNLSXwwqzNB9).
The problem is that training with this input images give bad results compared to my first try (only climate date). Results were already convincing after 30 epochs in my my first try, with smooth transition between climates, why here the boundaries are clearly visible in generated images (see https://photos.app.goo.gl/Q1vjjeY8ewWrCZYv5).
I tried to run the training several times to ensure that it was not purely bad luck, with the same result.

I don’t understand because climate index can clearly be stored on one channel without information loss, and heightmap provides additional data, so it should improve the results. Is it simply because it needs more epochs ?

Thank you in advance and sorry for the long post and for my english, it is not my native langua

Jason Brownlee December 17, 2019 at 6:42 am #

REPLY

Well done!

Very cool application.

Two thoughts off the cuff. One would be to make image synthesis conditional on two input images (source and the height map). A second would be to have 2 steps – one step to generate the initial translation and a second to improve the generated image with a height map.

I'm eager to hear how you go!

Avdudia December 18, 2019 at 7:32 am #

Thank you very much ! The aim is to develop a tool for worlbuidling and crea
Atrifexian's tutorials https://www.youtube.com/watch?v=5ICbxMZJ4zA&t=1s).

I used the idea of using R and G channels for heightmap and climate following this th
https://github.com/junyanz/pytorch-CycleGAN-and-pix2pix/issues/498. They recomme
your code is limited to 3 channels and as I'm a complete beginner I still don't know ho

However it seems indeed that training on more epochs actually gives good results wit
restarted it with a limit of 1000 epochs. However I have to redo the first 100, an I run t
unstable (I only managed to reach 100 epochs twice).

Do you have a tutorial on how to make complete checkpoints in order to continue the training in case of crash ? If I understand well, your summarize_performance function only save the generator model, so we should have to save the whole gan_model and reload it for later training. Do you have documentation or examples concerning this ?

Thank you so much for your tutorial. I'll keep you informed on later developments !

Jason Brownlee December 18, 2019 at 1:28 pm #

REPLY

Yes, the above code already saves the model every n steps.

See the summarize_performance() function.

Start Machine Learning

JS December 18, 2019 at 2:06 am #

REPLY

Should i approach this the same way if i have images containing white backgrounds? Similar to the edges2shoes dataset?

Jason Brownlee December 18, 2019 at 6:10 am #

REPLY

Perhaps test it with a prototype?

JS December 18, 2019 at 8:01 pm #

For some reason i end up only with blank white images…

Sirine December 25, 2019 at 5:56 am #

Hello,
Thanks for the wonderful tutorial, Please how can I adapt the generator and the descriminator
matrix(2,64)

Jason Brownlee December 25, 2019 at 10:37 am #

REPLY

Sorry, I don't follow.

Elbek Khoshimjonov January 2, 2020 at 7:59 pm #

Start Machine Learning

Thanks Jason for great post!
I have tried this code, but images do not appear to be good enough, and discriminator loss becomes 0 after 10-15 epochs.

Jason Brownlee January 3, 2020 at 7:27 am #

REPLY

Perhaps try using a different final model or re-fit the model?

Arsal January 28, 2020 at 10:55 pm #

REPLY

I want to continue training from the last checkpoint stored. Can you help me in resum

Jason Brownlee January 29, 2020 at 6:36 am #

Yes, load the model as per normal and call the train() function.

Arsal January 28, 2020 at 10:57 pm #

I have a dataset consisting of 216 images. I trained for 100 epochs but unfortunately the results are not good. Can you help me how can I improve the results?

Jason Brownlee January 29, 2020 at 6:36 am #

REPLY

Yes, try some of these suggestions:
https://machinelearningmastery.com/how-to-code-generative-adversarial-network-hacks/

Laura January 30, 2020 at 3:19 am #

REPLY

Thank you for this wonderful tutorial! It has been extremely helpful. I was wondering if you had considered data augmentation?

Jason Brownlee January 30, 2020 at 6:56 am #

REPLY

For GANs, not really, in general, yes:
https://machinelearningmastery.com/how-to-configure-image-data-augmentation-when-training-deep-learning-neural-networks/

Laura January 30, 2020 at 3:43 am #

Thanks!

This is probably a newbie question, but I am new to GANs. In my limited experience with deep
process to sort of evaluate how well it was “learning”> I then had another dataset I called the
was complete. Here it seems like you don’t use any validation during the training process. An
dataset. Is that something unique to GANs or can validation be included in the training proces

Jason Brownlee January 30, 2020 at 6:58 am #

No, I often use tests sets for validation to make tutorials simpler:
https://machinelearningmastery.com/faq/single-faq/why-do-you-use-the-test-dataset-as-the-validation-dataset

Michael February 1, 2020 at 12:34 am #

REPLY

Hi! Im running this on Titan V and it seems to be running extremely slow. Any ideas as to why?

Start Machine Learning

Jason Brownlee February 1, 2020 at 5:56 am #

REPLY

Training GANs is slow…

Perhaps check you have enough RAM and GPU RAM?
Perhaps compare results on an ec2 with a lot of RAM?
Perhaps adjust model config to use less RAM?

Lisa October 28, 2020 at 8:57 pm #

REPLY

I suggest using google colab. You can use their GPUs for training. It's much faste

Jason Brownlee October 29, 2020 at 8:02 am #

GPUs are practically a requirement when working with GANs.

pramod kumar February 9, 2020 at 2:42 am #

sir can i know how you downloaded the images of satellite and maps can you please help me to download my own dataset for this project

Jason Brownlee February 9, 2020 at 6:24 am #

REPLY

See the section of the above tutorial "Satellite to Map Image Translation Dataset" on how to download the dataset.

Vaibhav Vijay kotwal February 24, 2020 at 6:28 pm #

REPLY

Hi Jason,
From the theory, we understand that dicriminator learns the objective loss function.However referring to define_GAN(),line 56 in the code, I am not able to see the object loss learnt by discriminator getting passed to GAN model. I see that the model doesnot converge as expected

Thanks and Regards,
Vaibhav Kotwal

REPLY

Jason Brownlee February 25, 2020 at 7:42 am #

We create a composite model that combines G and D so that G is trained via inverse corrections to D.

