Library Imports

In [1]:
```
import numpy as np
import pandas as pd
import cv2
from pascal import PascalVOC
import os
from matplotlib import pyplot as plt
from sklearn.model_selection import train_test_split
%matplotlib inline
import tensorflow as tf
from tensorflow import keras
```

Read the annotation data and build a pandas dataframe with the filenames and filepaths of annotations and images.

In [2]:
```
annotation_filenames = []
annotation_filepaths = []
image_filenames = []
image_filepaths = []

dog_image_folder = "generative-dog-images/all-dogs/all-dogs"
for root, dirs, files in os.walk("generative-dog-images/Annotation/Annotation"):
    if len(dirs) == 0:
        for filename in files:
            annotation_filenames.append(filename)
            annotation_filepaths.append(root.replace("\\","/")+"/"+filename)
            image_filenames.append(filename+".jpg")
            image_filepaths.append(dog_image_folder + "/" + filename+".jpg")

paths_df = pd.DataFrame({
    "annotation_filename": annotation_filenames,
    "annotation_filepath": annotation_filepaths,
    "image_filename": image_filenames,
    "image_filepath": image_filepaths

})
```

In [3]:
```
display(paths_df.head())
```

	annotation_filename	annotation_filepath	image_filename	image_filepath
0	n02085620_10074	generative-dog-images/Annotation/Annotation/n0...	n02085620_10074.jpg	generative-dog-images/all-dogs/all-dogs/n02085...
1	n02085620_10131	generative-dog-images/Annotation/Annotation/n0...	n02085620_10131.jpg	generative-dog-images/all-dogs/all-dogs/n02085...
2	n02085620_10621	generative-dog-images/Annotation/Annotation/n0...	n02085620_10621.jpg	generative-dog-images/all-dogs/all-dogs/n02085...
3	n02085620_1073	generative-dog-images/Annotation/Annotation/n0...	n02085620_1073.jpg	generative-dog-images/all-dogs/all-dogs/n02085...
4	n02085620_10976	generative-dog-images/Annotation/Annotation/n0...	n02085620_10976.jpg	generative-dog-images/all-dogs/all-dogs/n02085...

Helper function to pull the bounding box data from the annotations.

In [4]:
```
def get_bounding_box_from_ann(path):
    ann = PascalVOC.from_xml(path)
    obj = ann.objects[0]
    bb = obj.bndbox
    xmin = bb.xmin
    xmax = bb.xmax
    ymin = bb.ymin
    ymax = bb.ymax
    return (xmin, ymin, xmax, ymax)
```

In [5]:

```
#Add the bounding boxes to the paths dataframe
paths_df["bounding_box"] = paths_df["annotation_filepath"].apply(lambda x: get_bounding_
#Split the bounding box values into separate columns
paths_df[["xmin", "ymin", "xmax", "ymax"]] = pd.DataFrame(paths_df["bounding_box"].tolis
```

In [6]:

```
#There's one annotation that exists for a file that wasn't included in the dataset for s
paths_df.drop(index = 13680, inplace=True) #Drop it
paths_df.reindex(axis = "rows") #Reindex
paths_df.head(5)
```

Out[6]:

	annotation_filename	annotation_filepath	image_filename	image_filepath	bounding_box	xm
0	n02085620_10074	generative-dog-images/Annotation/Annotation/n0...	n02085620_10074.jpg	generative-dog-images/all-dogs/all-dogs/n02085...	(25, 10, 276, 498)	2
1	n02085620_10131	generative-dog-images/Annotation/Annotation/n0...	n02085620_10131.jpg	generative-dog-images/all-dogs/all-dogs/n02085...	(49, 9, 393, 493)	4
2	n02085620_10621	generative-dog-images/Annotation/Annotation/n0...	n02085620_10621.jpg	generative-dog-images/all-dogs/all-dogs/n02085...	(142, 43, 335, 250)	14
3	n02085620_1073	generative-dog-images/Annotation/Annotation/n0...	n02085620_1073.jpg	generative-dog-images/all-dogs/all-dogs/n02085...	(0, 27, 312, 498)	
4	n02085620_10976	generative-dog-images/Annotation/Annotation/n0...	n02085620_10976.jpg	generative-dog-images/all-dogs/all-dogs/n02085...	(90, 104, 242, 452)	9

The images, need to be 64x64 pixel, RGB data.

In [7]:

```
img_shape = (64,64,3)
```

Function to take a row number from a table and pull an image, resize, and output using the OpenCV library.

