GAN On Senior images(selfie dataset)

```
import tensorflow as tf
from tensorflow.keras.layers import Input, Reshape, Dropout, Dense
from tensorflow.keras.layers import Flatten, BatchNormalization
from tensorflow.keras.layers import Activation, ZeroPadding2D
from tensorflow.keras.layers import LeakyReLU
from tensorflow.keras.layers import UpSampling2D, Conv2D
from tensorflow.keras.models import Sequential, Model, load_model
from tensorflow.keras.optimizers import Adam
import numpy as np
from PIL import Image
from tqdm import tqdm
import os
import time
import matplotlib.pyplot as plt
```

```
# Generation resolution - Must be square and Training data is also scaled to this.


GENERATE_RES = 3 # Generation resolution factor (1=32, 2=64, 3=96, 4=128, etc.)
GENERATE_SQUARE = 32 * GENERATE_RES # rows/cols into square format
IMAGE_CHANNELS = 3

# Preview image
PREVIEW_ROWS = 1
PREVIEW_COLS = 1
PREVIEW_MARGIN = 1
SEED_SIZE = 100

# Configuration
DATA_PATH = '/content/drive/MyDrive/Seniors'
EPOCHS = 12000
```

```
BATCH_SIZE = 32
BUFFER_SIZE = 60000

print(f"Will generate {GENERATE_SQUARE}px square images.")
```

```
Will generate 96px square images.
```

```
# For loading images folder
training_binary_path = os.path.join(DATA_PATH,f'training_data_{GENERATE_SQUARE}_{GENERATE_SQUARE}.npy')

print(f"Looking for file: {training_binary_path}")

if not os.path.isfile(training_binary_path):
  start = time.time()
  print("Loading training images...")

  training_data = []
  faces_path = os.path.join(DATA_PATH)
  for filename in tqdm(os.listdir(faces_path)):
      path = os.path.join(faces_path,filename)
      image = Image.open(path).resize((GENERATE_SQUARE,
            GENERATE_SQUARE),Image.ANTIALIAS)
      training_data.append(np.asarray(image))
  training_data = np.reshape(training_data,(-1,GENERATE_SQUARE,
            GENERATE_SQUARE,IMAGE_CHANNELS))
  training_data = training_data.astype(np.float32)
  training_data = training_data / 127.5 - 1.


  print("Saving training image binary...")
  np.save(training_binary_path,training_data)
  elapsed = time.time()-start
  print (f'Image preprocess time: {hms_string(elapsed)}')
else:
  print("Loading previous training pickle...")
  training_data = np.load(training_binary_path)
```

```
Looking for file: /content/drive/MyDrive/Seniors/training_data_96_96.npy
Loading previous training pickle...
```

```
# To calculate training time
def hms_string(sec_elapsed):
    h = int(sec_elapsed / (60 * 60))
    m = int((sec_elapsed % (60 * 60)) / 60)
    s = sec_elapsed % 60
    return "{}:{:>02}:{:>05.2f}".format(h, m, s)
```

```
# Batch and shuffle the data
train_dataset = tf.data.Dataset.from_tensor_slices(training_data) \
    .shuffle(BUFFER_SIZE).batch(BATCH_SIZE)
```

```
def build_generator(seed_size, channels):
    model = Sequential()

    model.add(Dense(4*4*256,activation="relu",input_dim=seed_size))
    model.add(Reshape((4,4,256)))

    model.add(UpSampling2D())
    model.add(Conv2D(256,kernel_size=3,padding="same"))
    model.add(BatchNormalization(momentum=0.8))
    model.add(Activation("relu"))

    model.add(UpSampling2D())
    model.add(Conv2D(256,kernel_size=3,padding="same"))
    model.add(BatchNormalization(momentum=0.8))
    model.add(Activation("relu"))

    # Output resolution, additional upsampling
    model.add(UpSampling2D())
    model.add(Conv2D(128,kernel_size=3,padding="same"))
    model.add(BatchNormalization(momentum=0.8))
    model.add(Activation("relu"))

    if GENERATE_RES>1:
      model.add(UpSampling2D(size=(GENERATE_RES,GENERATE_RES)))
```

```
        model.add(Conv2D(128,kernel_size=3,padding="same"))
        model.add(BatchNormalization(momentum=0.8))
        model.add(Activation("relu"))

    # Final CNN layer
    model.add(Conv2D(channels,kernel_size=3,padding="same"))
    model.add(Activation("tanh"))

    return model


def build_discriminator(image_shape):
    model = Sequential()

    model.add(Conv2D(32, kernel_size=3, strides=2, input_shape=image_shape,
                     padding="same"))
    model.add(LeakyReLU(alpha=0.2))

    model.add(Dropout(0.25))
    model.add(Conv2D(64, kernel_size=3, strides=2, padding="same"))
    model.add(ZeroPadding2D(padding=((0,1),(0,1))))
    model.add(BatchNormalization(momentum=0.8))
    model.add(LeakyReLU(alpha=0.2))

    model.add(Dropout(0.25))
    model.add(Conv2D(128, kernel_size=3, strides=2, padding="same"))
    model.add(BatchNormalization(momentum=0.8))
    model.add(LeakyReLU(alpha=0.2))

    model.add(Dropout(0.25))
    model.add(Conv2D(256, kernel_size=3, strides=1, padding="same"))
    model.add(BatchNormalization(momentum=0.8))
    model.add(LeakyReLU(alpha=0.2))

    model.add(Dropout(0.25))
    model.add(Conv2D(512, kernel_size=3, strides=1, padding="same"))
    model.add(BatchNormalization(momentum=0.8))
    model.add(LeakyReLU(alpha=0.2))
```

