오토인코더

In [1]:

```
import keras
from keras import layers
from keras import backend as K
from keras.models import Model
import numpy as np

img_shape = (28, 28, 1)
batch_size = 16
latent_dim = 2

input_img = keras.Input(shape=img_shape)

x = layers.Conv2D(32, 3, padding='same', activation='relu')(input_img)
x = layers.Conv2D(64, 3, padding='same', activation='relu', strides=(2, 2))(x)
x = layers.Conv2D(64, 3, padding='same', activation='relu')(x)
x = layers.Conv2D(64, 3, padding='same', activation='relu')(x)
shape_before_flattening = K.int_shape(x)

x = layers.Flatten()(x)
x = layers.Dense(32, activation='relu')(x)

z_mean = layers.Dense(latent_dim)(x)
z_log_var = layers.Dense(latent_dim)(x)
```

```
Using TensorFlow backend.

WARNING:tensorflow:From C:WProgramDataWAnaconda3WlibWsite-packagesWtensorflowWpyth
onWframeworkWop_def_library.py:263: colocate_with (from tensorflow.python.framewor
k.ops) is deprecated and will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
```

In [2]:

```
def sampling(args):
    z_mean, z_log_var = args
    epsilon = K.random_normal(shape=(K.shape(z_mean)[0], latent_dim),
                              mean=0., stddev=1.)
    return z_mean + K.exp(z_log_var) * epsilon

z = layers.Lambda(sampling)([z_mean, z_log_var])
```

In [3]:

```
decoder_input = layers.Input(K.int_shape(z)[1:])

x = layers.Dense(np.prod(shape_before_flattening[1:]), activation='relu')(decoder_input)
x = layers.Reshape(shape_before_flattening[1:])(x)
x = layers.Conv2DTranspose(32, 3, padding='same', activation='relu', strides=(2, 2))(x)
x = layers.Conv2D(1, 3, padding='same', activation='sigmoid')(x)

decoder = Model(decoder_input, x)

z_decoded = decoder(z)
```

In [4]:

```
class CustomVariationalLayer(keras.layers.Layer):

    def vae_loss(self, x, z_decoded):
        x = K.flatten(x)
        z_decoded = K.flatten(z_decoded)
        xent_loss = keras.metrics.binary_crossentropy(x, z_decoded)
        kl_loss = -5e-4 * K.mean(
            1 + z_log_var - K.square(z_mean) - K.exp(z_log_var), axis=-1)
        return K.mean(xent_loss + kl_loss)

    def call(self, inputs):
        x = inputs[0]
        z_decoded = inputs[1]
        loss = self.vae_loss(x, z_decoded)
        self.add_loss(loss, inputs=inputs)
        return x

y = CustomVariationalLayer()([input_img, z_decoded])
```

In [5]:

```
from keras.datasets import mnist

vae = Model(input_img, y)
vae.compile(optimizer='rmsprop', loss=None)
vae.summary()

(x_train, _), (x_test, y_test) = mnist.load_data()

x_train = x_train.astype('float32') / 255.
x_train = x_train.reshape(x_train.shape + (1,))
x_test = x_test.astype('float32') / 255.
x_test = x_test.reshape(x_test.shape + (1,))

vae.fit(x=x_train, y=None,
        shuffle=True,
        epochs=10,
        batch_size=batch_size,
        validation_data=(x_test, None))
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   Connected to
=================================================================
input_1 (InputLayer)         (None, 28, 28, 1)         0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 28, 28, 32)        320       input_1[0][0]
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 14, 14, 64)        18496     conv2d_1[0][0]
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 14, 14, 64)        36928     conv2d_2[0][0]
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 14, 14, 64)        36928     conv2d_3[0][0]
_________________________________________________________________
flatten_1 (Flatten)          (None, 12544)             0         conv2d_4[0][0]
_________________________________________________________________
dense_1 (Dense)              (None, 32)                401440    flatten_1[0][0]
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 66        dense_1[0][0]
_________________________________________________________________
dense_3 (Dense)              (None, 2)                 66        dense_1[0][0]
_________________________________________________________________
lambda_1 (Lambda)            (None, 2)                 0         dense_2[0][0]
                                                                 dense_3[0][0]
_________________________________________________________________
model_1 (Model)              (None, 28, 28, 1)         56385     lambda_1[0][0]
_________________________________________________________________
custom_variational_layer_1 (Cus [(None, 28, 28, 1),  0         input_1[0][0]
                                                                 model_1[1][0]
=================================================================
Total params: 550,629
Trainable params: 550,629
Non-trainable params: 0
_________________________________________________________________
WARNING:tensorflow:From C:WProgramDataWAnaconda3WlibWsite-packagesWtensorflowWpyth
onWopsWmath_ops.py:3066: to_int32 (from tensorflow.python.ops.math_ops) is deprecа
ted and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
Train on 60000 samples, validate on 10000 samples
Epoch 1/10
60000/60000 [==============================] - 28s 472us/step - loss: 280.7949 - v
al_loss: 0.1961
Epoch 2/10
60000/60000 [==============================] - 25s 419us/step - loss: 0.1928 - val
_loss: 0.1896
Epoch 3/10
```

