In [2]:
```
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
import os
from keras.layers import Conv2D, Dense, MaxPooling2D, Flatten, Dropout
from PIL import Image
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
from keras.preprocessing.image import array_to_img
from keras import Sequential
from sklearn.model_selection import train_test_split
from keras.layers import Lambda, Input, Dense
from keras.models import Model
from keras.losses import mse, binary_crossentropy
from keras.utils import plot_model
from keras import backend as K
from keras.optimizers import Adam
```

```
Using TensorFlow backend.
```

In [3]:
```
manifest=pd.read_csv('/kaggle/input/pokemon-images-and-types/pokemon.csv')
y=pd.get_dummies(manifest.Type1)
manifest=pd.concat([manifest['Name'].reset_index(drop=True), y], axis=1)
names=[]
index=0

for dirname, _, filenames in os.walk('/kaggle/input/pokemon-images-and-types/images/images/'):
    for filename in filenames:
        file_path_i = os.path.join(dirname, filename)
        name_i = filename.replace('.png','')
        names.append({'Name':name_i,'ix':index})
        if index==0:
            x=load_img(file_path_i)
            x=img_to_array(x)
            x.shape=(1,120,120,3)
            x=x/255.
        else:
            xi=load_img(file_path_i)
            xi=img_to_array(xi)
            xi.shape=(1,120,120,3)
            xi=xi/255.
            x = np.concatenate((x,xi),axis=0)
        index+=1
        #print(index)
names=pd.DataFrame(names)
manifest=manifest.merge(names,how='left')
dropme=np.where(manifest.ix.isna())
manifest.dropna(inplace=True)
manifest = manifest.set_index('ix',drop=True).sort_index()
x= np.delete(x,dropme,axis=0)
y=manifest.drop('Name',axis=1).values
x_train = np.float32(x)
x_test=x_train
```

```
/opt/conda/lib/python3.7/site-packages/PIL/Image.py:968: UserWarning: Palette images with Transparency   expressed in bytes shoul
d be converted to RGBA images
  ' expressed in bytes should be converted ' +
```

In [4]:
```
x_train,x_test=x_train[:,:,:,0],x_train[:,:,:,0]
```

In [5]:
```
%matplotlib inline
```

In [6]:
```
def sampling(args):
    """Reparameterization trick by sampling from an isotropic unit Gaussian.

    # Arguments
        args (tensor): mean and log of variance of Q(z|X)

    # Returns
        z (tensor): sampled latent vector
    """

    z_mean, z_log_var = args
    batch = K.shape(z_mean)[0]
    dim = K.int_shape(z_mean)[1]
    # by default, random_normal has mean = 0 and std = 1.0
    epsilon = K.random_normal(shape=(batch, dim))
    return z_mean + K.exp(0.5 * z_log_var) * epsilon




image_size = x_train.shape[1]
original_dim = image_size * image_size
x_train = np.reshape(x_train, [-1, original_dim])
x_test = np.reshape(x_test, [-1, original_dim])
x_train = x_train.astype('float32') / 255
x_test = x_test.astype('float32') / 255

# network parameters
input_shape = (original_dim, )
intermediate_dim = 512
batch_size = 128
latent_dim = 5
epochs = 50000

# VAE model = encoder + decoder
# build encoder model
inputs = Input(shape=input_shape, name='encoder_input')
x = Dense(intermediate_dim, activation='relu')(inputs)
x = Dense(intermediate_dim, activation='relu')(x)
x = Dense(intermediate_dim, activation='relu')(x)
x = Dense(intermediate_dim, activation='relu')(x)
x = Dense(intermediate_dim, activation='relu')(x)
x = Dense(intermediate_dim, activation='relu')(x)
```

```python
z_mean = Dense(latent_dim, name='z_mean')(x)
z_log_var = Dense(latent_dim, name='z_log_var')(x)

# use reparameterization trick to push the sampling out as input
# note that "output_shape" isn't necessary with the TensorFlow backend
z = Lambda(sampling, output_shape=(latent_dim,), name='z')([z_mean, z_log_var])

# instantiate encoder model
encoder = Model(inputs, [z_mean, z_log_var, z], name='encoder')
encoder.summary()
plot_model(encoder, show_shapes=True)

# build decoder model
latent_inputs = Input(shape=(latent_dim,), name='z_sampling')
x = Dense(intermediate_dim, activation='relu')(latent_inputs)
x = Dense(intermediate_dim, activation='relu')(x)
x = Dense(intermediate_dim, activation='relu')(x)
x = Dense(intermediate_dim, activation='relu')(x)
x = Dense(intermediate_dim, activation='relu')(x)
x = Dense(intermediate_dim, activation='relu')(x)
outputs = Dense(original_dim, activation='sigmoid')(x)

# instantiate decoder model
decoder = Model(latent_inputs, outputs, name='decoder')
decoder.summary()
plot_model(decoder, show_shapes=True)

# instantiate VAE model
outputs = decoder(encoder(inputs)[2])
vae = Model(inputs, outputs, name='vae_mlp')

data = (x_test, x_test)

# VAE loss = mse_loss or xent_loss + kl_loss
reconstruction_loss = binary_crossentropy(inputs,
                                          outputs)

reconstruction_loss *= original_dim
kl_loss = 1 + z_log_var - K.square(z_mean) - K.exp(z_log_var)
kl_loss = K.sum(kl_loss, axis=-1)
kl_loss *= -0.5
vae_loss = K.mean(reconstruction_loss + kl_loss)
vae.add_loss(vae_loss)
```

```
Model: "encoder"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
encoder_input (InputLayer)      (None, 14400)        0
__________________________________________________________________________________________________
dense_1 (Dense)                 (None, 512)          7373312     encoder_input[0][0]
__________________________________________________________________________________________________
dense_2 (Dense)                 (None, 512)          262656      dense_1[0][0]
__________________________________________________________________________________________________
dense_3 (Dense)                 (None, 512)          262656      dense_2[0][0]
__________________________________________________________________________________________________
dense_4 (Dense)                 (None, 512)          262656      dense_3[0][0]
__________________________________________________________________________________________________
dense_5 (Dense)                 (None, 512)          262656      dense_4[0][0]
__________________________________________________________________________________________________
dense_6 (Dense)                 (None, 512)          262656      dense_5[0][0]
__________________________________________________________________________________________________
z_mean (Dense)                  (None, 5)            2565        dense_6[0][0]
__________________________________________________________________________________________________
z_log_var (Dense)               (None, 5)            2565        dense_6[0][0]
__________________________________________________________________________________________________
z (Lambda)                      (None, 5)            0           z_mean[0][0]
                                                                 z_log_var[0][0]
==================================================================================================
Total params: 8,691,722
Trainable params: 8,691,722
Non-trainable params: 0
__________________________________________________________________________________________________
Model: "decoder"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
z_sampling (InputLayer)      (None, 5)                 0
_________________________________________________________________
dense_7 (Dense)              (None, 512)               3072
_________________________________________________________________
dense_8 (Dense)              (None, 512)               262656
_________________________________________________________________
dense_9 (Dense)              (None, 512)               262656
_________________________________________________________________
dense_10 (Dense)             (None, 512)               262656
_________________________________________________________________
dense_11 (Dense)             (None, 512)               262656
```

```
dense_12 (Dense)             (None, 512)               262656
_________________________________________________________________
dense_13 (Dense)             (None, 14400)             7387200
=================================================================
Total params: 8,703,552
Trainable params: 8,703,552
Non-trainable params: 0
_________________________________________________________________
```