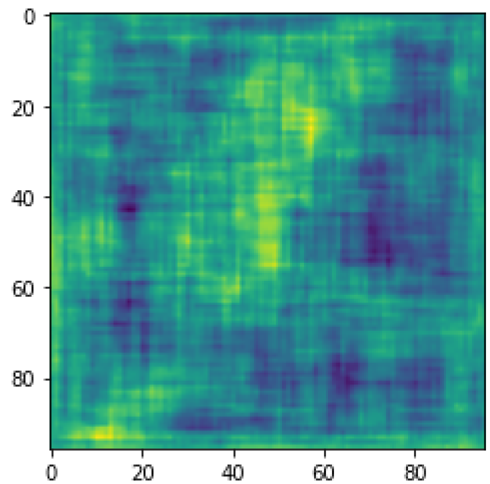
```
    model.add(Dropout(0.25))
    model.add(Flatten())
    model.add(Dense(1, activation='sigmoid'))

    return model
```

```
def save_images(cnt,noise):
  image_array = np.full((
      PREVIEW_MARGIN + (PREVIEW_ROWS * (GENERATE_SQUARE+PREVIEW_MARGIN)),
      PREVIEW_MARGIN + (PREVIEW_COLS * (GENERATE_SQUARE+PREVIEW_MARGIN)), 3),
      255, dtype=np.uint8)

  generated_images = generator.predict(noise)

  generated_images = 0.5 * generated_images + 0.5

  image_count = 0
  for row in range(PREVIEW_ROWS):
      for col in range(PREVIEW_COLS):
        r = row * (GENERATE_SQUARE+16) + PREVIEW_MARGIN
        c = col * (GENERATE_SQUARE+16) + PREVIEW_MARGIN
        image_array[r:r+GENERATE_SQUARE,c:c+GENERATE_SQUARE] \
            = generated_images[image_count] * 255
        image_count += 1


  output_path = os.path.join(DATA_PATH,'output1')
  if not os.path.exists(output_path):
    os.makedirs(output_path)

  filename = os.path.join(output_path,f"train-{cnt}.png")
  im = Image.fromarray(image_array)
  im.save(filename)
```

```
# Sample noise generated by generator
generator = build_generator(SEED_SIZE, IMAGE_CHANNELS)

noise = tf.random.normal([1, SEED_SIZE])
```

```
noise = tf.random.normal([1, SEED_SIZE])
generated_image = generator(noise, training=False)

plt.imshow(generated_image[0, :, :, 0])
```

```
<matplotlib.image.AxesImage at 0x7f0c8be54d68>
```




```
image_shape = (GENERATE_SQUARE,GENERATE_SQUARE,IMAGE_CHANNELS)

discriminator = build_discriminator(image_shape)
decision = discriminator(generated_image)
print (decision)
```

```
tf.Tensor([[0.5002568]], shape=(1, 1), dtype=float32)
```

```
# This method returns a helper function to compute cross entropy loss
cross_entropy = tf.keras.losses.BinaryCrossentropy()

def discriminator_loss(real_output, fake_output):
    real_loss = cross_entropy(tf.ones_like(real_output), real_output)
    fake_loss = cross_entropy(tf.zeros_like(fake_output), fake_output)
    total_loss = real_loss + fake_loss
    return total_loss

def generator_loss(fake_output):
```

```
    return cross_entropy(tf.ones_like(fake_output), fake_output)
```

```
generator_optimizer = tf.keras.optimizers.Adam(1.5e-4,0.5)
discriminator_optimizer = tf.keras.optimizers.Adam(1.5e-4,0.5)
```

```
# This annotation causes the function to be "compiled".
@tf.function
def train_step(images):
  seed = tf.random.normal([BATCH_SIZE, SEED_SIZE])

  with tf.GradientTape() as gen_tape, tf.GradientTape() as disc_tape:
    generated_images = generator(seed, training=True)
    real_output = discriminator(images, training=True)
    fake_output = discriminator(generated_images, training=True)

    gen_loss = generator_loss(fake_output)
    disc_loss = discriminator_loss(real_output, fake_output)


    gradients_of_generator = gen_tape.gradient(\
        gen_loss, generator.trainable_variables)
    gradients_of_discriminator = disc_tape.gradient(\
        disc_loss, discriminator.trainable_variables)

    generator_optimizer.apply_gradients(zip(
        gradients_of_generator, generator.trainable_variables))
    discriminator_optimizer.apply_gradients(zip(
        gradients_of_discriminator,
        discriminator.trainable_variables))
  return gen_loss,disc_loss
```

```
def train(dataset, epochs):
  fixed_seed = np.random.normal(0, 1, (PREVIEW_ROWS * PREVIEW_COLS,
                                       SEED_SIZE))
  start = time.time()
```

```
  for epoch in range(epochs):
    epoch_start = time.time()

    gen_loss_list = []
    disc_loss_list = []

    for image_batch in dataset:
      t = train_step(image_batch)
      gen_loss_list.append(t[0])
      disc_loss_list.append(t[1])

    g_loss = sum(gen_loss_list) / len(gen_loss_list)
    d_loss = sum(disc_loss_list) / len(disc_loss_list)

    epoch_elapsed = time.time()-epoch_start
    print (f'Epoch {epoch+1}, gen loss={g_loss},disc loss={d_loss},'\
           ' {hms_string(epoch_elapsed)}')
    save_images(epoch,fixed_seed)

  elapsed = time.time()-start
  print (f'Training time: {hms_string(elapsed)}')
```

```
train(train_dataset, 15000)
```