Perhaps see this:
https://machinelearningmastery.com/how-to-code-the-generative-adversarial-network-trai

Fırat Erdem February 26, 2020 at 10:49 pm #

Thanks for the great tutorial. I need help with something. I want to see accuracy met
the training process. And I want to see this for each epoch, not for each steps. Like a standar
things to code ? I couldn't apply it because it is different than standard CNN codes. I would re

REPLY

Jason Brownlee February 27, 2020 at 5:50 am #

Accuracy is a bad metric for GANs.

See this:
https://machinelearningmastery.com/how-to-evaluate-generative-adversarial-networks/

REPLY

Tom March 8, 2020 at 5:13 am #

Can pix2pix Gan save and load againt without training again

Start Machine Learning

Jason Brownlee March 8, 2020 at 6:16 am #

REPLY

Yes, see this tutorial for an example:
https://machinelearningmastery.com/how-to-develop-a-pix2pix-gan-for-image-to-image-translation/

Tom March 8, 2020 at 6:37 am #

REPLY

Yes,thank for your reply!!And I want to ask another questions!!Can the other kind of GAN versions save and load?

Jason Brownlee March 9, 2020 at 7:08 am #

Yes, they are all Keras models that can be saved and loaded.

Ehsan Karimi March 12, 2020 at 2:10 am #

Hi Jason,
Thanks for the great tutorial. I have a problem with image scales. In first step, after splitting the input images, I check the image size, instead of of 256*256 pixel they are 134*139 with background. Also, at translation a given standalone image using by model step, the output should be 256*256 same as input, but I get 252*262 output again with background.
I was wondering if you would mind letting me know where is the problem?
Thanks in advance
Ehsan

Jason Brownlee March 12, 2020 at 8:51 am #

REPLY

I don't know the cause of your fault. Sorry.

Paolo March 13, 2020 at 8:13 am #

REPLY

Great work Jason. Just one question: do you believe that this approach could work using a RGB satellite image against its mask image, to make some kind of image segmentation ?
Thanks in advance

Jason Brownlee March 13, 2020 at 8:24 am #

REPLY

Perhaps try it. Prototypes are a fast way to get answers.

Paolo March 13, 2020 at 8:31 am #

I mean, the mask image would have just 2 colors (yes/not) … this was my co

Iqbal March 19, 2020 at 9:05 pm #

It is an interesting article publish here. I am new using this, i want some question for the first script for clear explanation :

1. I saw the loaded data is maps in train and test folder. I want to know which 3 sample was loaded from the folder train? because the results was : (1096, 256, 256, 3) (1096, 256, 256, 3). I understand 1096 is the certain amount of image in that folder. and 256 I still dont understand because when I open picture 256 is not the same as the it was loaded.

2. I saw the folder contain image in train and test. I want to ask the train, example 1.jpg it contains two image from satellite. May I know how to develop the left picture and the right picture or it develop itself? Also in test does it develop itself or have to save it first?

Need some explanation for preparing using it in the future. Thank you

Jason Brownlee March 20, 2020 at 8:43 am #

REPLY

The images are 256×256 squares with 3 color channels. You can learn more about loading images here:
https://machinelearningmastery.com/how-to-load-and-manipulate-images-for-deep-learning-in-python-with-pil-pillow/

Sorry, I don't follow your second question, perhaps you can elaborate?

Iqbal March 20, 2020 at 9:51 am #

REPLY

thank you for replying. For my second question I saw the folder train and test contains an image. So the question come up :
1.Does the image build itself?
2. If No, how do you make the images side by side that contains two image in one ima

Thank you very much.

Jason Brownlee March 20, 2020 at 1:15 pm #

The tutorial shows how to load the images and prepare them for modelin

yasser March 22, 2020 at 10:12 am #

REPLY

hi thank you for your work !
I need your help,I need the same model but the input of the generator is one channel and not three .
I have tried to change it but it does'nt work .thank you

Jason Brownlee March 23, 2020 at 6:09 am #

REPLY

Sorry to hear that.

Perhaps confirm that your images are grayscale (1 channel), then change the model to expect 1 channel via the input shape.

runnergirl March 23, 2020 at 5:53 am #

REPLY

Hi! I wanted to train myself. I prepared them just like in this tutorial. Size X1 and X2 are the same. Data display works. But I get this error:

'Got inputs shapes: %s' % (input_shape))
ValueError: A `Concatenate` layer requires inputs with matching shapes except for the concat axis. Got inputs shapes: [(None, 2, 2, 512), (None, 1, 1, 512)]

What have I done wrong?

Jason Brownlee March 23, 2020 at 6:16 am #

I don't know sorry.

Mick October 28, 2020 at 10:26 pm #

Have you been able to solve this problem. I get the same issue.

GoComputing October 30, 2020 at 10:36 pm #

REPLY

This is probably because your input size is not divisible by 256.

ouis March 24, 2020 at 1:04 am #

REPLY

Start Machine Learning

hello I have this problem I dont know why :
ValueError: Graph disconnected: cannot obtain value for tensor Tensor("input_14:0", shape=(?, 256, 256, 3), dtype=float32) at layer "input_14". The following previous layers were accessed without issue: ['input_15']

Jason Brownlee March 24, 2020 at 6:05 am #

REPLY

Perhaps confirm that your keras and tensorflow versions are updated?

Harshit April 19, 2020 at 2:37 am #

REPLY

Hi. I tried to use a different dataset using this code. Specifically the edges2shoes dat
Everytime i ran into memory error. My ram is 16GB still that was not enough. I managed to cr

Also could you be kind enough to make tutorial of Tensorflow/Keras of pix2pixHD since it is m
compared to normal pix2pix.

Jason Brownlee April 19, 2020 at 6:01 am #

Perhaps use a sample of the data?
Perhaps use progressive loading?
Perhaps use an ec2 instance wth more ram?

Thanks for the suggestion.

Harshit April 19, 2020 at 11:48 am #

REPLY

Thanks I think your tutorial here https://machinelearningmastery.com/how-to-load-large-datasets-from-directories-for-deep-learning-with-keras/ will be helpful in my case.

I am using my own machine not AWS instance.

REPLY

Jason Brownlee April 19, 2020 at 1:18 pm #

Great!

REPLY

Phil April 30, 2020 at 11:23 pm #

Hi Jason. Great Article. Good explanation. Your articles gave me a good overview and starting point when I started developing my own networks. But I have two questions:
1:) From what I can see the original code on Github seems to be slightly different to your code encoder and a decoder layer. On Github data from an encoder layer passed to a decoder laye the data is passed directly after the convolution(or batch norm/dropout), in contrary to the solu
2.) What does the flag 'training=True' do when calling batch normalization layer or dropout lay
Thanks in advance.

