تمرین سری دهم درس تصویربرداری رقمی

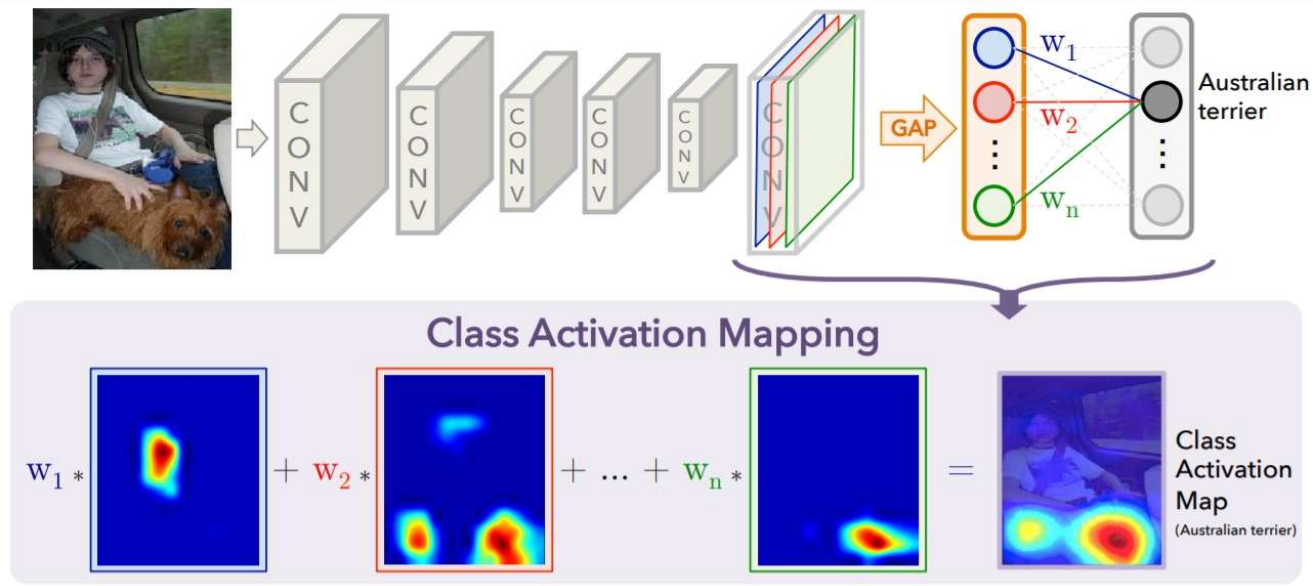
پوریا محمدی نسب

(۴۰۰۷۲۲۱۳۸)

۱- یکی از روشهای معروف برای آشکارسازی اشیاء در تصویر با نظارت ضعیف CAM(Class Activation Map) است. مقاله زیر را مطالعه کنید و روش آن را توضیح دهید. (۲۰ امتیاز)

Zhou, Bolei, et al. "Learning deep features for discriminative localization." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

در مقاله ذکر شده برای تولید class activation map از معماری شبکه ای شبیه به network in network و GoogLeNet که شبکه ای است که در آن به طور وسیعی به لایه های کانولوشنی تاکید دارد و فقط در انتهای معماری برای تسک categorization از Softmax استفاده میشود. از global average pooling در این مقاله روی لایه های کانولوشنی استفاده میشود و از آن ها به عنوان feature های لایه fully-connected استفاده میکند. دلیل استفاده از این تکنیک ساده این است که میتوان با استفاده از وزن های output layer روی convolution feature map ها نواحی مهم تصویر را پیدا کرد.