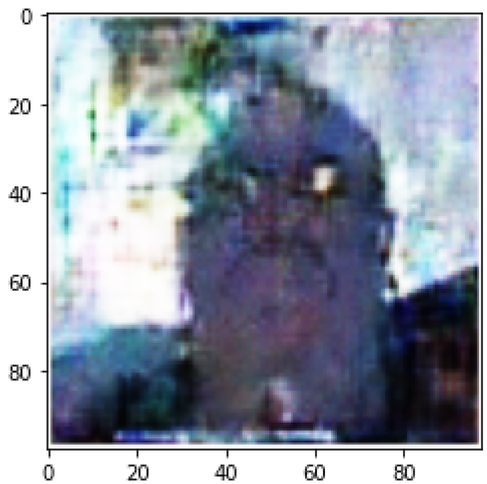
```
Epoch 14943, gen loss=14.309613227844238,disc loss=0.019359730184078217, {hms_string(epoch_elapsed)}
Epoch 14944, gen loss=13.879215240478516,disc loss=0.0007962162490002811, {hms_string(epoch_elapsed)}
Epoch 14945, gen loss=12.701329231262207,disc loss=0.00551072042435407б, {hms_string(epoch_elapsed)}
Epoch 14946, gen loss=10.617058753967285,disc loss=0.10150858014822006, {hms_string(epoch_elapsed)}
Epoch 14947, gen loss=21.599241256713867,disc loss=9.967257028620224e-06, {hms_string(epoch_elapsed)}
Epoch 14948, gen loss=23.245153427124023,disc loss=5.508782123797573e-05, {hms_string(epoch_elapsed)}
Epoch 14949, gen loss=19.684791564941406,disc loss=0.010981845669448376, {hms_string(epoch_elapsed)}
Epoch 14950, gen loss=19.638031005859375,disc loss=8.367277041543275e-05, {hms_string(epoch_elapsed)}
Epoch 14951, gen loss=16.569631576538086,disc loss=0.0020508584566414356, {hms_string(epoch_elapsed)}
Epoch 14952, gen loss=15.691271781921387,disc loss=0.03408218920230865, {hms_string(epoch_elapsed)}
Epoch 14953, gen loss=22.305774688720703,disc loss=0.018302669748663902, {hms_string(epoch_elapsed)}
Epoch 14954, gen loss=22.30660057067871,disc loss=0.14357516169548035, {hms_string(epoch_elapsed)}
Epoch 14955, gen loss=19.346786499023438,disc loss=0.36343511939048767, {hms_string(epoch_elapsed)}
Epoch 14956, gen loss=34.18204116821289,disc loss=0.31258824467658997, {hms_string(epoch_elapsed)}
Epoch 14957, gen loss=25.69986343383789,disc loss=0.268161416053772, {hms_string(epoch_elapsed)}
Epoch 14958, gen loss=18.289628982543945,disc loss=4.137934956816025e-05, {hms_string(epoch_elapsed)}
```

```
Epoch 14959, gen loss=14.50621223449707,disc loss=0.021638156846165657, {hms_string(epoch_elapsed)}
Epoch 14960, gen loss=15.095823287963867,disc loss=0.0018165496876463294, {hms_string(epoch_elapsed)}
Epoch 14961, gen loss=19.196044921875,disc loss=5.8796187659027055e-05, {hms_string(epoch_elapsed)}
Epoch 14962, gen loss=14.19803237915039,disc loss=0.00018095324048772454, {hms_string(epoch_elapsed)}
Epoch 14963, gen loss=12.159148216247559,disc loss=0.09219726920127869, {hms_string(epoch_elapsed)}
Epoch 14964, gen loss=28.447010040283203,disc loss=3.199108362197876, {hms_string(epoch_elapsed)}
Epoch 14965, gen loss=57.978233337402344,disc loss=2.182555675506592, {hms_string(epoch_elapsed)}
Epoch 14966, gen loss=39.34479904174805,disc loss=1.0221270322799683, {hms_string(epoch_elapsed)}
Epoch 14967, gen loss=23.925935745239258,disc loss=0.8541207313537598, {hms_string(epoch_elapsed)}
Epoch 14968, gen loss=11.174646377563477,disc loss=0.5040456056594849, {hms_string(epoch_elapsed)}
Epoch 14969, gen loss=7.4813737869262695,disc loss=0.282467782497406, {hms_string(epoch_elapsed)}
Epoch 14970, gen loss=16.18685531616211,disc loss=0.05077427625656128, {hms_string(epoch_elapsed)}
Epoch 14971, gen loss=16.50716781616211,disc loss=0.06668303161859512, {hms_string(epoch_elapsed)}