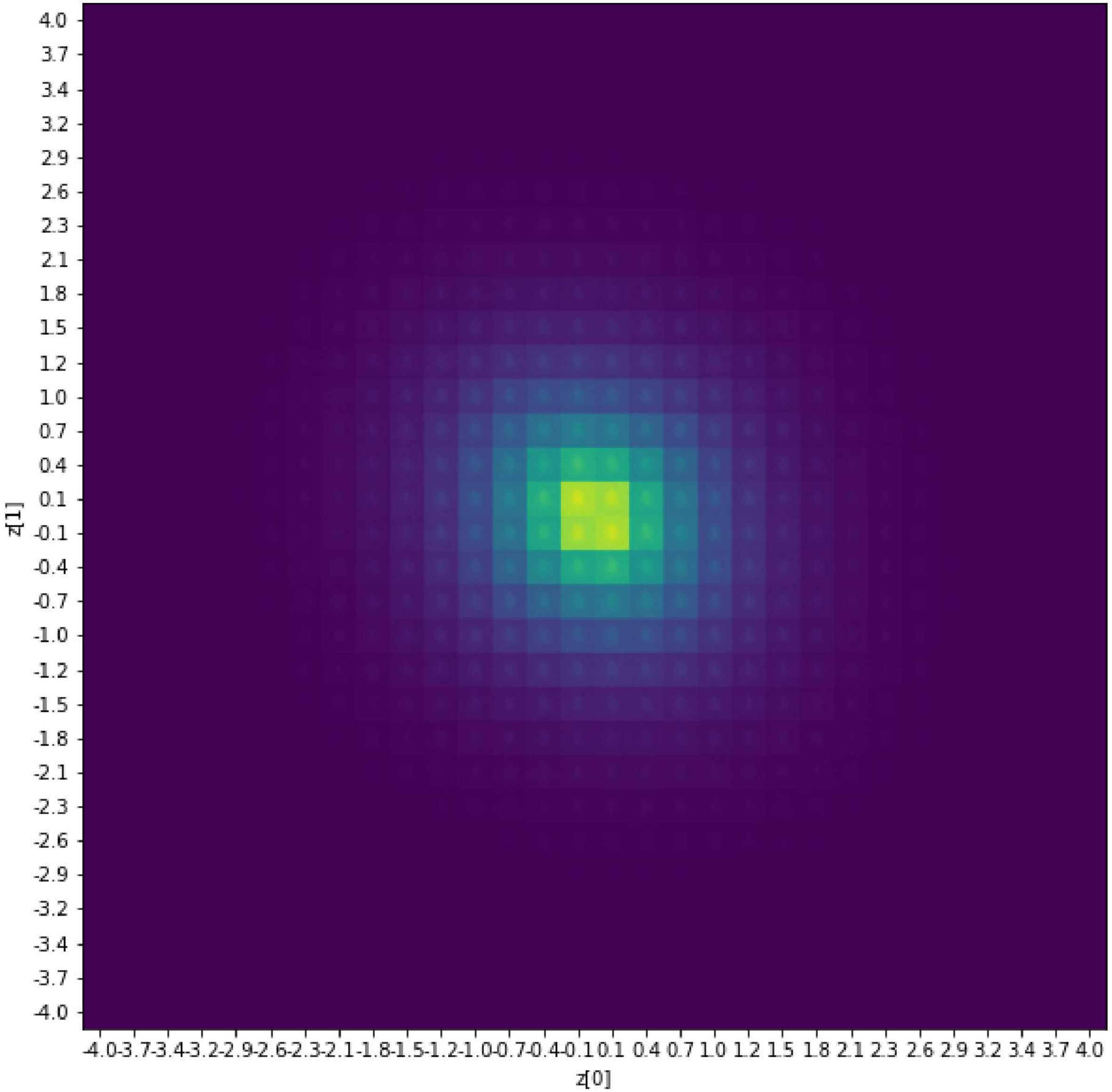
In [7]:

```
adam = Adam(lr=0.00001, beta_1=0.9, beta_2=0.999, amsgrad=False)
vae.compile(optimizer=adam)
vae.summary()
plot_model(vae,
           to_file='vae_mlp.png',
           show_shapes=True)


    # train the autoencoder
vae.fit(x_train,
        epochs=100,
        batch_size=batch_size,
        validation_data=(x_test, None))
vae.save_weights('vae_mlp_mnist.h5')
```

```
/opt/conda/lib/python3.7/site-packages/keras/engine/training_utils.py:819: UserWarning: Output decoder missing from loss dicti
onary. We assume this was done on purpose. The fit and evaluate APIs will not be expecting any data to be passed to decoder.
  'be expecting any data to be passed to {0}.'.format(name))
```