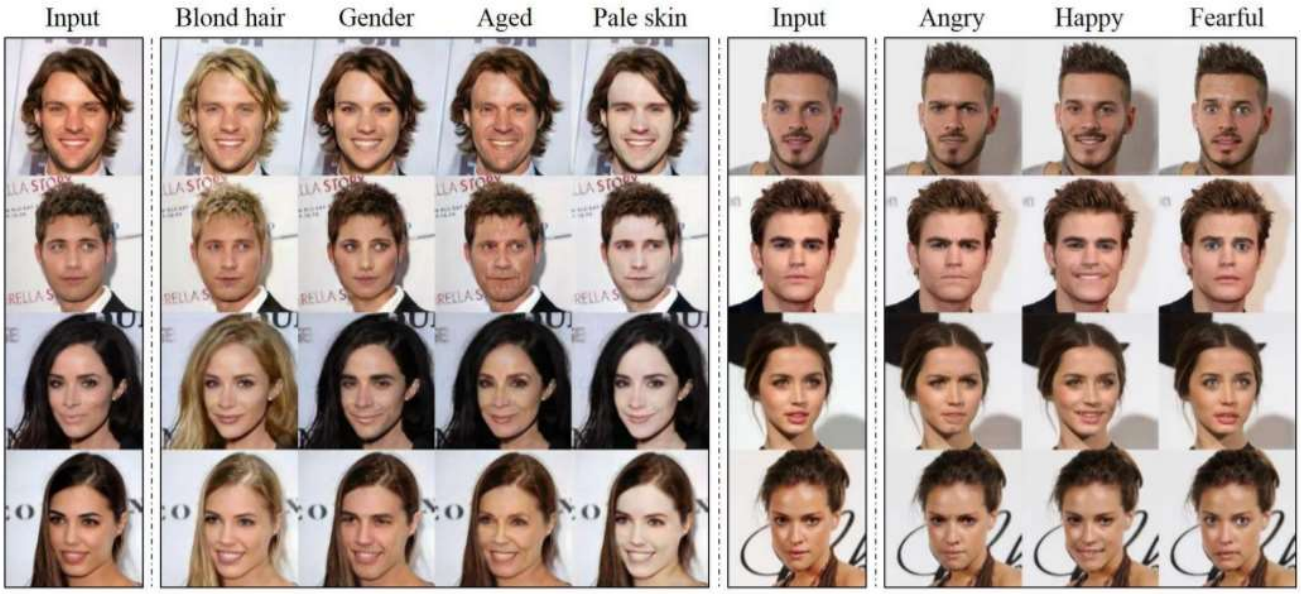


در شکل بالا مشاهده میشود که از global average pool پس از هر لایه convolution استفاده میشود.یک جمع وزن دار بین feature map های خروجی هر pooling خروجی را میسازد. وزن هر لایه نیز به عنوان ورودی (feature map) لایه ی FC است.

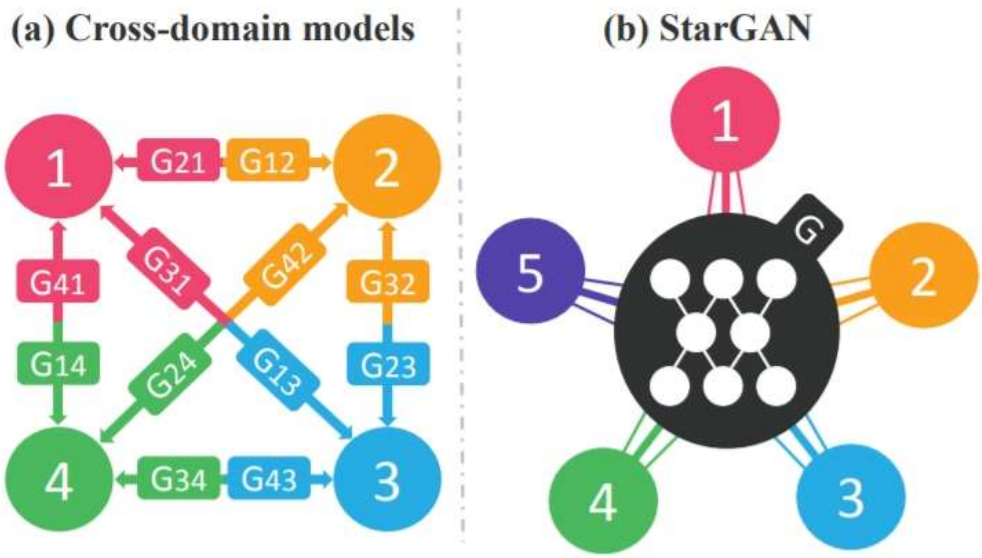
۲- مقاله زیر را مطالعه کرده و گزارشی از آن تهیه کنید. تفاوت اصلی StarGAN و CycleGAN چیست؟(۲۰ امتیاز)

Choi, Yunjey, et al. "Stargan: Unified generative adversarial networks for multi-domain image-to-image translation." Proceedings of the IEEE conference on computer vision and pattern recognition. 2018.