REPLY

Jason Brownlee May 1, 2020 at 6:40 am #

Perhaps. I thought I had the architecture spot on based on the paper and the released code.

Training=True causes the layer to think it is always in training model. E.g. normally batchnorm and dropout operate differently in training vs inference. More here:
https://machinelearningmastery.com/how-to-implement-pix2pix-gan-models-from-scratch-with-keras/

REPLY

ElenaRR May 3, 2020 at 4:08 am #

Hi Jason,
thank you very much for this tutorial, it's awesome!

I have the problem that you mentioned at the end of the article. D1 loss goes to zero after 80-90 steps. Could you explain me why this happens and how can I solve it?

In addition to this, I can see that only one image is used in every iteration (one real,one fake) where n_batch = 1. Shouln't we use more than one pair of images to train in each step?

select a batch of real samples
[X_realA, X_realB], y_real = generate_real_samples(dataset, n_batch, n_patch)

Thank you very much!

REPLY

Jason Brownlee May 3, 2020 at 6:18 am #

Yes, this is probably a failure mode:
https://machinelearningmastery.com/practical-guide-to-gan-failure-modes/

You can try fitting the same model again, save models to file along the way.

You can try tuning the model architecture or training algorithm:
https://machinelearningmastery.com/how-to-code-generative-adversarial-network-hacks/

ElenaRR May 3, 2020 at 7:23 pm #

So, the model as it is in this example is not going to work properly? This link is not a pixtopix architecture. I tried also the example and work perfectly for cGAN and GAN with fminst, but the problem is this Pixtopix architecture.

REPLY

Jason Brownlee May 4, 2020 at 6:19 am #

The example does work correctly, but you may need to train it a few times.

GANs are unstable by definition.

ElenaRR May 3, 2020 at 7:39 pm #

REPLY

I've seen that there are specific codes in your book about this. Is a more complete example for pixtopix? Or is it the same?

Jason Brownlee May 4, 2020 at 6:19 am #

REPLY

The examples of pix2pix in the book are based on this example.

shami May 11, 2020 at 1:02 am #

long live 100 years …. superb tutorial with clear explanation

Jason Brownlee May 11, 2020 at 6:02 am #

Thanks!

bobbyP May 14, 2020 at 2:29 pm #

REPLY

anyway to use a keras data loader/generator for on the fly image loading dsirng training? say for example your training size is really large and loading all at once would result in out of memory errors? thanks so much for all your tutorials, they are incredbile!

Jason Brownlee May 15, 2020 at 5:55 am #

REPLY

Yes, see this:
https://machinelearningmastery.com/how-to-load-large-datasets-from-directories-for-deep

cans May 20, 2020 at 6:12 am #

REPLY

Hi Sir,
First of all i read your all tutorials. You are helping me more than my consultant. Thank you so much.
I am new at Gans. Sorry for my quesitons. But i cant understand how i test this model?
I will use validation set okey.
After training model i wont give target, just give source image?I cant get it.
Or only ı load model and train with validity set?
Omg i cant explain myself. I hope u are understand me.

Jason Brownlee May 20, 2020 at 6:29 am #

You're welcome!

Evaluating GANs is challenging. We do not use a validation set. Instead we generate ima
enough.

cans May 21, 2020 at 2:39 am #

thank you for your answer Sir,
i wanna try pix2pix gans for image enhancement.
I'l use source images are low contrast, targets are high contrast ,what do you think? i'm trying to improve thermal image.
I hope this system will work.

Jason Brownlee May 21, 2020 at 6:21 am #

REPLY

Sounds great!

Gruhit Patel May 25, 2020 at 1:44 pm #

REPLY

Here in Discriminator what is the need for Concatenating Source and Target images ? What effect would it have ?

Jason Brownlee May 26, 2020 at 6:12 am #

REPLY

Sorry, which section are you referring to? Where (which section/line?) do we concatenate images?

Gruhit Patel May 27, 2020 at 2:48 pm #

In discriminator. where we are concatenating Source Image and target Imag
Actually I'm building a GAN model color transformation from Gray to RGB.
My discriminator and Generator model's loss falls to zero. So wanted to know that wh
discriminator. And if you have any advice for my model than tell it too.
THANKS in advance…

Jason Brownlee May 28, 2020 at 6:09 am #

Here we are training a conditional model, e.g. generate a target image conditional on the source image.

E.g. it is the purpose of the model.

Ujjayant Sinha May 27, 2020 at 12:54 am #

REPLY

Hello. I compared the images from the summarize_performance() to predictions on unseen ones, which turned out to be quite horrible.
Can you suggest some ways to tackle this problem ?

Start Machine Learning

Jason Brownlee May 27, 2020 at 7:58 am #

REPLY

Try training the model a few times, save many times during each run, choose a model that generates good images.

Zsolt Lipcsei May 27, 2020 at 8:50 am #

REPLY

Hi,

How can I use the generator model to predict any size of images? I mean not just squres sizes. Is there any way at all?

Jason Brownlee May 27, 2020 at 1:26 pm #

You will have to change the generator/discriminator and also the training dataset

Muhammad Ammar Malik June 12, 2020 at 2:49 am #

Thank you for the awesome post. I have 2 questions if you can answer please.

First question, you have mentioned:
“In this case, we will use the model saved at the end of the run, e.g. after 10 epochs or 109,600 training iterations.”

Shouldn’t the training iterations be 10,960 after 10 epochs.

Second question, what is the rationale behind using random index to generate real and fake samples? Why can’t we simply iterate over all the samples one by one to make sure no image is missed or used more than once in 1 training step?

Jason Brownlee June 12, 2020 at 6:18 am #

REPLY

Images are generated after every 10 epochs, it runs for 100 epochs, meaning we

Start Machine Learning

we used the model after 100 epochs. Fixed.

We can do it for all image, I wanted to work with one image, to show we can use the model ad hoc. Readers often find that step confusing so I must demonstrate it.

Riccardo June 18, 2020 at 11:27 pm #

REPLY

Hi!
thank you for the great tutorial, you helped me a lot!

i have just a question: am i doing something wrong or is it normal that for a X input i do not have a unique Y output.
Let me explain better: if i repeat n times the prediction i get n different Y images (i'm checking pixels differences).
I'm translating this this example to another application and having the exact same output everytime will make it works.