In [8]:

```
n = 30
digit_size = 120
figure = np.zeros((digit_size * n, digit_size * n))
# linearly spaced coordinates corresponding to the 2D plot
# of digit classes in the latent space
grid_x = np.linspace(-4, 4, n)
grid_y = np.linspace(-4, 4, n)[::-1]

for i, yi in enumerate(grid_y):
    for j, xi in enumerate(grid_x):
        z_sample = np.array([[xi, yi,0,0,0]])
        x_decoded = decoder.predict(z_sample)
        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))
start_range = digit_size // 2
end_range = (n - 1) * digit_size + start_range + 1
pixel_range = np.arange(start_range, end_range, digit_size)
sample_range_x = np.round(grid_x, 1)
sample_range_y = np.round(grid_y, 1)
plt.xticks(pixel_range, sample_range_x)
plt.yticks(pixel_range, sample_range_y)
plt.xlabel("z[0]")
plt.ylabel("z[1]")
plt.imshow(figure)
plt.savefig(filename)
plt.show()
```



4.0
3.7
3.4
3.2
2.9
2.6
2.3
2.1
1.8
1.5
1.2
1.0
0.7
0.4
0.1
-0.1
-0.4
-0.7
-1.0
-1.2
-1.5
-1.8
-2.1
-2.3
-2.6
-2.9
-3.2
-3.4
-3.7
-4.0
z[1]
-4.0 -3.7 -3.4 -3.2 -2.9 -2.6 -2.3 -2.1 -1.8 -1.5 -1.2 -1.0 -0.7 -0.4 -0.1 0.1 0.4 0.7 1.0 1.2 1.5 1.8 2.1 2.3 2.6 2.9 3.2 3.4 3.7 4.0
z[0]

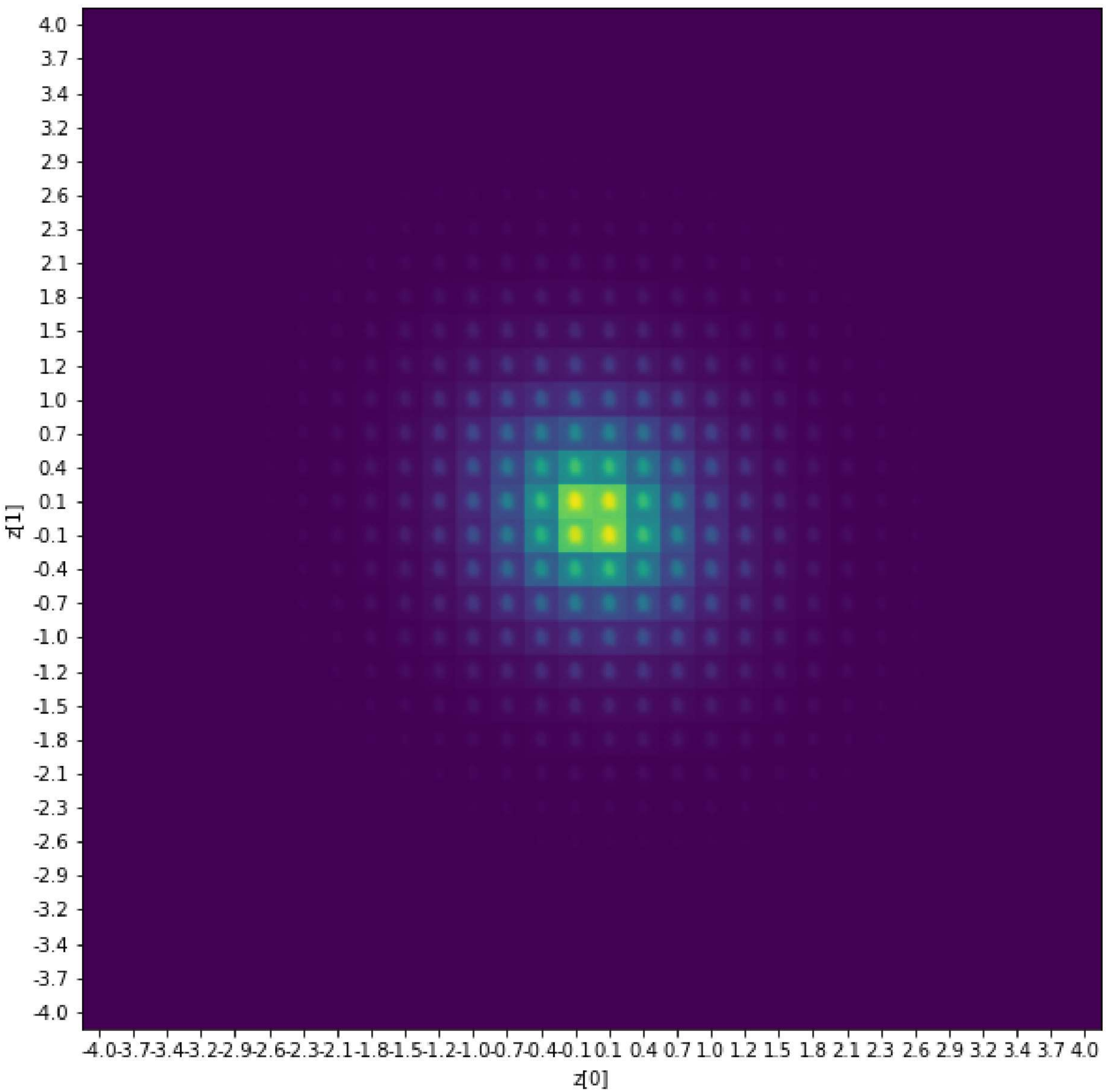
In [9]:

```
adam = Adam(lr=0.000001, beta_1=0.9, beta_2=0.999, amsgrad=False)
vae.compile(optimizer=adam)
vae.summary()
plot_model(vae,
           to_file='vae_mlp.png',
           show_shapes=True)


    # train the autoencoder
vae.fit(x_train,
        epochs=100,
        batch_size=batch_size,
        validation_data=(x_test, None))
vae.save_weights('vae_mlp_mnist.h5')
```

```
/opt/conda/lib/python3.7/site-packages/keras/engine/training_utils.py:819: UserWarning: Output decoder missing from loss dicti
onary. We assume this was done on purpose. The fit and evaluate APIs will not be expecting any data to be passed to decoder.
  'be expecting any data to be passed to {0}.'.format(name))
```