در این مقاله یک متد موثر و کارا برای ترجمه تصویر به تصویر (image to image translation) معرفی شده است. منظور از ترجمه تصویر به تصویر این است که شبکه میتواند با گرفتن یک تصویر ورودی تغییراتی خاص روی تصویر اعمال کند. به طور مثال چهره ی یک انسان که در حالت عادی از او عکس برداری شده را گرفته و چهره ی خندان آن فرد را تولید کند. برای تفهیم بیشتر این موضوع عکس زیر در مقاله آمده است.

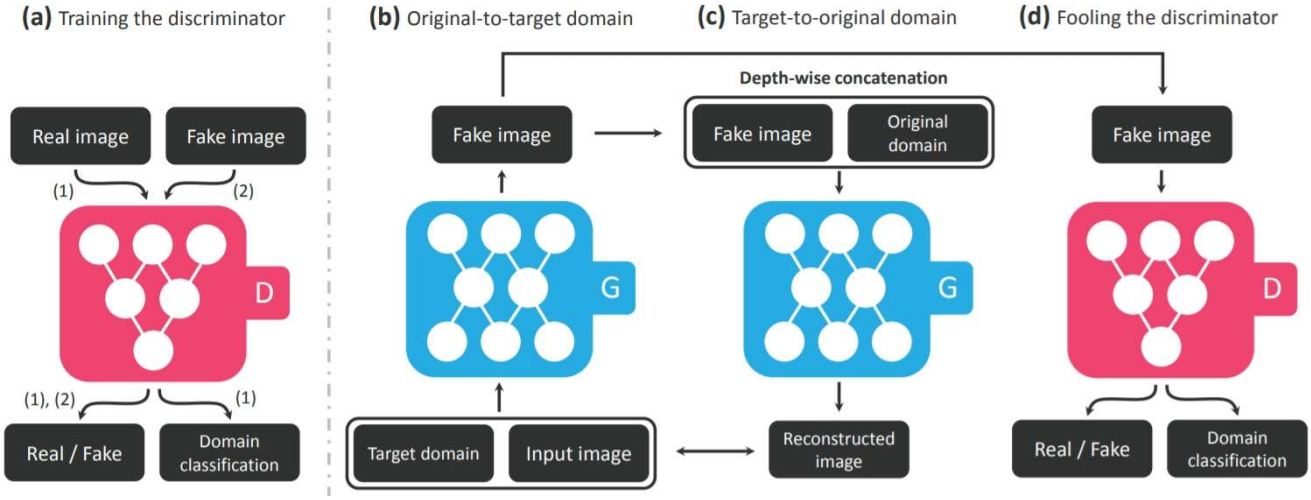


در عکس بالا مشاهده میشودکه میتوان با این روش حالت های مختلف چهره را ایجاد کرد. برای مثال میتوان رنگ موها، جنسیت، سن و حالت صورت (عصبانی، خوشحال و ترسیده) را از چهره ی فرد ساخت. تا قبل از این مقاله الگوریتم های معرفی شده در این زمینه برای کار کردن با بیش از دو domain مشکل داشتند. منظور از domain ها تعداد حالاتی هست که یک معماری از یک چهره میسازد. برای مثال الگوریتمی که از روی چهره حالت خوشحال و عصبانی رد را ایجاد میکند domain برابر ۲ دارد. در این مقاله starGAN معرفی میشود که میتواند multi domain باشد در حالی که فقط از یک مدل استفاده میکند. StarGAN در واقع توانایی training همزمان روی چند دیتاست و چند domain مختلف را دارد.



تصویر بالا کاملا تفاوت مشهود StarGAN با سایر مدل های دیگر را نشان میدهد که معروف به Cross-domain models هستند. در این شبکه ها د حالت multi domain برای هر جفت domain باید از یک cross-domain model استفاده کرد. اما ساختار مدل StarGAN یک تولید کننده ی واحد برای multi domain ها هست.