I tried to look for a random noise vector or something like that but it seems that this is not the

Jason Brownlee June 19, 2020 at 6:14 am #

Recall we are using a GAN, so we have two models, the first predicts whether in
images conditional on another image.

If you are new to GANs perhaps start here:
https://machinelearningmastery.com/what-are-generative-adversarial-networks-gans/

Riccardo June 18, 2020 at 11:29 pm #

REPLY

EDIT:
" I'm translating this this example to another application and having the exact same output will make it works" *

Riccardo June 18, 2020 at 11:31 pm #

REPLY

Hi!
this tutorial helped me a lot!

i have just a question: is it normal that if i repeat n times the prediction, with the same input, i have n different outpus?

I'm checking directly changes in pixel values in Y outputs.

Jason Brownlee June 19, 2020 at 6:15 am #

REPLY

Yes, this is expected given the stochastic nature of some of the layers.

Riccardo June 19, 2020 at 5:35 pm #

thank you for the reply!
I'm guessing that dropouts are introducing randomization, i'll try without that.

Jason Brownlee June 20, 2020 at 6:08 am #

Yes, also there are layers that inject noise directly.

Riccardo June 22, 2020 at 6:09 pm #

i'm sorry if i seem annoying but i do not see layers that can introduce directly noise in the code you provided.

I see convs, batchnorms and concatenates.
Can you please tell me which layer is introducing noise directly?

i think that i have missed something about these layers but reading through the documentation it seems like i know them pretty good.

Start Machine Learning

I really need to understand the position of these noise generator and remove them in order to use a GAN for my application (maybe it could be impossible but i wish to try =))

Jason Brownlee June 23, 2020 at 6:16 am #

Sorry, my mistake, I was thinking of a different GAN.

Sahil Singla June 22, 2020 at 7:17 am #

REPLY

Thanks for the great tutorial.

I have one small doubt:
Do we traverse over the complete dataset?
We passed our entire dataset to the generate_real_samples function and everytime it choose
traverse again and again.

So, we might not be traversing over the complete dataset in single epoch?

Please let me know your thoughts.

Thanks.

Jason Brownlee June 22, 2020 at 1:25 pm #

REPLY

You're welcome.

Correct. On average we cover the whole dataset many times.

Sahil Singla June 22, 2020 at 6:20 pm #

REPLY

So, there is a possibility of missing certain datapoints. This can become a pr

Start Machine Learning

So should I change the code to make sure, it traverse over entire data, or is it still ok, if we don't do that ?

Jason Brownlee June 23, 2020 at 6:17 am #

REPLY

If you prefer. I'm not convinced it makes a difference, but could be a fun experiment.

Riccardo June 23, 2020 at 6:08 pm #

REPLY

oh, ok no problem!
i think that i will investigate stochasticity trought the different convs and batch norm in ord
X input.

best regards

yacine June 30, 2020 at 7:30 pm #

Thank you so much for this super clear explanation and code.

Jason Brownlee July 1, 2020 at 5:52 am #

REPLY

You're welcome!

Steve Newbold July 3, 2020 at 11:59 pm #

REPLY

If I wanted to use an input with three colour channels and a target of four colour channels, can this be configured or is it best to just create an additional black 4th channel on the input?

I noticed some greyscale-to-colour models just use the same data in each channel to represent grey images so presumed it mush be easier to do this than make the model work with differing numbers of channels.

Also, thanks for the excellent resource!

REPLY

Jason Brownlee July 4, 2020 at 6:01 am #

Off the cuff I recall the images have to have the same number of channels. Perhaps experiment/research and see if you can deviate from this norm.

M July 8, 2020 at 10:42 am #

Thanks for this great post!
For your generator's loss, how can I know if are you minimizing 1: log(1 – D(G(x))) or maximiz
How can one change the loss function, any reading suggestions?
Some people say the choice of generator's loss can help the model to not get stuck in early s

Jason Brownlee July 8, 2020 at 1:43 pm #

Good question, this may help you understand the loss function for GANs:
https://machinelearningmastery.com/how-to-code-the-generative-adversarial-network-training-algorithm-and-loss-functions/

You can see examples of alternate loss functions in tutorials here:
https://machinelearningmastery.com/start-here/#gans

REPLY

Rao July 21, 2020 at 7:02 am #

Hello Jason,

What would be the optimal loss values (Generator and Discriminator loss) of a successful conditional GAN model? Are the values same as an unconditional GAN ? (i.e around 0.7 or 0.6, as mentioned in your unconditional GAN article)

Secondly, I have done the training of pix2pix for a certain image to image translation task in two different ways.

1st method: Trained the discriminator patch outcome against a matrix of real or fake labels (as mentioned in this article)
2nd method: The discriminator still gives a patch, but this time, the patch average was taken and was trained against a single value (i.e avg value of the patch against a real or fake label).

During the training (towards the saving of a good model), the first method, yields a patch avg value of about 0.4 for a real image pair and about 0.3 for a fake image pair.
But the second model, yields a patch avg value of about 0.0004 for both real and fake image pairs.

Both these models yielded a good quality image with its Generator and the Discriminator loss standing around 0.7 and 0.6 respectively. My doubt is why such discrepancy with the avg patch values even though both the models yields a good c 4 doesn't make sense even though this model yielded a good translated image.(Because as far for a real pair should be close to 1 for a real pair and 0 for a fake image pair. This would mean for a real pair and 0 for a fake image pair).

What should be the avg patch values for a good model? Any amount of insights into this woul

Thanks!

Jason Brownlee July 21, 2020 at 1:43 pm #

GANs don't coverage, so there is no optimal loss values, you can learn more here:
https://machinelearningmastery.com/faq/single-faq/why-is-my-gan-not-converging

Comparing loss between models/runs is not valid.

GANs are hard to evaluate, subjective image quality is about the best we can do, although there are some metrics described here that might help:
https://machinelearningmastery.com/how-to-evaluate-generative-adversarial-networks/

Gruhit Patel July 22, 2020 at 4:56 pm #

Sir, Why exactly are we merging two images in discriminator ?? What effect does it have ?? And why are we not keeping just the colored image in Discriminator ??

REPLY

Jason Brownlee July 23, 2020 at 6:03 am #

The discriminator is given the input image and a target image and comments on whether the target is a real translation or a generated translation.