In [10]:

```
n = 30
digit_size = 120
figure = np.zeros((digit_size * n, digit_size * n))
# linearly spaced coordinates corresponding to the 2D plot
# of digit classes in the latent space
grid_x = np.linspace(-4, 4, n)
grid_y = np.linspace(-4, 4, n)[::-1]

for i, yi in enumerate(grid_y):
    for j, xi in enumerate(grid_x):
        z_sample = np.array([[xi, yi,0,0,0]])
        x_decoded = decoder.predict(z_sample)
        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))
start_range = digit_size // 2
end_range = (n - 1) * digit_size + start_range + 1
pixel_range = np.arange(start_range, end_range, digit_size)
sample_range_x = np.round(grid_x, 1)
sample_range_y = np.round(grid_y, 1)
plt.xticks(pixel_range, sample_range_x)
plt.yticks(pixel_range, sample_range_y)
plt.xlabel("z[0]")
plt.ylabel("z[1]")
plt.imshow(figure)
plt.savefig(filename)
plt.show()
```



z[1]
z[0]

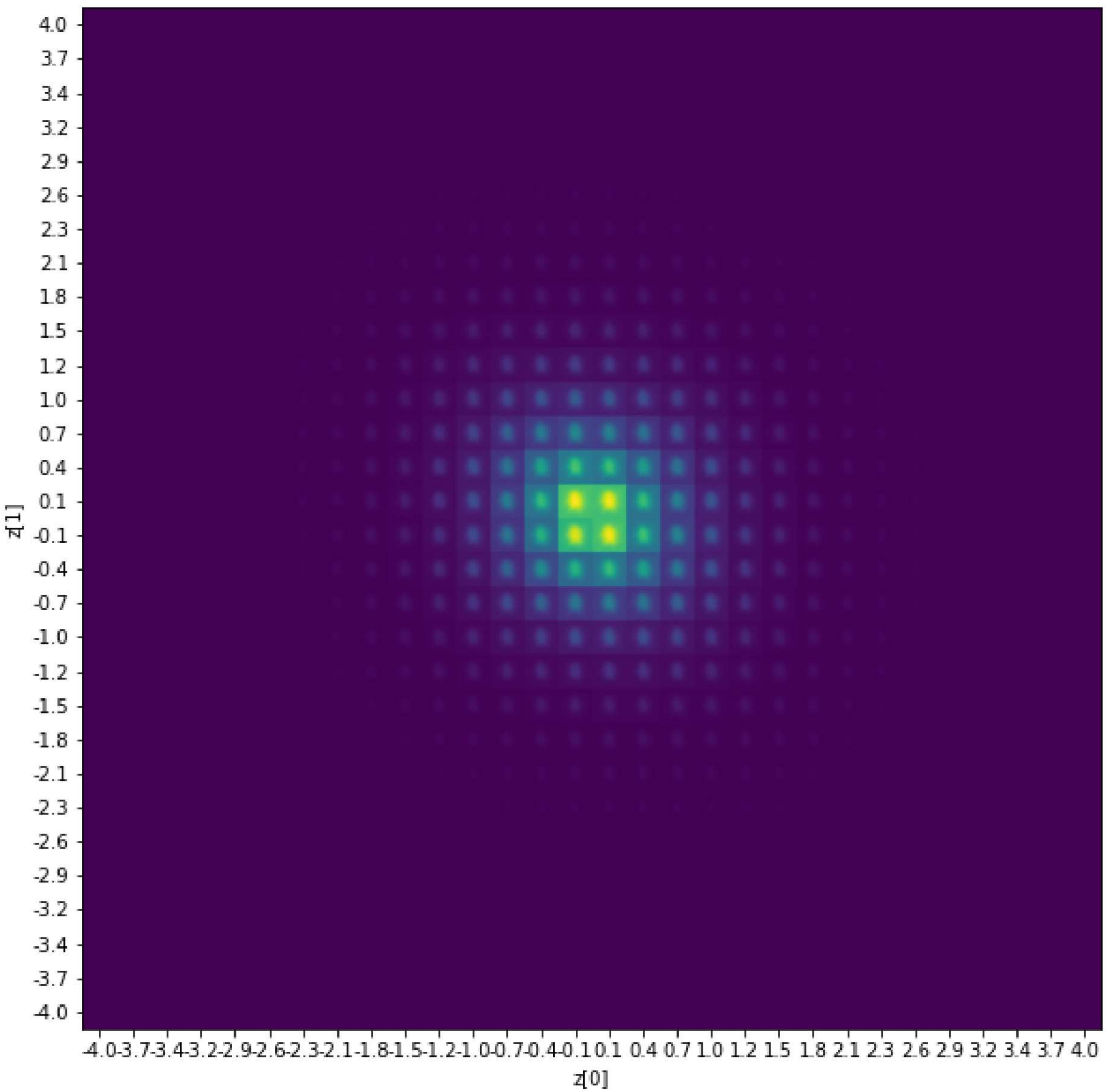
In [11]:

```python
adam = Adam(lr=0.0000001, beta_1=0.9, beta_2=0.999, amsgrad=False)
vae.compile(optimizer=adam)
vae.summary()
plot_model(vae,
           to_file='vae_mlp.png',
           show_shapes=True)


    # train the autoencoder
vae.fit(x_train,
        epochs=1000,
        batch_size=batch_size,
        validation_data=(x_test, None))
vae.save_weights('vae_mlp_mnist.h5')
```

```
/opt/conda/lib/python3.7/site-packages/keras/engine/training_utils.py:819: UserWarning: Output decoder missing from loss dicti
onary. We assume this was done on purpose. The fit and evaluate APIs will not be expecting any data to be passed to decoder.
  'be expecting any data to be passed to {0}.'.format(name))
```