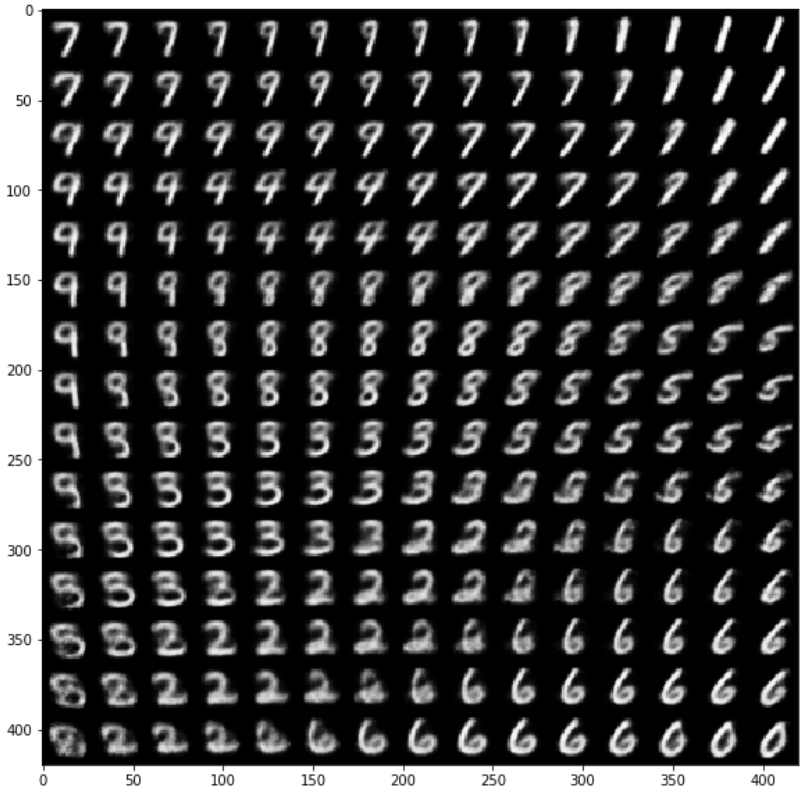
```
60000/60000 [==============================] - 25s 416us/step - loss: 0.1886 - val
_loss: 0.1880
Epoch 4/10
60000/60000 [==============================] - 25s 415us/step - loss: 0.1862 - val
_loss: 0.1856
Epoch 5/10
60000/60000 [==============================] - 25s 417us/step - loss: 0.1845 - val
_loss: 0.1835
Epoch 6/10
60000/60000 [==============================] - 25s 420us/step - loss: 0.1834 - val
_loss: 0.1837
Epoch 7/10
60000/60000 [==============================] - 25s 416us/step - loss: 0.1826 - val
_loss: 0.1825
Epoch 8/10
60000/60000 [==============================] - 25s 416us/step - loss: 0.1819 - val
_loss: 0.1819
Epoch 9/10
60000/60000 [==============================] - 25s 417us/step - loss: 0.1813 - val
_loss: 0.1811
Epoch 10/10
60000/60000 [==============================] - 25s 417us/step - loss: 0.1807 - val
_loss: 0.1808
```