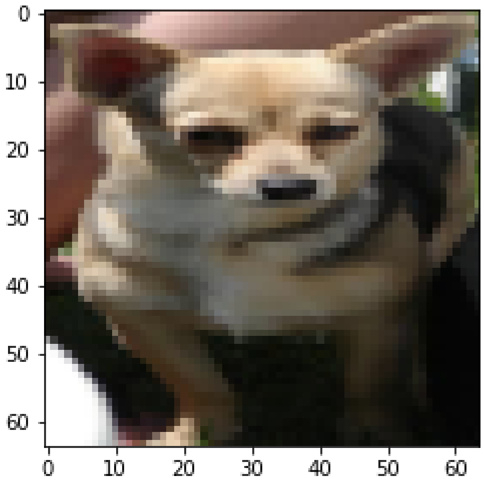
In [8]:

```
def img_from_row(df, rownum, resize_dim=(img_shape[0],img_shape[1]), show=False):
    i = rownum
    img_path = df["image_filepath"].iloc[i]
    img_bb = df["bounding_box"].iloc[i]
    img = cv2.imread(img_path)
    cc_img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    cropped_img = cc_img[img_bb[1]:img_bb[3], img_bb[0]:img_bb[2]]
    resized_img = cv2.resize(cropped_img, resize_dim, interpolation = cv2.INTER_AREA)
    if show:
        plt.imshow(resized_img)
    return resized_img

cropped = img_from_row(paths_df, 0)
plt.imshow(cropped)
```

Out[8]:

```
<matplotlib.image.AxesImage at 0x2a4054cb4f0>
```



Split the paths data into train/test (80/20).

In [9]:

```
train_paths_df, test_paths_df = train_test_split(paths_df, test_size = 0.2, random_state
print(train_paths_df.shape)
print(test_paths_df.shape)
```

```
(16463, 9)
(4116, 9)
```

Create numpy arrays to hold the images..

In [10]:

```
def path_df_to_np_img_array(df, img_shape):
    num_img = df.shape[0]
    arr = np.zeros((num_img, img_shape[0], img_shape[1], img_shape[2]))
    for i in range(num_img):
        img = img_from_row(df, i, resize_dim=(img_shape[0],img_shape[1]), show=False)
        img = img/255.0 #Rescale from 0 to 1.
#          img = 2*(img - 0.5) #Rescale from -1 to 1
        arr[i] = img
    return arr
```

In [11]:

```
train_array = path_df_to_np_img_array(train_paths_df, img_shape)
test_array = path_df_to_np_img_array(test_paths_df, img_shape)
print(train_array.shape)
print(test_array.shape)
```

```
(16463, 64, 64, 3)
(4116, 64, 64, 3)
```

Check/view sample of training images. They'll look distorted because the data is rescaled to include negative numbers that are truncated in the preview.

In [12]:

```
nrows, ncols = 4,4
fig, axs = plt.subplots(nrows, ncols, figsize = (16,16))
for i in range(nrows):
    for j in range(ncols):
        axs[i,j].imshow(train_array[(i*nrows)+j])
```

Check/view sample of testing images.

In [13]:

```python
nrows, ncols = 4,4
fig, axs = plt.subplots(nrows, ncols, figsize = (16,16))
for i in range(nrows):
    for j in range(ncols):
        axs[i,j].imshow(test_array[(i*nrows)+j])
```

I found a very straight-forward example for the basic approach to training a deep convolutional generative adversarial network with TensorFlow and Keras, but it requires extensive modification because it's developed around the Fashion MNIST dataset which is 28x28 pixel single-channel images and we've got 64x64 three-channel images. The initial model (with anticipated poor performance) is closely based on the example to assess the general technique and ensure the output is formatted correctly, but additional, more complex models are tested afterward.

Citation: https://www.geeksforgeeks.org/deep-convolutional-gan-with-keras/

In [14]:

```
batch_size = 1024
def gen_batch(train_array, batch_size = 16, seed_val = 0):
    data = tf.data.Dataset.from_tensor_slices(train_array).shuffle(seed = seed_val, buff
    data = data.batch(batch_size, drop_remainder = True).prefetch(1)
    return data
```