In [12]:

```
n = 30
digit_size = 120
figure = np.zeros((digit_size * n, digit_size * n))
# linearly spaced coordinates corresponding to the 2D plot
# of digit classes in the latent space
grid_x = np.linspace(-4, 4, n)
grid_y = np.linspace(-4, 4, n)[::-1]

for i, yi in enumerate(grid_y):
    for j, xi in enumerate(grid_x):
        z_sample = np.array([[xi, yi,0,0,0]])
        x_decoded = decoder.predict(z_sample)
        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))
start_range = digit_size // 2
end_range = (n - 1) * digit_size + start_range + 1
pixel_range = np.arange(start_range, end_range, digit_size)
sample_range_x = np.round(grid_x, 1)
sample_range_y = np.round(grid_y, 1)
plt.xticks(pixel_range, sample_range_x)
plt.yticks(pixel_range, sample_range_y)
plt.xlabel("z[0]")
plt.ylabel("z[1]")
plt.imshow(figure)
plt.savefig(filename)
plt.show()
```

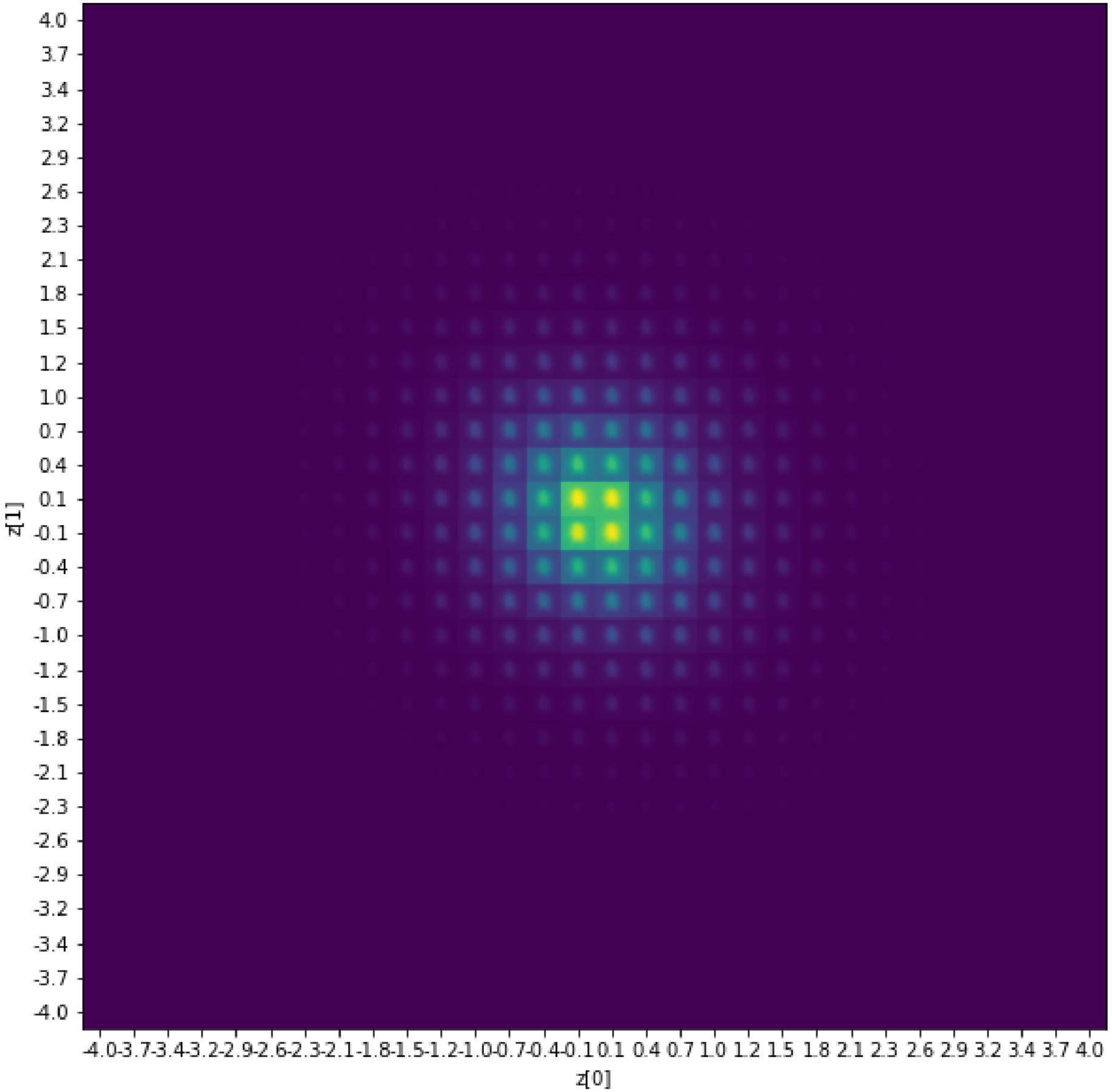


4.0
3.7
3.4
3.2
2.9
2.6
2.3
2.1
1.8
1.5
1.2
1.0
0.7
0.4
0.1
-0.1
-0.4
-0.7
-1.0
-1.2
-1.5
-1.8
-2.1
-2.3
-2.6
-2.9
-3.2
-3.4
-3.7
-4.0
z[1]
-4.0 -3.7 -3.4 -3.2 -2.9 -2.6 -2.3 -2.1 -1.8 -1.5 -1.2 -1.0 -0.7 -0.4 -0.1 0.1 0.4 0.7 1.0 1.2 1.5 1.8 2.1 2.3 2.6 2.9 3.2 3.4 3.7 4.0
z[0]

In [13]:
```
adam = Adam(lr=0.00000000001, beta_1=0.9, beta_2=0.999, amsgrad=False)
vae.compile(optimizer=adam)

vae.fit(x_train,
        epochs=1000,
        batch_size=batch_size,
        validation_data=(x_test, None))
vae.save_weights('vae_mlp_mnist.h5')
```

```
/opt/conda/lib/python3.7/site-packages/keras/engine/training_utils.py:819: UserWarning: Output decoder missing from loss dicti
onary. We assume this was done on purpose. The fit and evaluate APIs will not be expecting any data to be passed to decoder.
  'be expecting any data to be passed to {0}.'.format(name))
```