مدل StarGAN در واقع از دو ماژول اصلی تشکیل میشود. ماژول discriminator (تفکیک کننده) و ماژول generator(تولید کننده). کار ماژول تفیفک کننده این است که یادبگیرد بین تصویر واقعی و ساختگی تمیز قائل شود و تصاویر واقعی را به domain متناظر classify کند. ماژول generator به عنوان ورودی هر دو تصویر واقعی(یکی به عنوان ورودی اصلی و یکی به عنوان تصویر مرجع domain) و domain هدف را میگیرد و یک تصویر fake تولید میکند. سپس برای بهبود نتیجه ماژول generator سعی میکند تصویر اصلی را از روی تصویر fake ایجاد شده بسازد و در اصل تصویر fake ای ایجا کند که غیر قابل تمیز برای ماژول تفیکیک کننده باشد. تصویر زیر مراحل الگوریتم را به خوبی نمایش میدهد.



۳- کد مربوط DCGAN را که در فایل DCGAN.ipynb آمده است را بررسی کرده و مشخص کنید هر قسمت چه عملکردی دارد)نیازی به بررسی کد مربوط به visualization نیست(. سپس با تغییر کد DCGAN.ipynb شده یک conditional GAN طراحی کرده و آموزش دهید. مدل generator آموزش دیده را ذخیره کرده و همراه با تکالیف ارسال کنید(در صورتی که علاقه مند به یادگیری custom loop در tensorflow هستید، می توانید از فایل های موجود در این لینک استفاده کنید). (۳۰ امتیاز)

در ابتدا به تحلیل کد ضمیمه شده میپردازیم. در این کد از دیتاست معروف MNIST استفاده میکنیم. قطعه کد زیر این دیتاست را از داخل کتابخانه ی Keras بارگیری میکند. و چند preprocessing روی دیتاست اعمال میکند(تغییر سایز تصاویر و نرمال کردن مقادیر پیکسل ها بین صفر و یک).

```
[3]  (train_images, train_labels), (_, _) = tf.keras.datasets.mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11493376/11490434 [==============================] - 0s 0us/step
11501568/11490434 [==============================] - 0s 0us/step
```

```
[4]  train_images = train_images.reshape(train_images.shape[0], 28, 28, 1).astype('float32')
     train_images = (train_images - 127.5) / 127.5
```

در ادامه با تعیین دو پارامتر BATCH_SIZE و BUFFER_SIZE دیتاست را قطعه قطعه کرده و همچینین دیتای داخل دیتاست را به هم میزنیم تا از bias احتمالی جلوگیری کنیم.

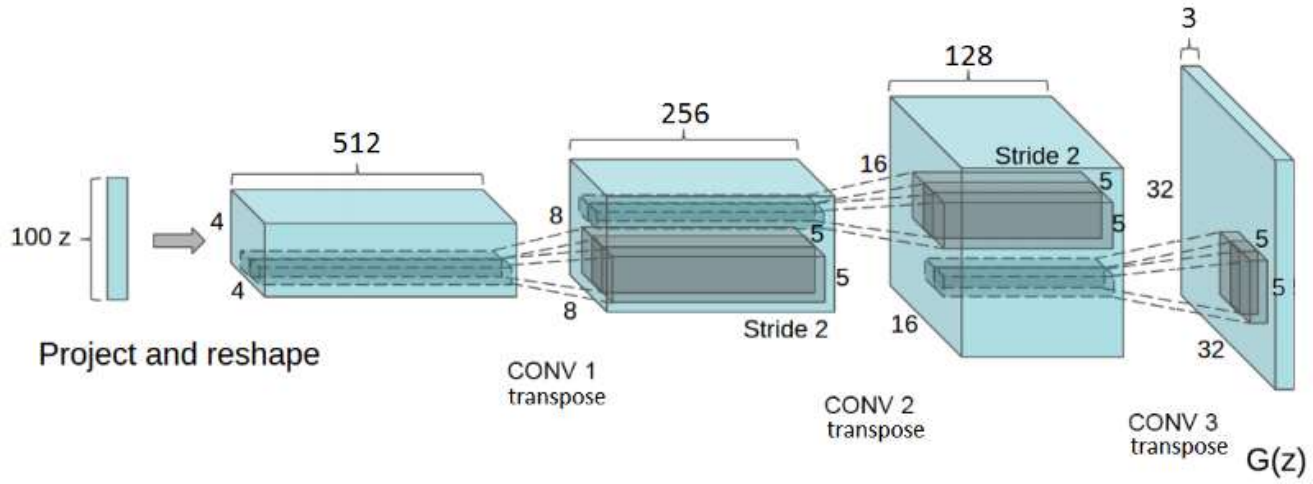
```
[5]  BUFFER_SIZE = 60000
     BATCH_SIZE = 256
```

```
[6]  train_dataset = tf.data.Dataset.from_tensor_slices(train_images).shuffle(BUFFER_SIZE).batch(BATCH_SIZE)
```