Epoch 14972, gen loss=8.940256118774414,disc loss=0.16045399010181427, {hms_string(epoch_elapsed)}
Epoch 14973, gen loss=16.375560760498047,disc loss=0.02177269198000431, {hms_string(epoch_elapsed)}
Epoch 14974, gen loss=21.21765899658203,disc loss=0.0001916298206197098, {hms_string(epoch_elapsed)}
Epoch 14975, gen loss=20.827430725097656,disc loss=0.04587710276246071, {hms_string(epoch_elapsed)}
Epoch 14976, gen loss=14.806978225708008,disc loss=0.03854428976774216, {hms_string(epoch_elapsed)}
Epoch 14977, gen loss=15.297588348388672,disc loss=0.0001457730249967426, {hms_string(epoch_elapsed)}
Epoch 14978, gen loss=15.336530685424805,disc loss=5.468484596349299e-05, {hms_string(epoch_elapsed)}
Epoch 14979, gen loss=15.110705375671387,disc loss=8.430182788288221e-05, {hms_string(epoch_elapsed)}
Epoch 14980, gen loss=12.09630298614502,disc loss=0.0014127036556601524, {hms_string(epoch_elapsed)}
Epoch 14981, gen loss=10.834489822387695,disc loss=0.0028406172059476376, {hms_string(epoch_elapsed)}
Epoch 14982, gen loss=12.900982856750488,disc loss=0.00020956755906809121, {hms_string(epoch_elapsed)}
Epoch 14983, gen loss=14.124222755432129,disc loss=0.0038665197789669037, {hms_string(epoch_elapsed)}
Epoch 14984, gen loss=11.700735092163086,disc loss=0.009959583170711994, {hms_string(epoch_elapsed)}
Epoch 14985, gen loss=8.806962966918945,disc loss=0.02915035933256149, {hms_string(epoch_elapsed)}
Epoch 14986, gen loss=13.386819839477539,disc loss=0.000958634540438652, {hms_string(epoch_elapsed)}
Epoch 14987, gen loss=15.068296432495117,disc loss=0.02442631311714649, {hms_string(epoch_elapsed)}
Epoch 14988, gen loss=10.361167907714844,disc loss=0.004318938124924898, {hms_string(epoch_elapsed)}
Epoch 14989, gen loss=11.826301574707031,disc loss=0.024608470499515533, {hms_string(epoch_elapsed)}
Epoch 14990, gen loss=13.688673973083496,disc loss=0.008769575506448746, {hms_string(epoch_elapsed)}
Epoch 14991, gen loss=16.124404907226562,disc loss=0.013638259842991829, {hms_string(epoch_elapsed)}
Epoch 14992, gen loss=17.53520965576172,disc loss=0.0003468205395620316, {hms_string(epoch_elapsed)}
Epoch 14993, gen loss=15.249874114990234,disc loss=0.00016016815789043903, {hms_string(epoch_elapsed)}
Epoch 14994, gen loss=13.583885192871094,disc loss=0.00012064370093867183, {hms_string(epoch_elapsed)}
Epoch 14995, gen loss=13.694002151489258,disc loss=0.0017913555493578315, {hms_string(epoch_elapsed)}
Epoch 14996, gen loss=12.200922012329102,disc loss=0.0010113099124282598, {hms_string(epoch_elapsed)}
Epoch 14997, gen loss=11.274087905883789,disc loss=0.010249610990285873, {hms_string(epoch_elapsed)}
Epoch 14998, gen loss=16.91964340209961,disc loss=8.673769480083138e-05, {hms_string(epoch_elapsed)}
Epoch 14999, gen loss=17.39356231689453,disc loss=0.0001618165842955932, {hms_string(epoch_elapsed)}
Epoch 15000, gen loss=12.538640975952148,disc loss=0.00027191199478693306, {hms_string(epoch_elapsed)}
Training time: 1:24:15.36
```