In [14]:
```
n = 30
digit_size = 120
figure = np.zeros((digit_size * n, digit_size * n))
# linearly spaced coordinates corresponding to the 2D plot
# of digit classes in the latent space
grid_x = np.linspace(-4, 4, n)
grid_y = np.linspace(-4, 4, n)[::-1]

for i, yi in enumerate(grid_y):
    for j, xi in enumerate(grid_x):
        z_sample = np.array([[xi, yi,0,0,0]])
        x_decoded = decoder.predict(z_sample)
        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))
start_range = digit_size // 2
end_range = (n - 1) * digit_size + start_range + 1
pixel_range = np.arange(start_range, end_range, digit_size)
sample_range_x = np.round(grid_x, 1)
sample_range_y = np.round(grid_y, 1)
plt.xticks(pixel_range, sample_range_x)
plt.yticks(pixel_range, sample_range_y)
plt.xlabel("z[0]")
plt.ylabel("z[1]")
plt.imshow(figure)
plt.savefig(filename)
plt.show()
```

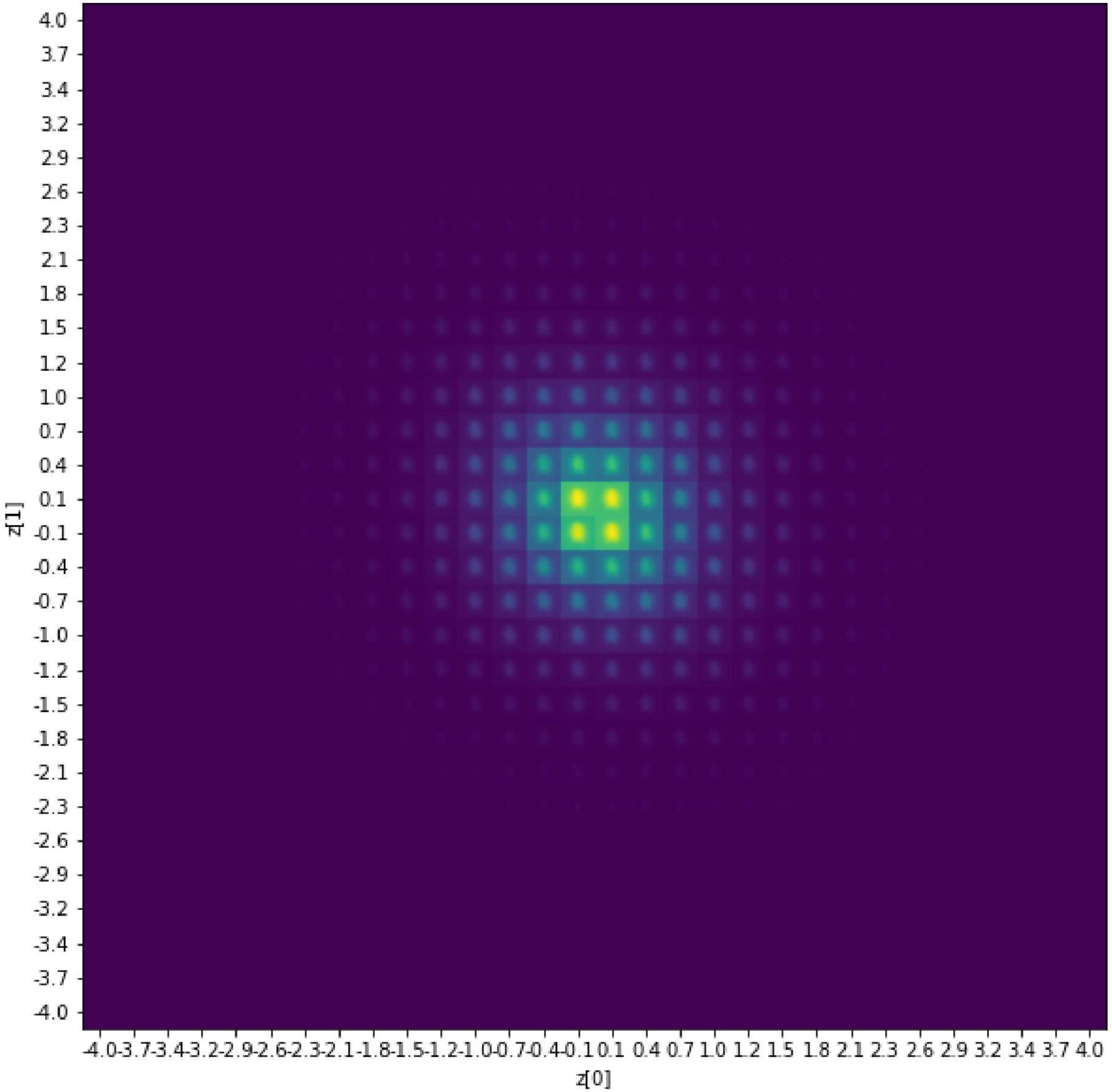


z[1]
4.0
3.7
3.4
3.2
2.9
2.6
2.3
2.1
1.8
1.5
1.2
1.0
0.7
0.4
0.1
-0.1
-0.4
-0.7
-1.0
-1.2
-1.5
-1.8
-2.1
-2.3
-2.6
-2.9
-3.2
-3.4
-3.7
-4.0
-4.0 -3.7 -3.4 -3.2 -2.9 -2.6 -2.3 -2.1 -1.8 -1.5 -1.2 -1.0 -0.7 -0.4 -0.1 0.1 0.4 0.7 1.0 1.2 1.5 1.8 2.1 2.3 2.6 2.9 3.2 3.4 3.7 4.0
z[0]

In [15]:
```python
adam = Adam(lr=1e-30, beta_1=0.9, beta_2=0.999, amsgrad=False)
vae.compile(optimizer=adam)

vae.fit(x_train,
        epochs=5000,
        batch_size=batch_size,
        validation_data=(x_test, None))
```