در شبکه های GAN باید دو ماژول Generator و Discriminator باید تعریف شوند ک هر کدام معماری خاص خود را دارند. در ابتدا معماری ماژول تولیدکننده تعریف شده است:

```
def make_generator_model():
    model = tf.keras.Sequential()
    model.add(layers.Dense(7*7*256, use_bias=False, input_shape=(100,)))
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU())

    model.add(layers.Reshape((7, 7, 256)))
    assert model.output_shape == (None, 7, 7, 256)  # Note: None is the batch size

    model.add(layers.Conv2DTranspose(128, (5, 5), strides=(1, 1), padding='same', use_bias=False))
    assert model.output_shape == (None, 7, 7, 128)
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU())

    model.add(layers.Conv2DTranspose(64, (5, 5), strides=(2, 2), padding='same', use_bias=False))
    assert model.output_shape == (None, 14, 14, 64)
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU())

    model.add(layers.Conv2DTranspose(1, (5, 5), strides=(2, 2), padding='same', use_bias=False, activation='tanh'))
    assert model.output_shape == (None, 28, 28, 1)

    return model
```

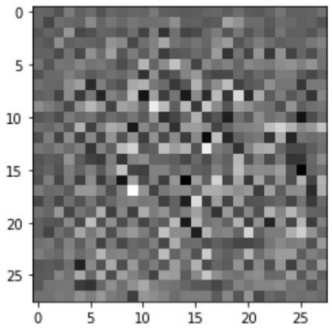
برای درک بهتر قطعه کد بالا که مربوط به معماری generator در DCGAN است شکل معماری را در پایین مشاهده میکنید.



پس از تعریف کردن معماری generator یک مدل از آن میسازیم. به عنوان ورودی اول Dense آن، اعدادی رندم با توزیع نرمال وارد میکنیم. و نتیجه مدل (تصویر تولید شده) را که یک تصویر RGB است را به صورت grayscale نمایش میدهیم.

```
generator = make_generator_model()

noise = tf.random.normal([1, 100])
generated_image = generator(noise, training=False)

plt.imshow(generated_image[0, :, :, 0], cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x7f40b04bfb50>
```



در این مرحله ماژول تفکیک کننده ی شبکه را مدلسازی میکنیم. این ماژول معماری نسبتا ساده ای دارد و به نحوی ابعاد آن به صورت متقارن برابر با معماری generator است. تصویر پایین این معماری را نمایش میدهد.

```
def make_discriminator_model():
    model = tf.keras.Sequential()
    model.add(layers.Conv2D(64, (5, 5), strides=(2, 2), padding='same',
                            input_shape=[28, 28, 1]))
    model.add(layers.LeakyReLU())
    model.add(layers.Dropout(0.3))

    model.add(layers.Conv2D(128, (5, 5), strides=(2, 2), padding='same'))
    model.add(layers.LeakyReLU())
    model.add(layers.Dropout(0.3))

    model.add(layers.Flatten())
    model.add(layers.Dense(1))

    return model
```

قطعه کد بالا پیاده سازی مدل تفکیک کننده است.