Result

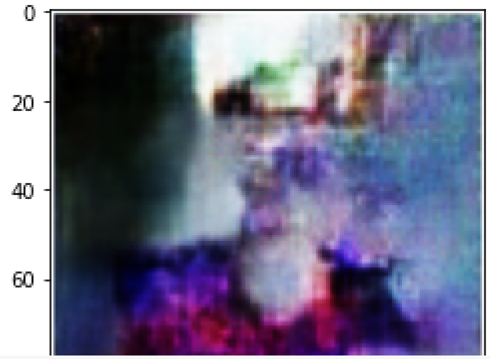
```
# Result after 10365 epochs

import cv2
import matplotlib.pyplot as plt

image=cv2.imread("/content/drive/MyDrive/Seniors/output1/train-10365.png")
plt.imshow(image)
```

```
<matplotlib.image.AxesImage at 0x7f0c8b1e9400>
```



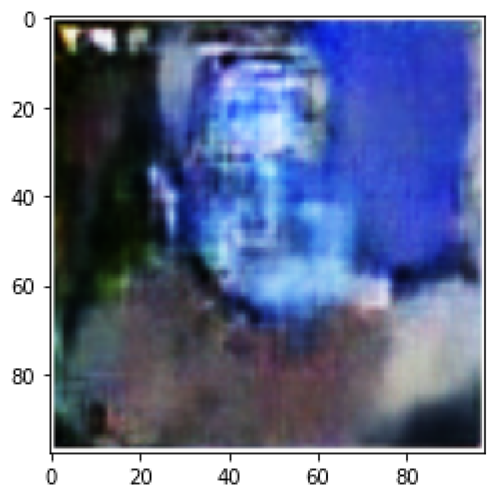

```
# Result after 14899 epochs

a=cv2.imread("/content/drive/MyDrive/Seniors/output1/train-14899.png")
plt.imshow(a)
```