REPLY

Jamal July 24, 2020 at 7:54 pm #

Can PIX 2 PIX GAN works for gray-scale images??
what if if we use the above same architecture for gray-scale source and target images

Jason Brownlee July 25, 2020 at 6:17 am #

Modification of the model architecture is required.

REPLY

Zoya July 24, 2020 at 5:10 pm #

I used different data of source and target image. My source and target images are gray scale.
But when i run the code , the discriminator loss is going to zero with very few iterations but generator loss is very high that is ,,9782.150 up to so on.
It cannot be decreasing ….What can i do ??

REPLY

Jason Brownlee July 25, 2020 at 6:13 am #

GAN loss does not converge, you can learn more here:
https://machinelearningmastery.com/faq/single-faq/why-is-my-gan-not-converging

John July 24, 2020 at 7:44 pm #

REPLY

I have different source and Target images. And My source and target images are in gray scale but my discriminator loss is going to very low reaches to zero but generator loss is very high.

what can I do now ?? Can Pix to Pix GAN work for gray scale images.

Jason Brownlee July 25, 2020 at 6:17 am #

You may need to tune the model – explore – in order to discover how to best mo
images.

Gruhit Patel July 26, 2020 at 12:11 pm #

What sort of modification you think are required ?? Like the architecture won't be U-s

Jason Brownlee July 26, 2020 at 1:40 pm #

REPLY

It is hard to know – experimentation is required, perhaps start with tuning the learning rate with a similar network structure adjusted for the changed number of channels.

Jeremy Bolton July 28, 2020 at 7:50 pm #

REPLY

Hi!

Start Machine Learning

Thanks for your great work!

I found your model above and tried with the cityscapes images. I trained ~3000 image pairs from segmentation to photographic pictures. First I convert the images to 256×256 and kept the 100 epochs, then trained with 250 epochs. The results were good, but blurry, so I converted the original 1024×2048 resolution images to 512*512 and trained them till 250 epochs.

The results didn't really improve, but somehow I'd like to get less blurry pictures. I think increasing the number of epochs or the image resolution didn't change a lot, so my question would be: Do I need to change on the architecture of the models? If yes, can you give me a hint what further layers should I use?

Thank you very much and keep up the good work!

Jason Brownlee July 29, 2020 at 5:50 am #

You may need to experiment with the model architecture and learning hyperpara
specific dataset.

Jeremy Bolton July 30, 2020 at 8:40 pm #

Thanks for your reply, Jason.

Can you give me a hint, what architectural changes I should start with if I want to trair
instead of 256×256? More conv2d layers, dropout layers or multiple discriminators/generators as in pix2pixhd?

Thank you.

Jason Brownlee July 31, 2020 at 6:18 am #

REPLY

A good approach is to scale the number of blocks up or down from the current to meet the desired image dimensions.

I would encourage you to experiment and observe the effects on input/output shapes to get a feeling for it.

phaneeshwar August 2, 2020 at 4:18 am #

REPLY

dataset = LoadRealData (‘C:/Users/Eeshwar/Desktop/deep learning/maps11.npz’)
print(‘Loaded’,dataset[0].shape, dataset[1].shape)
Imgshape = dataset[0].shape[1:]
dmodel = DiscriminatorModel(Imgshape)
gmodel = GeneratorModel(Imgshape)
ganmodel = GANModel(dmodel,gmodel,Imgshape)

TrainModel(dmodel,gmodel,ganmodel,dataset)

Loaded (1096, 256, 256, 3) (1096, 256, 256, 3)
WARNING:tensorflow:Discrepancy between trainable weights and collected trainable weights, did you set `model.trainable` without calling `model.compile` after ?

InvalidArgumentError: data[0].shape = [4] does not start with indices[0].shape = [2]
[[{{node training/Adam/gradients/gradients/loss_3/dense_2_loss/Mean_grad/DynamicStitch}}]

Sir could you please help me to resolve this issue. I Thank You in advance

Jason Brownlee August 2, 2020 at 5:47 am #

Sorry to hear that, this will help:
https://machinelearningmastery.com/faq/single-faq/why-does-the-code-in-the-tutorial-not-work-for-me

Dhruv Agarwal August 2, 2020 at 4:41 pm #

REPLY

Hello sir, the tutorial was great, but i have 2 questions.

1) In the define_discriminator() function, you have set the loss_weights parameter to 0.5, to slow down the training of discriminator. Can’t we reduce the learning rate of the discriminator model to slow the training, instead of specifying the loss_weights parameter?

2) In the define_gan() function, why was there even a need to specify loss_weights parameter over there?

Start Machine Learning

Jason Brownlee August 3, 2020 at 5:45 am #

REPLY

Thanks.

Perhaps try it an see.

We do se a loss_weights for the gan.

Dhruv Agarwal August 4, 2020 at 1:14 am #

REPLY

Ok , i will try reducing the learning rate instead of specifiying the loss_weights parameter in the define_discriminator(). But i am sorry, but i still do not get the answer of the second question, i.e, why do we need to s function.

Jason Brownlee August 4, 2020 at 6:42 am #

To match the implementation described in the paper.

It has the effect of giving most attention to L1 and a tiny bit of attention to cross en

This is explained in that section of the tutorial, perhaps re-read?

Amine Zera August 4, 2020 at 9:35 pm #

REPLY

Hello sir, thank you for the great tutorial !
I am new to Machine Learning,
I want to change the clothings of people in images or videos. So i should train pix2pix on a clothes dataset ?
The second question is that i dont want to change anything else in the image except the clothes, so if i apply pix2pix on the image it will change everything, how can i target only clothing in a image ?
Thank's again for your great work !

Jason Brownlee August 5, 2020 at 6:11 am #

REPLY

Perhaps try it and see how well it can do.

Alex Westcott August 19, 2020 at 11:04 pm #

REPLY

Hi,

I have trained the exact model outlined in the tutorial with the same data-set quite a few times and the losses of the discriminator are always consistently 0.000 after around 5000 steps. Looking at the loss to more significant figures, shows that the loss is greater than zero, hence, when you state that, if the discriminator loss stays at zero for a long time then there is training failur

The generator still improves after the discriminator loss states 0.000, however I presume that impact on the training of the generator.

Thank you for the great tutorial, it helped a lot!