In [15]:

```
num_features = 100

generator1 = keras.models.Sequential([
    keras.layers.Dense(8 * 8 * 128, input_shape =[num_features]),
    keras.layers.Reshape([8, 8, 128]),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2DTranspose(64,(5,5),(4,4),padding="same",activation="relu"),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2DTranspose(img_shape[2],(5,5),(2,2),padding="same",activation="tan
], name = "generator1")
generator1.summary()
```

```
Model: "generator1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 8192)              827392

 reshape (Reshape)           (None, 8, 8, 128)         0

 batch_normalization (BatchN  (None, 8, 8, 128)        512
 ormalization)

 conv2d_transpose (Conv2DTra  (None, 32, 32, 64)       204864
 nspose)

 batch_normalization_1 (Batc  (None, 32, 32, 64)       256
 hNormalization)

 conv2d_transpose_1 (Conv2DT  (None, 64, 64, 3)        4803
 ranspose)

=================================================================
Total params: 1,037,827
Trainable params: 1,037,443
Non-trainable params: 384
_________________________________________________________________
```

In [16]:

```
discriminator1 = keras.models.Sequential([
    keras.layers.Conv2D(64,(5, 5),(2, 2),padding="same",input_shape=img_shape),
    keras.layers.LeakyReLU(0.2),
    keras.layers.Dropout(0.3),
    keras.layers.Conv2D(128,(5, 5),(2, 2),padding ="same"),
    keras.layers.LeakyReLU(0.2),
    keras.layers.Dropout(0.3),
    keras.layers.Flatten(),
    keras.layers.Dense(1, activation ='sigmoid')
], name = "discriminator1")
discriminator1.summary()
```

```
Model: "discriminator1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 32, 32, 64)        4864

 leaky_re_lu (LeakyReLU)     (None, 32, 32, 64)        0

 dropout (Dropout)           (None, 32, 32, 64)        0

 conv2d_1 (Conv2D)           (None, 16, 16, 128)       204928

 leaky_re_lu_1 (LeakyReLU)   (None, 16, 16, 128)       0

 dropout_1 (Dropout)         (None, 16, 16, 128)       0

 flatten (Flatten)           (None, 32768)             0

 dense_1 (Dense)             (None, 1)                 32769
```

```
=================================================================
Total params: 242,561
Trainable params: 242,561
Non-trainable params: 0
_________________________________________________________________
```

In [17]:

```
discriminator1.compile(loss = "binary_crossentropy", optimizer = "adam")
discriminator1.trainable = False
GAN1 = keras.models.Sequential([generator1, discriminator1],name = "GAN1")
GAN1.compile(loss = "binary_crossentropy", optimizer = "adam")
GAN1.summary()
```

```
Model: "GAN1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 generator1 (Sequential)     (None, 64, 64, 3)         1037827   
                                                                 
 discriminator1 (Sequential)  (None, 1)                242561    
                                                                 
=================================================================
Total params: 1,280,388
Trainable params: 1,037,443
Non-trainable params: 242,945
_________________________________________________________________
```

In [18]:

```
def save_epoch_images(model, epoch, test_input, output_path = ""):
    predictions = model(test_input, training = False)
    nrows, ncols = 4,4
    fig, axs = plt.subplots(nrows, ncols, figsize = (16,16))
    adj = (255.0/2.0)
    for i in range(nrows):
        for j in range(ncols):
            axs[i,j].imshow(((predictions[(i*nrows)+j].numpy()*adj)+adj).astype(int))

    plt.savefig(output_path + "/epoch_{:04d}.png".format(epoch))
    plt.close('all')
```

In [19]:

```
seed = tf.random.normal(shape = [batch_size, num_features])

def train_DCGAN(GAN, data, batch_size, num_features, epochs = 10, output_path = ""):
    generator, discriminator = GAN.layers
    for epoch in range(epochs):
        print(f"Epoch: {epoch}/{epochs}")
        for data_batch in data:
            data_batch = tf.cast(data_batch, tf.float32)
            noise = tf.random.normal(shape = [batch_size, num_features])
            generated_images = generator(noise)

            synthetic_real = tf.concat([generated_images, data_batch], axis = 0)
            lbls1 = tf.constant([[0.0]]*batch_size + [[1.0]]*batch_size)
            discriminator.trainable = True
            discriminator.train_on_batch(synthetic_real, lbls1)

            noise = tf.random.normal(shape = [batch_size, num_features])
            lbls2 = tf.constant([[1.0]]*batch_size)
            discriminator.trainable = False
            GAN.train_on_batch(noise, lbls2)

        save_epoch_images(model = generator, epoch = epoch, test_input = seed, output_pa
```

In [20]:

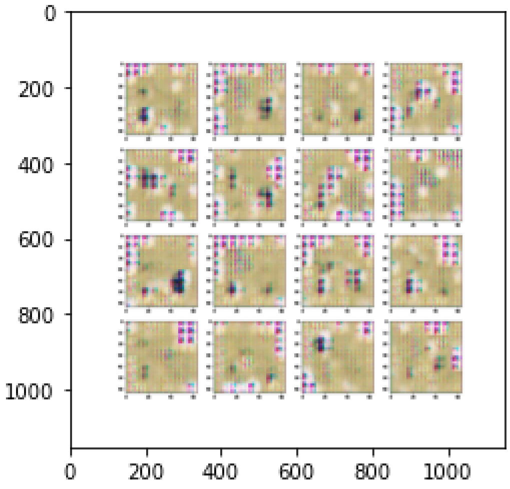
```
dataset = gen_batch(train_array, batch_size = batch_size, seed_val = 0)
train_DCGAN(GAN1, dataset, batch_size, num_features, epochs = 5, output_path = "model01"
```

```
Epoch: 0/5
```

```
Epoch: 1/5
Epoch: 2/5
Epoch: 3/5
Epoch: 4/5
```

In [21]:

```
GAN1_epoch_image = cv2.imread("model01/epoch_0004.png")
GAN1_epoch_image = cv2.cvtColor(GAN1_epoch_image, cv2.COLOR_BGR2RGB)
plt.imshow(GAN1_epoch_image)
plt.show()
```



In [22]:

```
num_features = 100

generator2 = keras.models.Sequential([
    keras.layers.Dense(4 * 4 * 128, input_shape =[num_features]),
    keras.layers.Reshape([4, 4, 128]),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2DTranspose(64,(5,5),(2,2),padding="same",activation="relu"),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2DTranspose(64,(5,5),(2,2),padding="same",activation="relu"),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2DTranspose(64,(5,5),(2,2),padding="same",activation="relu"),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2DTranspose(img_shape[2],(5,5),(2,2),padding="same",activation="tan
], name = "generator2")
generator2.summary()
```

```
Model: "generator2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_2 (Dense)             (None, 2048)              206848

 reshape_1 (Reshape)         (None, 4, 4, 128)         0

 batch_normalization_2 (Batc  (None, 4, 4, 128)        512
 hNormalization)

 conv2d_transpose_2 (Conv2DT  (None, 8, 8, 64)         204864
 ranspose)

 batch_normalization_3 (Batc  (None, 8, 8, 64)         256
 hNormalization)

 conv2d_transpose_3 (Conv2DT  (None, 16, 16, 64)       102464
 ranspose)

 batch_normalization_4 (Batc  (None, 16, 16, 64)       256
 hNormalization)
 conv2d_transpose_4 (Conv2DT  (None, 32, 32, 64)       102464
 ranspose)

 batch_normalization_5 (Batc  (None, 32, 32, 64)       256
 hNormalization)

 conv2d_transpose_5 (Conv2DT  (None, 64, 64, 3)        4803
 ranspose)

=================================================================
Total params: 622,723
Trainable params: 622,083
Non-trainable params: 640
_________________________________________________________________
```

In [23]:

```
discriminator2 = keras.models.Sequential([
    keras.layers.Conv2D(64,(5, 5),(2, 2),padding="same",input_shape=img_shape),
    keras.layers.LeakyReLU(0.2),
    keras.layers.Dropout(0.3),
    keras.layers.Conv2D(128,(5, 5),(2, 2),padding ="same"),
    keras.layers.LeakyReLU(0.2),
    keras.layers.Dropout(0.3),
    keras.layers.Conv2D(256,(5, 5),(2, 2),padding ="same"),
    keras.layers.LeakyReLU(0.2),
    keras.layers.Dropout(0.3),
    keras.layers.Flatten(),
    keras.layers.Dense(1, activation ='sigmoid')
], name = "discriminator2")
discriminator2.summary()
```

```
Model: "discriminator2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_2 (Conv2D)           (None, 32, 32, 64)        4864

 leaky_re_lu_2 (LeakyReLU)   (None, 32, 32, 64)        0

 dropout_2 (Dropout)         (None, 32, 32, 64)        0

 conv2d_3 (Conv2D)           (None, 16, 16, 128)       204928

 leaky_re_lu_3 (LeakyReLU)   (None, 16, 16, 128)       0

 dropout_3 (Dropout)         (None, 16, 16, 128)       0

 conv2d_4 (Conv2D)           (None, 8, 8, 256)         819456

 leaky_re_lu_4 (LeakyReLU)   (None, 8, 8, 256)         0

 dropout_4 (Dropout)         (None, 8, 8, 256)         0

 flatten_1 (Flatten)         (None, 16384)             0

 dense_3 (Dense)             (None, 1)                 16385

=================================================================
Total params: 1,045,633
Trainable params: 1,045,633
Non-trainable params: 0
_________________________________________________________________
```