<matplotlib.image.AxesImage at 0x7f0c8c26c7b8>



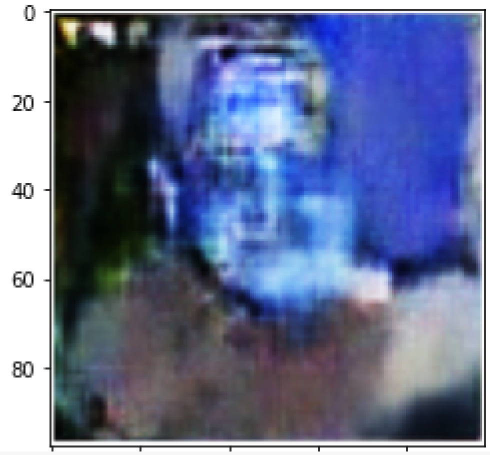
```
# Result after 14991 epochs

a=cv2.imread("/content/drive/MyDrive/Seniors/output1/train-14991.png")
plt.imshow(a)
```

<matplotlib.image.AxesImage at 0x7f0c8c2acd30>



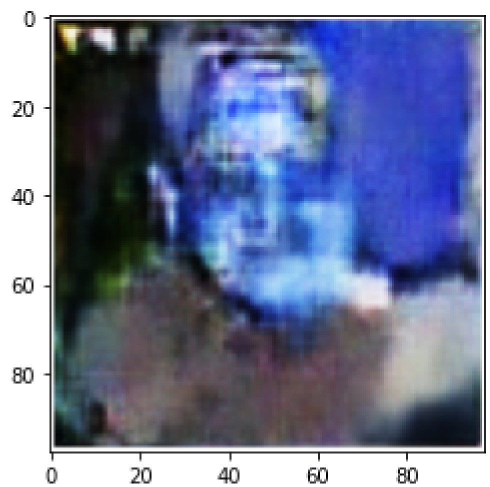
```
# Result after 14992 epochs

a=cv2.imread("/content/drive/MyDrive/Seniors/output1/train-14992.png")
plt.imshow(a)
```

<matplotlib.image.AxesImage at 0x7f0c8b218e80>



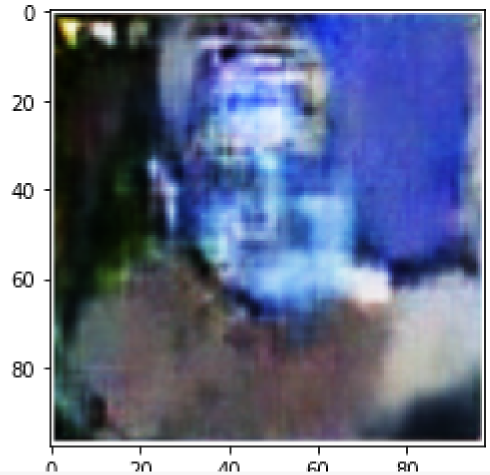
```
# Result after 14993 epochs
a=cv2.imread("/content/drive/MyDrive/Seniors/output1/train-14993.png")
plt.imshow(a)
```

<matplotlib.image.AxesImage at 0x7f0c81a02828>



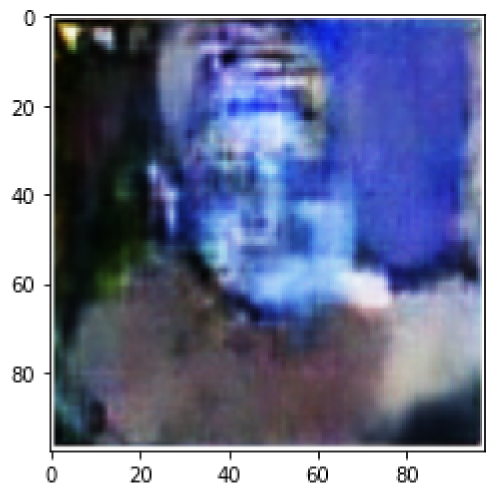

```
# Result after 14994 epochs
a=cv2.imread("/content/drive/MyDrive/Seniors/output1/train-14994.png")
plt.imshow(a)
```

<matplotlib.image.AxesImage at 0x7f0c7bdd49e8>



```
# Result after 14998 epochs
a=cv2.imread("/content/drive/MyDrive/Seniors/output1/train-14998.png")
plt.imshow(a)
```

<matplotlib.image.AxesImage at 0x7f0c7bd27ba8>



```
# Result after 14999 epochs
a=cv2.imread("/content/drive/MyDrive/Seniors/output1/train-14999.png")
plt.imshow(a)
```

```
<matplotlib.image.AxesImage at 0x7f0c7bcfbd30>
```