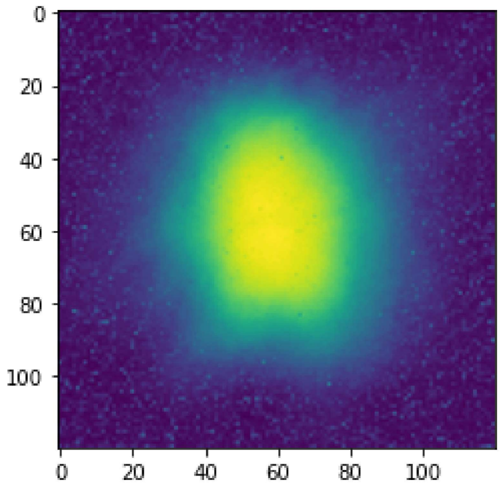
```
Train on 721 samples, validate on 721 samples
Epoch 1/5000
721/721 [==============================] - 1s 1ms/step - loss: 107.6757 - val_loss: 109.7793
Epoch 2/5000
721/721 [==============================] - 0s 216us/step - loss: 112.9569 - val_loss: 109.3140
Epoch 3/5000
721/721 [==============================] - 0s 189us/step - loss: 107.3469 - val_loss: 109.9749
Epoch 4/5000
721/721 [==============================] - 0s 192us/step - loss: 105.9505 - val_loss: 109.1091
Epoch 5/5000
721/721 [==============================] - 0s 190us/step - loss: 108.4177 - val_loss: 109.3794
Epoch 6/5000
721/721 [==============================] - 0s 195us/step - loss: 104.9517 - val_loss: 106.8757
Epoch 7/5000
721/721 [==============================] - 0s 185us/step - loss: 108.6077 - val_loss: 109.6600
Epoch 8/5000
721/721 [==============================] - 0s 183us/step - loss: 111.1306 - val_loss: 107.9051
Epoch 9/5000
721/721 [==============================] - 0s 181us/step - loss: 111.4917 - val_loss: 113.3282
```

In [16]:
```
n = 30
digit_size = 120
figure = np.zeros((digit_size * n, digit_size * n))
# linearly spaced coordinates corresponding to the 2D plot
# of digit classes in the latent space
grid_x = np.linspace(-4, 4, n)
grid_y = np.linspace(-4, 4, n)[::-1]

for i, yi in enumerate(grid_y):
    for j, xi in enumerate(grid_x):
        z_sample = np.array([[xi, yi,0,0,0]])
        x_decoded = decoder.predict(z_sample)
        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))
start_range = digit_size // 2
end_range = (n - 1) * digit_size + start_range + 1
pixel_range = np.arange(start_range, end_range, digit_size)
sample_range_x = np.round(grid_x, 1)
sample_range_y = np.round(grid_y, 1)
plt.xticks(pixel_range, sample_range_x)
plt.yticks(pixel_range, sample_range_y)
plt.xlabel("z[0]")
plt.ylabel("z[1]")
plt.imshow(figure)
plt.savefig(filename)
plt.show()
```


z[1]
z[0]
4.0
3.7
3.4
3.2
2.9
2.6
2.3
2.1
1.8
1.5
1.2
1.0
0.7
0.4
0.1
-0.1
-0.4
-0.7
-1.0
-1.2
-1.5
-1.8
-2.1
-2.3
-2.6
-2.9
-3.2
-3.4
-3.7
-4.0
-4.0 -3.7 -3.4 -3.2 -2.9 -2.6 -2.3 -2.1 -1.8 -1.5 -1.2 -1.0 -0.7 -0.4 -0.1 0.1 0.4 0.7 1.0 1.2 1.5 1.8 2.1 2.3 2.6 2.9 3.2 3.4 3.7 4.0

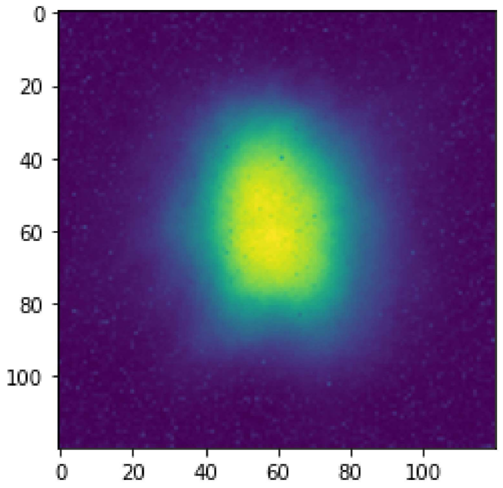
In [17]:
```
plt.imshow(decoder.predict(np.array([[0,0,0,0,0]])).reshape(120,120))
```

Out[17]: <matplotlib.image.AxesImage at 0x7f5d2008e690>



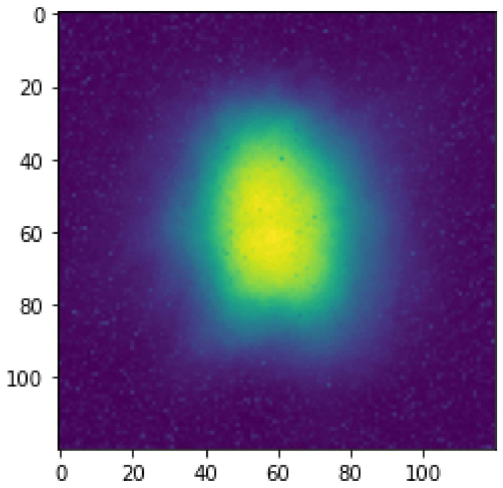
In [18]:
```
plt.imshow(decoder.predict(np.random.rand(5).reshape(1,5)).reshape(120,120))
```

Out[18]: <matplotlib.image.AxesImage at 0x7f5d200d1550>

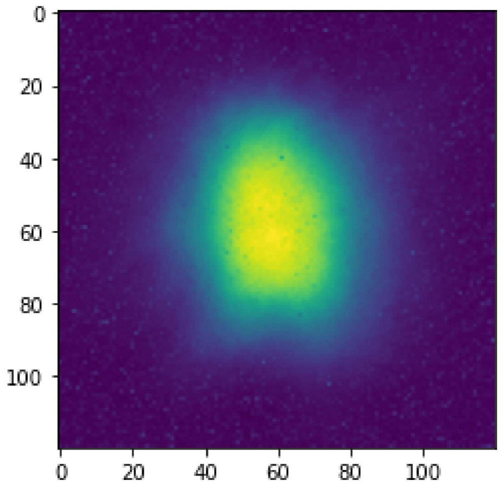


In [19]:
```
plt.imshow(decoder.predict(np.random.rand(5).reshape(1,5)).reshape(120,120))
```