In [24]:

```
discriminator2.compile(loss = "binary_crossentropy", optimizer = "adam")
discriminator2.trainable = False
GAN2 = keras.models.Sequential([generator2, discriminator2], name = "GAN2")
```

```
GAN2.compile(loss = "binary_crossentropy", optimizer = "adam")
GAN2.summary()
```

```
Model: "GAN2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 generator2 (Sequential)     (None, 64, 64, 3)         622723    
                                                                 
 discriminator2 (Sequential)  (None, 1)                1045633   
                                                                 
=================================================================
Total params: 1,668,356
Trainable params: 622,083
Non-trainable params: 1,046,273
_________________________________________________________________
```

In [25]:

```
dataset = gen_batch(train_array, batch_size = batch_size, seed_val = 0)
train_DCGAN(GAN2, dataset, batch_size, num_features, epochs = 250, output_path = "model0
```

```
Epoch: 0/250
Epoch: 1/250
Epoch: 2/250
Epoch: 3/250
Epoch: 4/250
Epoch: 5/250
Epoch: 6/250
Epoch: 7/250
Epoch: 8/250
Epoch: 9/250
Epoch: 10/250
Epoch: 11/250
Epoch: 12/250
Epoch: 13/250
Epoch: 14/250
Epoch: 15/250
Epoch: 16/250
Epoch: 17/250
Epoch: 18/250
Epoch: 19/250
Epoch: 20/250
Epoch: 21/250
Epoch: 22/250
Epoch: 23/250
Epoch: 24/250
Epoch: 25/250
Epoch: 26/250
Epoch: 27/250
Epoch: 28/250
Epoch: 29/250
Epoch: 30/250
Epoch: 31/250
Epoch: 32/250
Epoch: 33/250
Epoch: 34/250
Epoch: 35/250
Epoch: 36/250
Epoch: 37/250
Epoch: 38/250
Epoch: 39/250
Epoch: 40/250
Epoch: 41/250
Epoch: 42/250
Epoch: 43/250
Epoch: 44/250
Epoch: 45/250
```

```
Epoch: 46/250
Epoch: 47/250
Epoch: 48/250
Epoch: 49/250
Epoch: 50/250
Epoch: 51/250
Epoch: 52/250
Epoch: 53/250
Epoch: 54/250
Epoch: 55/250
Epoch: 56/250
Epoch: 57/250
Epoch: 58/250
Epoch: 59/250
Epoch: 60/250
Epoch: 61/250
Epoch: 62/250
Epoch: 63/250
Epoch: 64/250
Epoch: 65/250
Epoch: 66/250
Epoch: 67/250
Epoch: 68/250
Epoch: 69/250
Epoch: 70/250
Epoch: 71/250
Epoch: 72/250
Epoch: 73/250
Epoch: 74/250
Epoch: 75/250
Epoch: 76/250
Epoch: 77/250
Epoch: 78/250
Epoch: 79/250
Epoch: 80/250
Epoch: 81/250
Epoch: 82/250
Epoch: 83/250
Epoch: 84/250
Epoch: 85/250
Epoch: 86/250
Epoch: 87/250
Epoch: 88/250
Epoch: 89/250
Epoch: 90/250
Epoch: 91/250
Epoch: 92/250
Epoch: 93/250
Epoch: 94/250
Epoch: 95/250
Epoch: 96/250
Epoch: 97/250
Epoch: 98/250
Epoch: 99/250
Epoch: 100/250
Epoch: 101/250
Epoch: 102/250
Epoch: 103/250
Epoch: 104/250
Epoch: 105/250
Epoch: 106/250
Epoch: 107/250
Epoch: 108/250
Epoch: 109/250
Epoch: 110/250
Epoch: 111/250
```