Out[5]:

```
<keras.callbacks.History at 0x1cf82859208>
```

In [8]:

```
import matplotlib.pyplot as plt
from scipy.stats import norm

n = 15
digit_size = 28
figure = np.zeros((digit_size * n, digit_size * n))
grid_x = norm.ppf(np.linspace(0.05, 0.95, n))
grid_y = norm.ppf(np.linspace(0.05, 0.95, n))

for i, yi in enumerate(grid_x):
    for j, xi in enumerate(grid_y):
        z_sample = np.array([[xi, yi]])
        z_sample = np.tile(z_sample, batch_size).reshape(batch_size, 2)
        x_decoded = decoder.predict(z_sample, batch_size=batch_size)
        digit = x_decoded[0].reshape(digit_size, digit_size)
        figure[i * digit_size: (i + 1) * digit_size,
               j * digit_size: (j + 1) * digit_size] = digit

plt.figure(figsize=(10, 10))
plt.imshow(figure, cmap='Greys_r')
plt.show()
```



In [18]:

```
latent_dim = 32
height = 32
width = 32
channels = 3

generator_input = keras.Input(shape=(latent_dim,))

x = layers.Dense(128 * 16 * 16)(generator_input)
x = layers.LeakyReLU()(x)
x = layers.Reshape((16, 16, 128))(x)

x = layers.Conv2D(256, 5, padding='same')(x)
x = layers.LeakyReLU()(x)

x = layers.Conv2DTranspose(256, 4, strides=2, padding='same')(x)
x = layers.LeakyReLU()(x)

x = layers.Conv2D(256, 5, padding='same')(x)
x = layers.LeakyReLU()(x)
x = layers.Conv2D(256, 5, padding='same')(x)
x = layers.LeakyReLU()(x)

x = layers.Conv2D(channels, 7, activation='tanh', padding='same')(x)
generator = keras.models.Model(generator_input, x)
generator.summary()
```

GAN

생성자

Layer (type)	Output Shape	Param #
input_8 (InputLayer)	(None, 32)	0
dense_9 (Dense)	(None, 32768)	1081344
leaky_re_lu_13 (LeakyReLU)	(None, 32768)	0
reshape_4 (Reshape)	(None, 16, 16, 128)	0
conv2d_15 (Conv2D)	(None, 16, 16, 256)	819456
leaky_re_lu_14 (LeakyReLU)	(None, 16, 16, 256)	0
conv2d_transpose_3 (Conv2DTr	(None, 32, 32, 256)	1048832
leaky_re_lu_15 (LeakyReLU)	(None, 32, 32, 256)	0
conv2d_16 (Conv2D)	(None, 32, 32, 256)	1638656
leaky_re_lu_16 (LeakyReLU)	(None, 32, 32, 256)	0
conv2d_17 (Conv2D)	(None, 32, 32, 256)	1638656
leaky_re_lu_17 (LeakyReLU)	(None, 32, 32, 256)	0
conv2d_18 (Conv2D)	(None, 32, 32, 3)	37635

Total params: 6,264,579
Trainable params: 6,264,579
Non-trainable params: 0

판별자

In [19]:

```
discriminator_input = layers.Input(shape=(height, width, channels))
x = layers.Conv2D(128, 3)(discriminator_input)
x = layers.LeakyReLU()(x)
x = layers.Conv2D(128, 4, strides=2)(x)
x = layers.LeakyReLU()(x)
x = layers.Conv2D(128, 4, strides=2)(x)
x = layers.LeakyReLU()(x)
x = layers.Conv2D(128, 4, strides=2)(x)
x = layers.LeakyReLU()(x)
x = layers.Flatten()(x)

x = layers.Dropout(0.4)(x)

x = layers.Dense(1, activation='sigmoid')(x)

discriminator = keras.models.Model(discriminator_input, x)
discriminator.summary()

discriminator_optimizer = keras.optimizers.RMSprop(lr=0.0008, clipvalue=1.0, decay=1e-8)
discriminator.compile(optimizer=discriminator_optimizer, loss='binary_crossentropy')
```