در ادامه مدلی از این تفکیک کننده میسازیم و تصویر تولید شده ی ماژول تولیدکننده را به آن پاس میدهیم تا تصمیم خروجی این مدل را ببینیم.

```
[10] discriminator = make_discriminator_model()
     decision = discriminator(generated_image)
     print (decision)

     tf.Tensor([[0.00253611]], shape=(1, 1), dtype=float32)
```

خروجی مدل عددی بسیار نزدیک به صفر است که نشان میدهد تفکیک کننده این تصویر را رد کرده است. برای ادامه کار، باید توابع loss برای مدل کلی تعریف شود.

```
[11] cross_entropy = tf.keras.losses.BinaryCrossentropy(from_logits=True)
```

```
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
```

```
[13] def generator_loss(fake_output):
         return cross_entropy(tf.ones_like(fake_output), fake_output)
```

پارامترهای شبکه شامل نوع optimizer دو ماژول که هر دو Adam هستند. تعداد epoch ها 50 ، ابعاد نویز متناسب با لایه اول generator 100 و تعداد example هایی که ایجاد میشوند 16 قرار داده میشود.

```
generator_optimizer = tf.keras.optimizers.Adam(1e-4)
discriminator_optimizer = tf.keras.optimizers.Adam(1e-4)
```

```
[15] checkpoint_dir = './training_checkpoints'
     checkpoint_prefix = os.path.join(checkpoint_dir, "ckpt")
     checkpoint = tf.train.Checkpoint(generator_optimizer=generator_optimizer,
                                      discriminator_optimizer=discriminator_optimizer,
                                      generator=generator,
                                      discriminator=discriminator)
```

```
[16] EPOCHS = 50
     noise_dim = 100
     num_examples_to_generate = 16
     seed = tf.random.normal([num_examples_to_generate, noise_dim])
```

در ادامه تابع train_step نوشته شده در کتابخانه tensorflow را بازنویسی(overwrite) میکنیم. به این شکل که در ابتدا نویزی ایجاد میکنیم و آن را به generator میدهیم سپس تصویر اصلی و تصویر جعلی را به ترتیب به discriminator داده و خروجی آن ها را به ترتیب به تابع generator loss و discriminator داده میشود. در ادامه مشتق خروجی نسبت به وزن های trainable محاسبه میشود. و این مشتق ها به optimizer های دو ماژول داده میشوند تا عمل بهینه سازی انجام شود.

```
@tf.function
def train_step(images):
    noise = tf.random.normal([BATCH_SIZE, noise_dim])

    with tf.GradientTape() as gen_tape, tf.GradientTape() as disc_tape:
      generated_images = generator(noise, 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))
```

در ادامه تعریف تابع train را داریم که در این تابع با دریافت دیتاست و تعداد epoch ها با توجه به batch_size تصاویر از دیتاست دریافت میشوند. پس از نمایش آنها تصاویر توسط generator تولید و ذخیره میشوند. در هر 15 epoch نیز اطلاعات checkpoint ذخیره میشود تا عملکرد مدل برای تحلیل های آینده مشخص باشد.

```
def train(dataset, epochs):
  for epoch in range(epochs):
    start = time.time()
    for image_batch in dataset:
      train_step(image_batch)

    display.clear_output(wait=True)
    generate_and_save_images(generator,epoch + 1,seed)

    if (epoch + 1) % 15 == 0:
      checkpoint.save(file_prefix = checkpoint_prefix)

    print ('Time for epoch {} is {} sec'.format(epoch + 1, time.time()-start))

  display.clear_output(wait=True)
  generate_and_save_images(generator,epochs,seed)
```

تابع generate_and_save_images نیز تابعی ساده است که prediction مدل را محاسبه کرده و آن را نمایش میدهد.

```
def generate_and_save_images(model, epoch, test_input):
  predictions = model(test_input, training=False)

  fig = plt.figure(figsize=(4, 4))

  for i in range(predictions.shape[0]):
      plt.subplot(4, 4, i+1)
      plt.imshow(predictions[i, :, :, 0] * 127.5 + 127.5, cmap='gray')
      plt.axis('off')

  plt.savefig('image_at_epoch_{:04d}.png'.format(epoch))
  plt.show()
```

در انتهای کد مدل را train میکنیم. و خروجی شبکه در انتهای هر epoch مشاهده میشود و میتوان دریافت که شبکه رو به بهبودی حرکت میکند و نتایج به مرور بهتر میشوند. تصویر نهایی را در شکل زیر مشاهده میکنیم.