Jason Brownlee August 20, 2020 at 6:43 am #

Zero loss indicates a failure mode:
https://machinelearningmastery.com/practical-guide-to-gan-failure-modes/

Recall that GANs do not converge:
https://machinelearningmastery.com/faq/single-faq/why-is-my-gan-not-converging

Are you saving models along the way during training?
Are you able to inspect the progress of training, does it get good then go bad or is it bad the entire time?

Alex Westcott August 20, 2020 at 6:02 pm #

REPLY

I am saving the model every 5 epochs, and the predicted images do improve slightly during training, and by the end look reasonably good, (I presume that the discriminator hasn't had an impact on the quality and it is just the generator improving by itself).The losses of both the discriminator and generator decrease to start with, but the discriminator slowly decreases to 0 and the generator stays pretty low (between 1 and 5).

I have assumed that the discriminator is too good at determining the real and fake images, as I have removed a few layers from it and it's loss doesn't decay to 0 during training.

Jason Brownlee August 21, 2020 at 6:26 am #

REPLY

Interesting. Thanks for sharing Alex.

Reshma Jindal August 20, 2020 at 1:51 am #

I have around 3700 images to train on.

Can you roughly guide for the hyperparameters(like n_epochs,n_batch to be set as I'm encou
Please help in resolving it.

/home/reshmajindal/.local/lib/python3.6/site-packages/keras/engine/training.py:490: UserWar
collected trainable weights, did you set `model.trainable` without calling `model.compile` aft
'Discrepancy between trainable weights and collected trainable'
Killed

Reshma Jindal August 20, 2020 at 1:53 am #

REPLY

https://machinelearningmastery.com/faq/single-faq/why-does-the-code-in-the-tutorial-not-work-for-me/
I even tried all of this, but didn't work.

Jason Brownlee August 20, 2020 at 6:49 am #

REPLY

We cannot know the best way to configure the model, instead we must use experiments to tune and discover what configuration works best for a given dataset.

These tutorials will teach you:
https://machinelearningmastery.com/start-here/#gans

Awadelrahman M. A. Ahmed August 31, 2020 at 2:33 am #

REPLY

Thanks for this GREAT detailed tutorial. One question I have in mind is how to adapt
training/validation images have different height and width values?

Jason Brownlee August 31, 2020 at 6:17 am #

You're welcome.

Typically images are all resized to the same width and height expected by the model.

Awadelrahman M. A. Ahmed September 2, 2020 at 6:17 am #

REPLY

resizing is a bit flexible term ☺
cropping big images leads to loosing some information. enlarging small images might lead to blurry images. Super-resolution is computationally expensive and needs auxiliary models. What do you think the good way to "resize" images to work properly with this model ?

Jason Brownlee September 2, 2020 at 6:35 am #

REPLY

I recommend exploring many diffrent approaches and discover what works best for your specific project.

Awadelrahman M. A. Ahmed September 2, 2020 at 9:02 am #

YES!! the best way to find out is by doing it! this why I feel addicted to this machinelearningmastery :p

Jason Brownlee September 2, 2020 at 1:29 pm #

Thanks!

Nalin Nagar September 9, 2020 at 11:36 am #

Is there a way to input your own image? I haven't seen any demonstrations that are a
myself but to no avail.

Jason Brownlee September 9, 2020 at 1:34 pm #

Yes, the last part of the tutorial shows this.

REPLY

Harry September 9, 2020 at 4:10 pm #

Hi everyone. Thank you for super guideline for implementation. I have one question. Can i generate 1024×1024 px image by using pix2pix-GAN?

Start Machine Learning

Jason Brownlee September 10, 2020 at 6:22 am #

REPLY

Perhaps try scaling up the model for large images and see what kind of results you get.

I would expect quality to fall off. It might be easier with a model based on the progressive-growing architecture.

Harry September 9, 2020 at 4:23 pm #

REPLY

By the way, my dataset image size is smaller than 1024px

Bidesh Sengupta September 14, 2020 at 2:52 pm #

Hi!

It is a really good tutorial. I wish to apply this concept to my work. But I want to give some nun
image as input and wish to get the image as output.

Can you guide me on how to change the code to implement this? Is it at all possible?

Jason Brownlee September 15, 2020 at 5:17 am #

REPLY

Thanks.

Perhaps take a look at some alternate GANs like conditional GAN or InfoGan:
https://machinelearningmastery.com/how-to-develop-an-information-maximizing-generative-adversarial-network-infogan-in-keras/

Manohar Sai October 5, 2020 at 10:56 pm #

REPLY

Thanks for this great tutorial.
Both losses for the discriminator has gone to zero in the first 100 epochs.
Can you help me?

Jason Brownlee October 6, 2020 at 6:51 am #

REPLY

Perhaps restart training and stop once the generated images are good enough.

Manohar October 5, 2020 at 11:18 pm #

REPLY

Great tutorial sir.
I have my both discriminator loss heading to zero, in the first 200 steps. I cannot solve my issu
with the version?

Jason Brownlee October 6, 2020 at 6:52 am #

Perhaps try changing/tuning the learning hyperparameters of the model.

Perhaps try some of the suggestions here:
https://machinelearningmastery.com/how-to-code-generative-adversarial-network-hacks/

Manohar October 6, 2020 at 7:17 pm #

REPLY

Thanks for the reply sir. I will try that.
But why shouldn't it work if the code is same as above?

Start Machine Learning

Jason Brownlee October 7, 2020 at 6:43 am #

REPLY

The model uses a stochastic learning algorithm, you may need to run the example a few times to get a good result.

you can learn more about this here:
https://machinelearningmastery.com/faq/single-faq/why-do-i-get-different-results-each-time-i-run-the-code

Mick October 28, 2020 at 8:06 pm #

REPLY

Great tutorial!

I am trying to apply this architecture to a MRI image-to-image translation task. I have two que
1) After slicing the MRI data to 2D slices. Do I need to convert the NIFTI-files to JPEG or can
array)?
2) MRI images are grayscale whereas the example code in this tutorial uses RGB images. WI
deal with grayscale images?

Thanks Jason.

Jason Brownlee October 29, 2020 at 8:00 am #

The model takes image data as numpy arrays. I don't know if converting data to jpeg first is required for your data.

Yes, you can adapt the model for grayscale images, e.g. change the number of channels for input images to D() and output from G().

Adrien November 2, 2020 at 7:08 am #

REPLY

Jason,

What an incredible article. I reproduced your methodology on a research project on mechanical networks, where the model learns to draw mechanical linkages between parts of the system. It works perfectly, despite a small sample c

I re-used one of your images (the Unet architecture of the Generator) on a blog post I made on Medium, carefully citing your article as source and your work as reference. You can check it out here:
https://adriensaremi.medium.com/develop-a-image-to-image-translation-model-to-capture-local-interactions-in-mechanical-networks-9c2f45230849

I wanted to make sure you approved the re-use of the image in question. Thanks again for you work here and more broadly on Machine Learning Mastery.

REPLY

Jason Brownlee November 2, 2020 at 7:52 am #

Thanks.

That's fine, well done on your post!

ZiZi November 11, 2020 at 2:13 am #

Thank you for your great tutorial
I read a few posts about GANs and i realized GANs applyed in square images. is it right? can

Jason Brownlee November 11, 2020 at 6:50 am #

Yes, they can, bit square is much simpler.

REPLY

Rekka Mastouri December 1, 2020 at 11:04 pm #

Thank you for your great tutorial.

please how can I use GAN for deformable image registration?

Thanks Jason.

Start Machine Learning

Jason Brownlee December 2, 2020 at 7:44 am #

REPLY

You're welcome.

Perhaps start by checking the literature for existing approaches and try them first.

Zaineb December 2, 2020 at 11:15 pm #

REPLY

Hi,
Hope you are doing good.

I have tried your code and it works perfectly well.
I need to know, how about testing this module on a separate dataset,because i have found ou
include testing dataset also.

If i use a part of validation dataset (and call it my test dataset) on saved model (e..g model_1
test dataset, the segmentation results are not desirable.

I would be glad if you can shed some light on this. Also please tell me, is there any way that t
there any reference that signifies that testing pix2pix for image to image translation is not a g

Thanks

Jason Brownlee December 3, 2020 at 8:18 am #

REPLY

Thanks.

Sorry, I don't have an example of combining GAN output with a predictive model – I don't think I can give you good off the cuff advice on the topic. Perhaps check the literature.

Michal December 16, 2020 at 12:15 pm #

REPLY

Start Machine Learning

Hey, really well explained, good job!
I have implemented similar cGAN for b&w image colorization. It is very hard to train, and somehow after many, many epochs on big datasets I got some ‘good enough’ results, but I wonder how can I measure accuracy for translated images?

Also during training and after finishing it my cGAN is resulting in very big Losses of gen like 10.0 and 2.0 at the end of training. Disctiminator’s loss is near 0 and peaking sometimes to even 3 or 5. How can I measure accuracy of trained model or during training?
Thanks

Jason Brownlee December 16, 2020 at 1:41 pm #

REPLY

Thanks.

Good question, this may give you some ideas of how to evaluate a GAN model:
https://machinelearningmastery.com/how-to-evaluate-generative-adversarial-networks/

Muhammad Gulfam December 27, 2020 at 7:34 am #

Hi Jason,
Thank you very much for detailed explanation with examples. It is very helpful.
I am trying to edit the code through notepad++ but it is giving me indentation error. Seems like
Can you please tell me what IDE or editor you used?
Apologies for a silly question.

Jason Brownlee December 27, 2020 at 9:26 am #

REPLY

You’re welcome.

This will help you copy the code correctly:
https://machinelearningmastery.com/faq/single-faq/how-do-i-copy-code-from-a-tutorial

Start Machine Learning

Gavin December 28, 2020 at 6:01 am #

REPLY

Amazing tutorial, even more impressive that you've responded to every comment over year later! Quick question: you said that if either discriminant loss plateaus at 0 for an extended period of time that it has most likely failed and should be restarted. I am running it for the third time and both have landed on zero again, am I doing something wrong? Anything I can do to improve chances of it succeeding or just keep trying? (P.S. I am using different images and am using 2000 images as opposed to your ~1100 (still 100 epochs) but I assume that this does not affect the base of the model). Thanks in advance.

Jason Brownlee December 28, 2020 at 6:06 am #

REPLY

Thanks!

Sorry to hear that.

Perhaps try fewer epochs?
Perhaps try changing other learning hyperparameters?
Perhaps try adjusting the architecture?
Perhaps try some of the ideas here:
https://machinelearningmastery.com/how-to-code-generative-adversarial-network-hacks/

Gavin December 30, 2020 at 10:34 am #

Just an update, tried some things you suggested, in the article as well as in the well appreciated comment, not much changed. Let it run just to see what would happen and even though the model read 0 for both discriminators for more than 6 epochs, it still gave me decent results, so I'm happy. Thanks for the amazing article and the helpful advice, will definitely be reading up on some of your other articles.

Jason Brownlee December 30, 2020 at 10:44 am #

REPLY

Thanks for the update and well done!

Eric January 2, 2021 at 4:51 am #

REPLY

Hi, I love your whole blog and tutorials!

Just a question, Is it possible to train a model that uses 2 source images for one target?
For example from a traditional satellite image + an Infra Red (IR) image recreate the corresponding map?

Thanks a lot

Jason Brownlee January 2, 2021 at 6:27 am #

REPLY

Thanks!

I don't see why not. I expect there are papers on exactly this – I recommend seeking then

Wass January 3, 2021 at 7:54 am #

Thank you very much for the amazing tutorial!
My question is if it is possible to continue training from a saved model ? what would be the inp

Jason Brownlee January 3, 2021 at 1:29 pm #

REPLY

You're welcome.

Yes, you can load the saved model and continue training. You can use the same code as the first round of training as a starting point.

D.GHOSH January 11, 2021 at 6:02 am #

REPLY

Is this model applicable to generate super resolution data?

REPLY

Jason Brownlee January 11, 2021 at 6:23 am #

No, I believe there are more specialized models for that problem described in the literature.

REPLY

Muhammad Gulfam January 12, 2021 at 8:38 am #

Can you please share the link of some articles for those specialized models? for generating super resolution data

Jason Brownlee January 12, 2021 at 10:33 am #

You can search for papers on the topic here:
https://scholar.google.com/

Muhammad Gulfam January 12, 2021 at 8:06 am #

What is the significance of converting the pixel values from [0, 255] to [-1, 1]?
Is it because of the tanh activation function being used in the generator model for the last layer?
This architecture can be used to matrix to matrix mapping as well. but a matrix might have pixel (arr[row, col]) values as real values (from [0, inf] instead if [0, 255]). In that case, what would you suggest for transformation (to [-1, 1])? Should that still be done?

Thanks,
Apologies for multiple questions.

Jason Brownlee January 12, 2021 at 10:31 am #

Yes, exactly.

Yes, it is standard practice to use tanh for the output layer of gan generator models and to scale data to match the distribution of the activation function.

Muhammad Gulfam January 12, 2021 at 5:05 pm #

REPLY

Thank you very much. I appreciate the responses.

Jason Brownlee January 13, 2021 at 6:10 am #

You're welcome.

Muhammad Gulfam January 15, 2021 at 5:11 am #

I have noticed that in the code that the discriminator model is being compiled and the
model is not being compiled. generator is being saved. Whenever I load the generator model
"No training configuration found in save file: the model was *not* compiled. Compile it manual
Can you please guide me if it can affect model's performance? seems like my models are not
After googling it I got a perception that it is just a warning but still wanted to check with you.

Thanks

Jason Brownlee January 15, 2021 at 5:59 am #

REPLY

No need as we are not training it directly. You can ignore the warning.

Muhammad Gulfam January 15, 2021 at 1:25 pm #

REPLY

Thank you for the response.

WinK January 21, 2021 at 2:43 pm #

REPLY

I am fan of your site. Always thanks for your great article.

I would like to ask loss function that you utilize in the logical gan model. In your code block, 2 loss function was used in define_gan function.

model.compile(loss=['binary_crossentropy', 'mae'], optimizer=opt, loss_weights=[1,100])

If I understand correctly, 'mae' takes labels (true and predicted labels) instead of images. But

$L1(G) = E_{x,y,z}[\|y - G(x,z)\|]$

The output of G model is image and their loss function is defined based on differences betwe
labels.

Is it the same effect with labels instead of using images?

Jason Brownlee January 22, 2021 at 7:15 am #

Yes, MAE is the L1 norm between image pixels.

WinK January 25, 2021 at 12:54 pm #

REPLY

Thank you for your answer.

Muhammad Gulfam February 1, 2021 at 12:02 pm #

Start Machine Learning

Hi Dr. Brownlee,

In your last version there was a line in the define_gan method:

make weights in the discriminator not trainable

d_model.trainable = False

my question is that it the discriminator is not trainable then how will it improve?

In current version of your code you have replaced it by following lines:

make weights in the discriminator not trainable

for layer in d_model.layers:

if not isinstance(layer, BatchNormalization):

layer.trainable = False

if the weights are not trainable then how will discriminator learn and get better, and contribute to make the generator better?

my understanding was that weights are the ones that are supposed to be trained in the training process. Please correct me if I am wrong.

Apologies as I am not an expert. I am learning through your articles and other stuff.

Thanks in advance.

Jason Brownlee February 1, 2021 at 1:48 pm #

The D is only not trainable when part of the composite model. This is called layer

model.

REPLY

Muhammad Gulfam February 2, 2021 at 2:30 am #

Thank you so much Dr. Brownlee.

REPLY

Glenn Q February 16, 2021 at 10:23 am #

Hi Dr. Brownlee, if I want to have a higher learning rate for the discriminator and a lower one for the generator, say 2e^-4 for discriminator and 1e^-4 for the generator, should I just change the learning rate setting of the composite model?

Jason Brownlee February 16, 2021 at 1:38 pm #

REPLY

Yes, the composite model is used to update the generator.

Let me know how you go with your approach.

Alice February 17, 2021 at 8:19 pm #

REPLY

Hi Jason,

Thank you for your great tutorial.
I just want to ask you one question: why during the inference we have to keep the batch norm
I understand that the dropout is performed to add some noise, but I thought it was necessary

Moreover, I have performed the training with a batch size = 1 and in the prediction phase I ha
images of dimension [N, 256, 256, 3] and the results were very different. Using a batch size =
think that this is correlated to the adoption of BN in training modality.

Thank you for your time

Jason Brownlee February 18, 2021 at 5:14 am #

No. Batchnorm and dropout can are flipped to inference mode. Batchnorm will use learned mu and sigma and dropout will stop dropping out.

Alice February 23, 2021 at 10:16 pm #

REPLY

but the flag training is set to True for both BN and dropout, I think that this flag makes them work as during the training phase

anarchitect February 28, 2021 at 5:21 am #

REPLY

Hi,

Is it possible to plot losses in realtime? I couldn't manage to do it. Could you please help me?

Jason Brownlee February 28, 2021 at 5:42 am #

REPLY

Perhaps via tensorboard?

Lin April 10, 2021 at 4:55 pm #

Hello, thank you for the sharing.
I'd like to know what is d1[0.362] d2[0.405] g[78.143] each loss value's meaning?
Does it mean that is fake when discriminator's loss value close to zero?
And what is the composite's loss calue mean?

Jason Brownlee April 11, 2021 at 4:49 am #

The values are hard to interpet.

Nevertheless d1 and d2 are the discriminators loss on real samples and fake samples (of "B") respectively, and g is loss of the composite model on real samples.

This may help you interpret cross entropy more generally:
https://machinelearningmastery.com/cross-entropy-for-machine-learning/

Nada April 27, 2021 at 1:39 pm #

Hi Jason,

Thank you very much for this informative article.

I have a question about a good GAN model to create more synthesis images from a small set of medical images? Is styleGaN good for this problem?

Jason Brownlee April 28, 2021 at 6:00 am #

REPLY

Perhaps trial a few methods and discover what works well or best for your dataset.

Rishabh singh April 27, 2021 at 11:53 pm #

Hey,

If possible , can you please share the .h5 model after complete training. As I am trying but not
power.
I have tried on colab too, but gets stopped after some time.

Jason Brownlee April 28, 2021 at 6:02 am #

Sorry, I cannot share saved models.

Leave a Reply

Start Machine Learning

Name (required)

Email (will not be published) (required)

Website

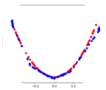
SUBMIT COMMENT

Welcome!
I'm *Jason Brownlee* PhD
and I **help developers** get results with **machine learning**.
Read more

Never miss a tutorial:

Picked for you:

How to Develop a Pix2Pix GAN for Image-to-Image Translation



How to Develop a 1D Generative Adversarial Network From Scratch in Keras

How to Develop a CycleGAN for Image-to-Image Translation with Keras

How to Develop a Conditional GAN (cGAN) From Scratch

How to Train a Progressive Growing GAN in Keras for Synthesizing Faces

Start Machine Learning