Layer (type)	Output Shape	Param #
input_9 (InputLayer)	(None, 32, 32, 3)	0
conv2d_19 (Conv2D)	(None, 30, 30, 128)	3584
leaky_re_lu_18 (LeakyReLU)	(None, 30, 30, 128)	0
conv2d_20 (Conv2D)	(None, 14, 14, 128)	262272
leaky_re_lu_19 (LeakyReLU)	(None, 14, 14, 128)	0
conv2d_21 (Conv2D)	(None, 6, 6, 128)	262272
leaky_re_lu_20 (LeakyReLU)	(None, 6, 6, 128)	0
conv2d_22 (Conv2D)	(None, 2, 2, 128)	262272
leaky_re_lu_21 (LeakyReLU)	(None, 2, 2, 128)	0
flatten_3 (Flatten)	(None, 512)	0
dropout_2 (Dropout)	(None, 512)	0
dense_10 (Dense)	(None, 1)	513

Total params: 790,913
Trainable params: 790,913
Non-trainable params: 0

In [20]:

```
discriminator.trainable = False

gan_input = keras.Input(shape=(latent_dim,))
gan_output = discriminator(generator(gan_input))
gan = keras.models.Model(gan_input, gan_output)

gan_optimizer = keras.optimizers.RMSprop(lr=0.0004, clipvalue=1.0, decay=1e-8)
gan.compile(optimizer=gan_optimizer, loss='binary_crossentropy')
```

In [21]:

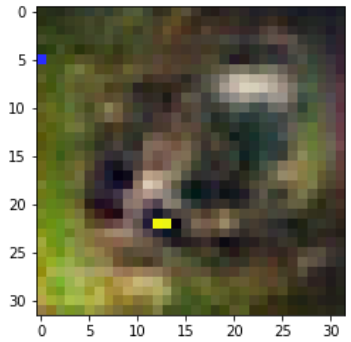
```
(x_train, y_train), (_, _) = keras.datasets.cifar10.load_data()

x_train = x_train[y_train.flatten() == 6]

x_train = x_train.reshape((x_train.shape[0],) + (height, width, channels)).astype('float32') / 2
55.

iterations = 10000
batch_size = 20
save_dir = './datasets/gan_images/'
if not os.path.exists(save_dir):
    os.mkdir(save_dir)

start = 0
for step in range(iterations):
    random_latent_vectors = np.random.normal(size=(batch_size, latent_dim))
    generated_images = generator.predict(random_latent_vectors)
    stop = start + batch_size
    real_images = x_train[start: stop]
    combined_images = np.concatenate([generated_images, real_images])

    labels = np.concatenate([np.ones((batch_size, 1)), np.zeros((batch_size, 1))])
    labels += 0.05 * np.random.random(labels.shape)

    d_loss = discriminator.train_on_batch(combined_images, labels)

    random_latent_vectors = np.random.normal(size=(batch_size, latent_dim))
    misleading_targets = np.zeros((batch_size, 1))
    a_loss = gan.train_on_batch(random_latent_vectors, misleading_targets)

    start += batch_size
    if start > len(x_train) - batch_size:
        start = 0

    if step % 100 == 0:
        gan.save_weights('gan.h5')

        print('스텝 %s에서 판별자 손실: %s' % (step, d_loss))
        print('스텝 %s에서 적대적 손실: %s' % (step, a_loss))

        img = image.array_to_img(generated_images[0] * 255., scale=False)
        img.save(os.path.join(save_dir, 'generated_frog' + str(step) + '.png'))

        img = image.array_to_img(real_images[0] * 255., scale=False)
        img.save(os.path.join(save_dir, 'real_frog' + str(step) + '.png'))
```

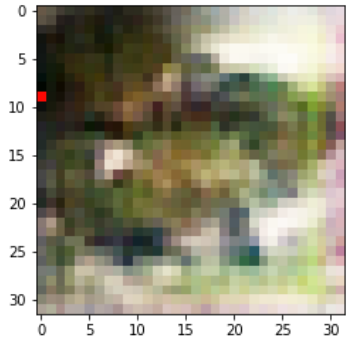
스텝 0에서 판별자 손실: 0.67389405
스텝 0에서 적대적 손실: 0.67728555
스텝 100에서 판별자 손실: 0.6869162
스텝 100에서 적대적 손실: 1.202899
스텝 200에서 판별자 손실: 0.70100033
스텝 200에서 적대적 손실: 0.7126546
스텝 300에서 판별자 손실: 0.7169893
스텝 300에서 적대적 손실: 0.76923954
스텝 400에서 판별자 손실: 0.69473076
스텝 400에서 적대적 손실: 0.737267
스텝 500에서 판별자 손실: 0.6941329
스텝 500에서 적대적 손실: 0.7298473
스텝 600에서 판별자 손실: 0.6963733
스텝 600에서 적대적 손실: 0.72651666
스텝 700에서 판별자 손실: 0.7072137
스텝 700에서 적대적 손실: 0.79943717
스텝 800에서 판별자 손실: 0.69327605
스텝 800에서 적대적 손실: 0.7437229
스텝 900에서 판별자 손실: 0.6961517
스텝 900에서 적대적 손실: 0.75318635
스텝 1000에서 판별자 손실: 0.6959578
스텝 1000에서 적대적 손실: 0.7573913
스텝 1100에서 판별자 손실: 0.69207036
스텝 1100에서 적대적 손실: 0.7491814
스텝 1200에서 판별자 손실: 0.67976844
스텝 1200에서 적대적 손실: 0.8084177
스텝 1300에서 판별자 손실: 0.69770163
스텝 1300에서 적대적 손실: 0.7527974
스텝 1400에서 판별자 손실: 0.6873482
스텝 1400에서 적대적 손실: 0.77956927
스텝 1500에서 판별자 손실: 0.69123936
스텝 1500에서 적대적 손실: 0.74569875
스텝 1600에서 판별자 손실: 0.67613286
스텝 1600에서 적대적 손실: 0.8407283
스텝 1700에서 판별자 손실: 0.6909586
스텝 1700에서 적대적 손실: 0.7334469
스텝 1800에서 판별자 손실: 0.70976275
스텝 1800에서 적대적 손실: 0.75621605
스텝 1900에서 판별자 손실: 0.6895846
스텝 1900에서 적대적 손실: 0.54084903
스텝 2000에서 판별자 손실: 0.6880598
스텝 2000에서 적대적 손실: 0.7652202
스텝 2100에서 판별자 손실: 0.6976644
스텝 2100에서 적대적 손실: 0.7682227
스텝 2200에서 판별자 손실: 0.68904305
스텝 2200에서 적대적 손실: 0.75449944
스텝 2300에서 판별자 손실: 0.6823585
스텝 2300에서 적대적 손실: 1.4339652
스텝 2400에서 판별자 손실: 0.7128985
스텝 2400에서 적대적 손실: 0.77560997
스텝 2500에서 판별자 손실: 0.68881327
스텝 2500에서 적대적 손실: 0.75204
스텝 2600에서 판별자 손실: 0.7401123
스텝 2600에서 적대적 손실: 0.74167556
스텝 2700에서 판별자 손실: 0.69787633
스텝 2700에서 적대적 손실: 0.7567729
스텝 2800에서 판별자 손실: 0.6893015
스텝 2800에서 적대적 손실: 0.72939366
스텝 2900에서 판별자 손실: 0.72084755
스텝 2900에서 적대적 손실: 0.7448244
스텝 3000에서 판별자 손실: 0.6906905
스텝 3000에서 적대적 손실: 0.722713
스텝 3100에서 판별자 손실: 0.6938766
스텝 3100에서 적대적 손실: 0.75551546
스텝 3200에서 판별자 손실: 0.6925848
스텝 3200에서 적대적 손실: 0.76472884
스텝 3300에서 판별자 손실: 0.6875025
스텝 3300에서 적대적 손실: 0.7547171
스텝 3400에서 판별자 손실: 0.7406846
스텝 3400에서 적대적 손실: 0.7609933
스텝 3500에서 판별자 손실: 0.7102865
스텝 3500에서 적대적 손실: 0.7557062
스텝 3600에서 판별자 손실: 0.69656676
스텝 3600에서 적대적 손실: 0.7479655
스텝 3700에서 판별자 손실: 0.69042337
스텝 3700에서 적대적 손실: 0.7343094
스텝 3800에서 판별자 손실: 0.6899109
스텝 3800에서 적대적 손실: 0.7540313
스텝 3900에서 판별자 손실: 0.71365976
스텝 3900에서 적대적 손실: 0.7256447
스텝 4000에서 판별자 손실: 0.68878037
스텝 4000에서 적대적 손실: 0.72575605
스텝 4100에서 판별자 손실: 0.6977674
스텝 4100에서 적대적 손실: 0.7699025
스텝 4200에서 판별자 손실: 0.68355215
스텝 4200에서 적대적 손실: 0.7590171
스텝 4300에서 판별자 손실: 0.6964672
스텝 4300에서 적대적 손실: 0.72968525
스텝 4400에서 판별자 손실: 0.69410485
스텝 4400에서 적대적 손실: 0.74912375
스텝 4500에서 판별자 손실: 0.67900145
스텝 4500에서 적대적 손실: 0.71069306
스텝 4600에서 판별자 손실: 0.69253474
스텝 4600에서 적대적 손실: 0.7448413
스텝 4700에서 판별자 손실: 0.67833114
스텝 4700에서 적대적 손실: 0.7914194
스텝 4800에서 판별자 손실: 0.6893244
스텝 4800에서 적대적 손실: 0.73302746
스텝 4900에서 판별자 손실: 0.70090854
스텝 4900에서 적대적 손실: 0.7736103
스텝 5000에서 판별자 손실: 0.6949382
스텝 5000에서 적대적 손실: 0.9099487
스텝 5100에서 판별자 손실: 0.6938237
스텝 5100에서 적대적 손실: 0.72613174
스텝 5200에서 판별자 손실: 0.6948127
스텝 5200에서 적대적 손실: 0.81342775
스텝 5300에서 판별자 손실: 0.7046625
스텝 5300에서 적대적 손실: 0.69168407
스텝 5400에서 판별자 손실: 0.704661
스텝 5400에서 적대적 손실: 0.76752204
스텝 5500에서 판별자 손실: 0.6973317
스텝 5500에서 적대적 손실: 0.74182194
스텝 5600에서 판별자 손실: 0.70051295
스텝 5600에서 적대적 손실: 0.7181288
스텝 5700에서 판별자 손실: 0.68522435
스텝 5700에서 적대적 손실: 0.72570467
스텝 5800에서 판별자 손실: 0.68131286
스텝 5800에서 적대적 손실: 0.6796528
스텝 5900에서 판별자 손실: 0.70924467
스텝 5900에서 적대적 손실: 0.8087176
스텝 6000에서 판별자 손실: 0.6951924
스텝 6000에서 적대적 손실: 0.77186704

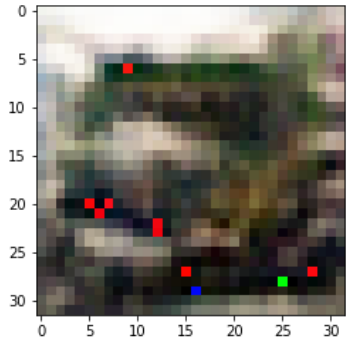
스텝 6100에서 판별자 손실: 0.701895
스텝 6100에서 적대적 손실: 0.75545484
스텝 6200에서 판별자 손실: 0.70053965
스텝 6200에서 적대적 손실: 0.75994676
스텝 6300에서 판별자 손실: 0.69953954
스텝 6300에서 적대적 손실: 0.7191021
스텝 6400에서 판별자 손실: 0.7089031
스텝 6400에서 적대적 손실: 0.7503309
스텝 6500에서 판별자 손실: 0.76820385
스텝 6500에서 적대적 손실: 4.6862025
스텝 6600에서 판별자 손실: 0.6955652
스텝 6600에서 적대적 손실: 0.727486
스텝 6700에서 판별자 손실: 0.69113827
스텝 6700에서 적대적 손실: 0.8438532
스텝 6800에서 판별자 손실: 0.6904483
스텝 6800에서 적대적 손실: 0.78808844
스텝 6900에서 판별자 손실: 0.7131284
스텝 6900에서 적대적 손실: 0.82017374
스텝 7000에서 판별자 손실: 0.6666603
스텝 7000에서 적대적 손실: 0.9003515
스텝 7100에서 판별자 손실: 0.69361717
스텝 7100에서 적대적 손실: 0.7785653
스텝 7200에서 판별자 손실: 0.6868307
스텝 7200에서 적대적 손실: 0.81473124
스텝 7300에서 판별자 손실: 0.72025526
스텝 7300에서 적대적 손실: 0.7955084
스텝 7400에서 판별자 손실: 0.69274586
스텝 7400에서 적대적 손실: 0.6999842
스텝 7500에서 판별자 손실: 0.67852086
스텝 7500에서 적대적 손실: 0.8024667
스텝 7600에서 판별자 손실: 0.6955534
스텝 7600에서 적대적 손실: 0.7643263
스텝 7700에서 판별자 손실: 0.706426
스텝 7700에서 적대적 손실: 0.9583293
스텝 7800에서 판별자 손실: 0.68668324
스텝 7800에서 적대적 손실: 0.7429688
스텝 7900에서 판별자 손실: 0.6912948
스텝 7900에서 적대적 손실: 0.93313426
스텝 8000에서 판별자 손실: 0.67487305
스텝 8000에서 적대적 손실: 0.8119267
스텝 8100에서 판별자 손실: 0.6870252
스텝 8100에서 적대적 손실: 0.67820966
스텝 8200에서 판별자 손실: 0.6949711
스텝 8200에서 적대적 손실: 0.8117798
스텝 8300에서 판별자 손실: 0.6674655
스텝 8300에서 적대적 손실: 0.9393929
스텝 8400에서 판별자 손실: 0.6923276
스텝 8400에서 적대적 손실: 0.80607784
스텝 8500에서 판별자 손실: 0.6677327
스텝 8500에서 적대적 손실: 0.8854497
스텝 8600에서 판별자 손실: 0.6777681
스텝 8600에서 적대적 손실: 0.79696536
스텝 8700에서 판별자 손실: 0.6784049
스텝 8700에서 적대적 손실: 0.78293407
스텝 8800에서 판별자 손실: 0.67968893
스텝 8800에서 적대적 손실: 0.77046967
스텝 8900에서 판별자 손실: 0.7200328
스텝 8900에서 적대적 손실: 0.8383153
스텝 9000에서 판별자 손실: 0.6625947
스텝 9000에서 적대적 손실: 0.69347984
스텝 9100에서 판별자 손실: 0.7076703
스텝 9100에서 적대적 손실: 0.74378866
스텝 9200에서 판별자 손실: 0.6830262
스텝 9200에서 적대적 손실: 1.0857048
스텝 9300에서 판별자 손실: 0.6886126
스텝 9300에서 적대적 손실: 0.7960647
스텝 9400에서 판별자 손실: 0.6828648
스텝 9400에서 적대적 손실: 0.74228036
스텝 9500에서 판별자 손실: 0.67550147
스텝 9500에서 적대적 손실: 0.79393995
스텝 9600에서 판별자 손실: 0.71815276
스텝 9600에서 적대적 손실: 0.7951985
스텝 9700에서 판별자 손실: 0.72488534
스텝 9700에서 적대적 손실: 0.983703
스텝 9800에서 판별자 손실: 0.67307127
스텝 9800에서 적대적 손실: 0.8518568
스텝 9900에서 판별자 손실: 0.7295101
스텝 9900에서 적대적 손실: 0.80139863

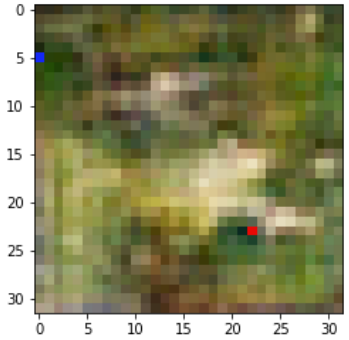
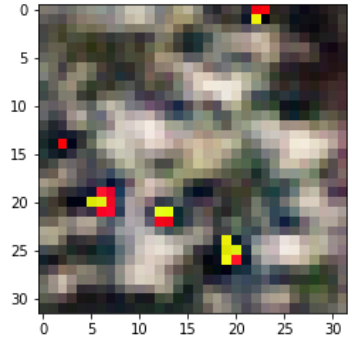
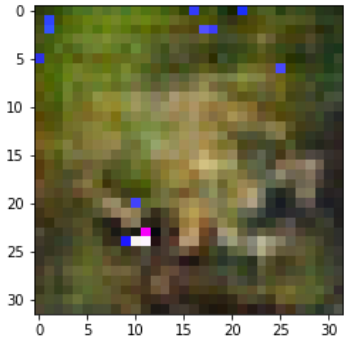
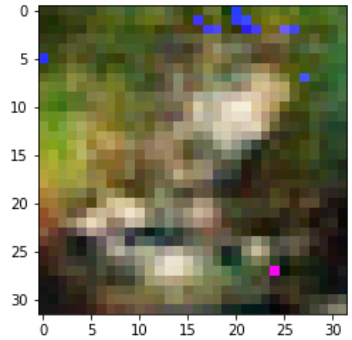
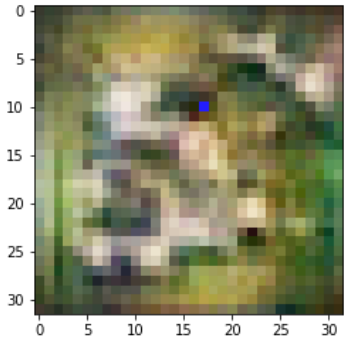
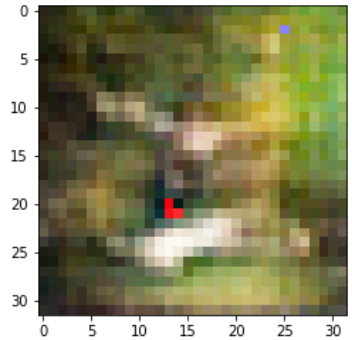
In [22]:

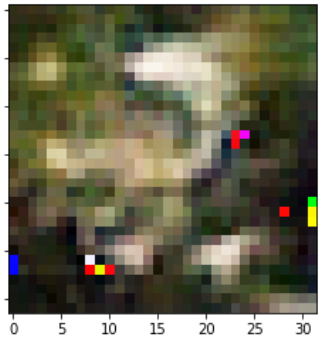
```
random_latent_vectors = np.random.normal(size=(10, latent_dim))

generated_images = generator.predict(random_latent_vectors)

for i in range(generated_images.shape[0]):
    img = image.array_to_img(generated_images[i] * 255., scale=False)
    plt.figure()
    plt.imshow(img)

plt.show()
```








0
5
10
15
20
25
30
0
5
10
15
20
25
30