✓ 12m [39] `train(train_dataset, EPOCHS)`

۴- فرض کنید قصد طراحی CycleGAN را دارید و متغیرهای realA ، realB، genAB، genBA، discA و discB به ترتیب تصاویر واقعی از دامنه A، تصاویر واقعی از دامنه B ، مدل مولد تبدیل دامنه A به دامنه B ، مدل مولد تبدیل دامنه B به دامنه A ، مدل ممیز دامنه A و مدل ممیز دامنهB باشند. یک شبه کد برای تابع ضرر (loss function) آموزش genAB بنویسید. (۲۰ امتیاز)

راهنمایی: شما باید دو تابع ضرر کلی را در نظر بگیرید : تابع ضرر adversarial ، تابع ضرر مربوط به . cycle consistency تابع ضرر اول همان تابع ضرر رایج در شبکه های GAN است و تابع ضرر دوم برای سنجش معکوس بودن دو مدل مولد.

```
# For CycleGAN, we need to calculate different
        # kinds of losses for the generators and discriminators.
        # We will perform the following steps here:
        #
        # 1. Pass real images through the generators and get the generated images
        # 2. Pass the generated images back to the generators to check if we
        #    we can predict the original image from the generated image.
        # 3. Do an identity mapping of the real images using the generators.
        # 4. Pass the generated images in 1) to the corresponding discriminators.
        # 5. Calculate the generators total loss (adverserial + cycle + identity)
        # 6. Calculate the discriminators loss
        # 7. Update the weights of the generators
        # 8. Update the weights of the discriminators
        # 9. Return the losses in a dictionary
```

۵- با جستجو به سوالات زیر پاسخ دهید:

۵- الف) منظور از mode collapse در شبکه های GAN چیست؟ (۱۰ امتیاز)

معمولا انتظاری که از شبکه های GAN داریم این است که بتوانند طیف وسیعی از خروجی ها را تولید کنند. به طور مثال ما انتظار داریم به ازای هر تصویر ورودی از یک چهره ی تصادفی یک خروجی متفاوت با سایرین مشاهده کنیم. اما اگر generator برای حالتی خاص یک خروجی بسیار خوب داشته باشد ممکن است که یاد بگیرد در ادامه فقط آن خروجی را ایجاد کند. به عبارت دیگر generator سعی میکند فقط خروجی تولید کند که از نظر ماژول تفکیک کننده بسیار مناسب باشد. و این باعث میشود که ماژول تفکیک کننده بر اساس strategy که دارد آن output را reject کند و اگر در local minima گیر کند خارج شدن از آن کار بسیار مشکلی خواهد بود زیر نیاز است که best strategy را بیابد.

۵- ب) منظور از convergence failure در شبکه های GAN چیست و چه راههایی برای شناسایی آن وجود دارد؟ (۱۰ امتیاز)

اساسا در شبکه های GAN عمل training بسیار دشوار است زیرا هر دو ماژول generator و discriminator به صورت موازی و همزمان از ابتدا train میشوند و این یعنی بهبود در یک ماژول باعث ایجاد هزینه و ضرر در ماژول دیگر خواهد بود. در ابتدا اصلاح convergence failure را بررسی میکنیم. این اصطلاح بدین معنی است که در حین train کردن شبکه loss تعریف شده کم نمیشود و ثابت میماند. در شبکه های GAN این اصطلاح زمانی به کار گرفته میشود که نتوانسته باشیم trade off بین دو ماژول شبکه را بیابیم. یک راه ساده برای تشخیص این مشکل این است که loss ماژول discriminator به صفر یا نزدیک صفر رسیده باشد. در بعضی موارد loss ماژول generator افزایش میابد.

References

1) openaccess.thecvf.com/content_cvpr_2016/papers/Zhou_Learning_Deep_Features_CVPR_2016_paper

2) M. Lin, Q. Chen, and S. Yan. Network in network. International Conference on Learning Representations, 2014. 1, 2,4

3) openaccess.thecvf.com/content_cvpr_2018/papers/Choi_StarGAN_Unified_Generative_CVPR_2018

4) https://github.com/yunjey/stargan

5) https://machinelearningmastery.com/practical-guide-to-gan-failure-modes/

6) https://developers.google.com/machine-learning/gan/problems

7) https://machinelearningmastery.com/how-to-develop-cyclegan-models-from-scratch-with-keras/

8) https://keras.io/examples/generative/cyclegan/

9) https://www.tensorflow.org/tutorials/generative/cyclegan