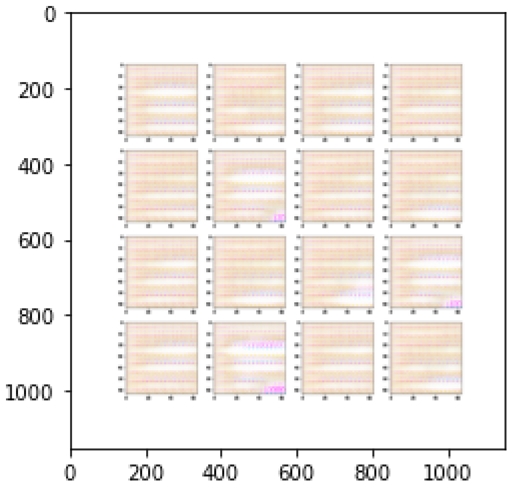
```
Epoch: 112/250
Epoch: 113/250
Epoch: 114/250
Epoch: 115/250
Epoch: 116/250
Epoch: 117/250
Epoch: 118/250
Epoch: 119/250
Epoch: 120/250
Epoch: 121/250
Epoch: 122/250
Epoch: 123/250
Epoch: 124/250
Epoch: 125/250
Epoch: 126/250
Epoch: 127/250
Epoch: 128/250
Epoch: 129/250
Epoch: 130/250
Epoch: 131/250
Epoch: 132/250
Epoch: 133/250
Epoch: 134/250
Epoch: 135/250
Epoch: 136/250
Epoch: 137/250
Epoch: 138/250
Epoch: 139/250
Epoch: 140/250
Epoch: 141/250
Epoch: 142/250
Epoch: 143/250
Epoch: 144/250
Epoch: 145/250
Epoch: 146/250
Epoch: 147/250
Epoch: 148/250
Epoch: 149/250
Epoch: 150/250
Epoch: 151/250
Epoch: 152/250
Epoch: 153/250
Epoch: 154/250
Epoch: 155/250
Epoch: 156/250
Epoch: 157/250
Epoch: 158/250
Epoch: 159/250
Epoch: 160/250
Epoch: 161/250
Epoch: 162/250
Epoch: 163/250
Epoch: 164/250
Epoch: 165/250
Epoch: 166/250
Epoch: 167/250
Epoch: 168/250
Epoch: 169/250
Epoch: 170/250
Epoch: 171/250
Epoch: 172/250
Epoch: 173/250
Epoch: 174/250
Epoch: 175/250
Epoch: 176/250
Epoch: 177/250
```

```
Epoch: 178/250
Epoch: 179/250
Epoch: 180/250
Epoch: 181/250
Epoch: 182/250
Epoch: 183/250
Epoch: 184/250
Epoch: 185/250
Epoch: 186/250
Epoch: 187/250
Epoch: 188/250
Epoch: 189/250
Epoch: 190/250
Epoch: 191/250
Epoch: 192/250
Epoch: 193/250
Epoch: 194/250
Epoch: 195/250
Epoch: 196/250
Epoch: 197/250
Epoch: 198/250
Epoch: 199/250
Epoch: 200/250
Epoch: 201/250
Epoch: 202/250
Epoch: 203/250
Epoch: 204/250
Epoch: 205/250
Epoch: 206/250
Epoch: 207/250
Epoch: 208/250
Epoch: 209/250
Epoch: 210/250
Epoch: 211/250
Epoch: 212/250
Epoch: 213/250
Epoch: 214/250
Epoch: 215/250
Epoch: 216/250
Epoch: 217/250
Epoch: 218/250
Epoch: 219/250
Epoch: 220/250
Epoch: 221/250
Epoch: 222/250
Epoch: 223/250
Epoch: 224/250
Epoch: 225/250
Epoch: 226/250
Epoch: 227/250
Epoch: 228/250
Epoch: 229/250
Epoch: 230/250
Epoch: 231/250
Epoch: 232/250
Epoch: 233/250
Epoch: 234/250
Epoch: 235/250
Epoch: 236/250
Epoch: 237/250
Epoch: 238/250
Epoch: 239/250
Epoch: 240/250
Epoch: 241/250
Epoch: 242/250
Epoch: 243/250
```

```
Epoch: 244/250
Epoch: 245/250
Epoch: 246/250
Epoch: 247/250
Epoch: 248/250
Epoch: 249/250
```

In [26]:

```
GAN2_epoch_image = cv2.imread("model02/epoch_0249.png")
GAN2_epoch_image = cv2.cvtColor(GAN2_epoch_image, cv2.COLOR_BGR2RGB)
plt.imshow(GAN2_epoch_image)
plt.show()
```



Well, that was disappointing. Clearly, I'm doing something wrong. Unfortunately, I've spent over 20 hours on this because the training time is so long, so I made some blobs that may or may not get a few points towards being a dog if the Kaggle leaderboard wasn't already closed. I'll debug this in the future (mostly because I want a cat picture generator) but that's going to have to wait until I have more time.