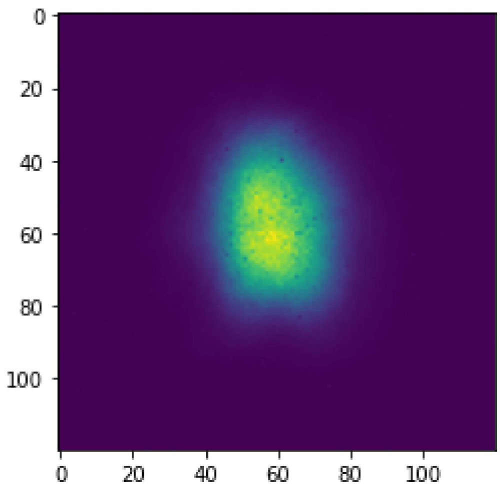
Out[19]: <matplotlib.image.AxesImage at 0x7f5d3e404f50>



In [20]:
```
plt.imshow(decoder.predict(np.random.rand(5).reshape(1,5)).reshape(120,120))
```
Out[20]: <matplotlib.image.AxesImage at 0x7f5d0c0fd7d0>

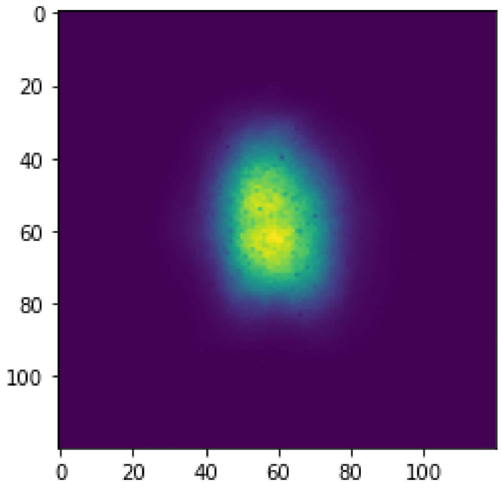


In [21]:
```
plt.imshow(decoder.predict(np.random.rand(5).reshape(1,5)*4).reshape(120,120))
```
Out[21]: <matplotlib.image.AxesImage at 0x7f5d0c073bd0>



In [22]:
```
plt.imshow(decoder.predict(np.random.rand(5).reshape(1,5)*(-4)).reshape(120,120))
```

Out[22]:
```
<matplotlib.image.AxesImage at 0x7f5cb81c7d10>
```



In [23]:
```
codes=[]
for i in range(x_train.shape[0]):
    codes_i=encoder.predict(x_train[i].reshape(1,14400))[0]
    codes.append(codes_i)
```

In [24]:
```
codes=pd.DataFrame(np.array(codes).reshape(-1,5))
```

In [25]:
```
types=manifest.drop('Name',axis=1).idxmax(axis=1)
```

In [26]:
```
types=pd.DataFrame(types)
```

In [27]:
```
result = pd.concat([types.reset_index(drop=True), codes], axis=1)
```

In [28]:
```
result
```

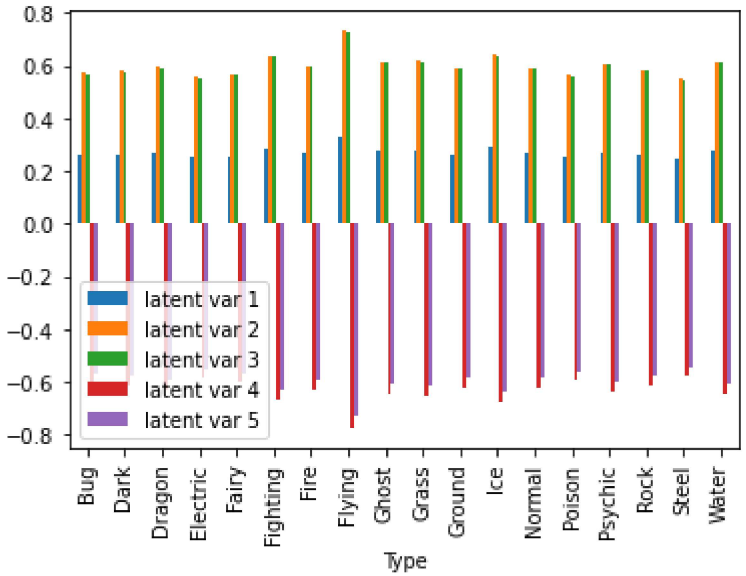
Out[28]:

	0	0	1	2	3	4
0	Rock	0.205003	0.454066	0.453426	-0.478685	-0.450245
1	Electric	0.288344	0.641430	0.638724	-0.676928	-0.636810
2	Ghost	0.318149	0.706633	0.701182	-0.745844	-0.703167
3	Ground	0.306448	0.676655	0.664425	-0.712955	-0.674683
4	Electric	0.245642	0.545277	0.543512	-0.575420	-0.541679
...	...	...	...	...	...	...
716	Fire	0.183147	0.405198	0.403268	-0.426508	-0.402152
717	Bug	0.284819	0.633251	0.630564	-0.668815	-0.629688
718	Fire	0.058533	0.130343	0.114257	-0.135340	-0.135414
719	Water	0.333508	0.740445	0.732675	-0.781020	-0.736775
720	Psychic	0.054068	0.124492	0.108928	-0.128682	-0.131087

721 rows × 6 columns

In [29]:
```
result.columns=['Type','latent var 1','latent var 2', 'latent var 3','latent var 4','latent var 5']
```

In [30]:
```
result.groupby('Type').median().plot(kind='bar')
```
Out[30]: `<matplotlib.axes._subplots.AxesSubplot at 0x7f5cb827e910>`



In [31]:
```
result.groupby('Type').median().plot(kind='box')
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-31-8ad4de5de469> in <module>
----> 1 result.groupby('Type').median().plot(kind='boxplot')

/opt/conda/lib/python3.7/site-packages/pandas/plotting/_core.py in __call__(self, *args, **kwargs)
    778
    779         if kind not in self._all_kinds:
--> 780             raise ValueError(f"{kind} is not a valid plot kind")
    781
    782         # The original data structured can be transformed before passed to the

ValueError: boxplot is not a valid plot kind
```

In [ ]: