Assignment NO.10 Solutions

Digital Image Processing | Fall 1400 | Dr.Mohammadi

Teacher Assistant : Ramin Kamali

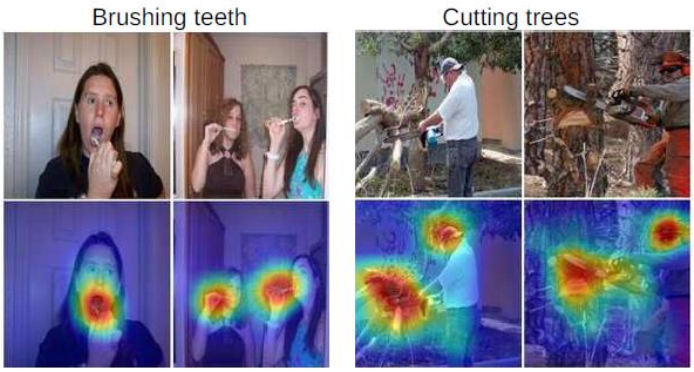
Student name : **Amin Fathi**

Student id : **400722102**

Problem 1

علی‌رغم اینکه هیچ نظارتی بر محل اشیا وجود ندارد، واحدهای کانولوشن لایه‌های مختلف CNN، به‌عنوان object detectors عمل می‌کنند. با وجود داشتن این توانایی قابل‌توجه در لایه‌های کانولوشن، این توانایی زمانی که از لایه‌های Fully Connected برای طبقه‌بندی استفاده می‌شود، از بین می‌رود. اخیراً برخی از شبکه‌های عصبی کاملاً پیچیده مانند Network in Network (NIN) و GoogLeNet برای اجتناب از استفاده از لایه‌هایFully Connected برای به حداقل رساندن تعداد پارامترها و در عین حال بالا بودن کارایی، پیشنهاد شده‌اند. برای دستیابی به این هدف، از global average pooilg استفاده می‌شود که به‌عنوان یک تنظیم‌کننده ساختاری (structural regularizer) عمل می‌کند و از overfitting در طول آموزش جلوگیری می‌کند. در آزمایش‌هایمان، متوجه شدیم که مزایای این global average pooilg فراتر از عمل کردن به‌عنوان یک منظم‌کننده است و در واقع، با کمی تغییر، شبکه می‌تواند توانایی محلی‌سازی قابل‌توجه خود را تا لایه نهایی حفظ کند. این بهینه‌سازی امکان شناسایی آسان مناطق تصویر متمایز را در یک گذر به جلو برای طیف گسترده ای از وظایف، حتی آن‌هایی که شبکه در ابتدا برای آن‌ها آموزش ندیده است، می‌دهد.

همان‌طور که در شکل زیر نشان داده شده است، یک CNN آموزش دیده با استفاده از یک تغییر جزئی در global average pooilg (GAP) همراه با تکنیک Class Activation Mapping (CAM)، در زمینه طبقه‌بندی اشیا با موفقیت قادر است تا مناطق متمایز را برای طبقه‌بندی محلی‌سازی کند. به عبارتی هم می‌تواند طبقه‌بندی کند و هم آن را محلی‌سازی کند. به عنوان مثال در تصویر زیر، مسواک برای مسواک زدن و اره برای بریدن درختان.



کارهای اخیر نشان داده است که علیرغم آموزش بر روی برچسب‌های سطح تصویر، CNNها توانایی قابل توجهی در بومی‌سازی اشیا دارند. می‌توان با استفاده از یک معماری مناسب، این توانایی را فراتر از بومی‌سازی اشیاء نیز تعمیم داد، همچنین شروع به شناسایی دقیق مناطقی از یک تصویر کرد تا برای تمایز استفاده شوند.

در اینجا، ما دو خط کار مرتبط با این مقاله را مورد بحث قرار می‌دهیم که یکی از آن ها محلی‌سازی شی با نظارت ضعیف (Weakly-supervised object localization) است.

اخیراً تعدادی کار در مورد بومی‌سازی شی با نظارت ضعیف با استفاده از CNN انجام شده است.

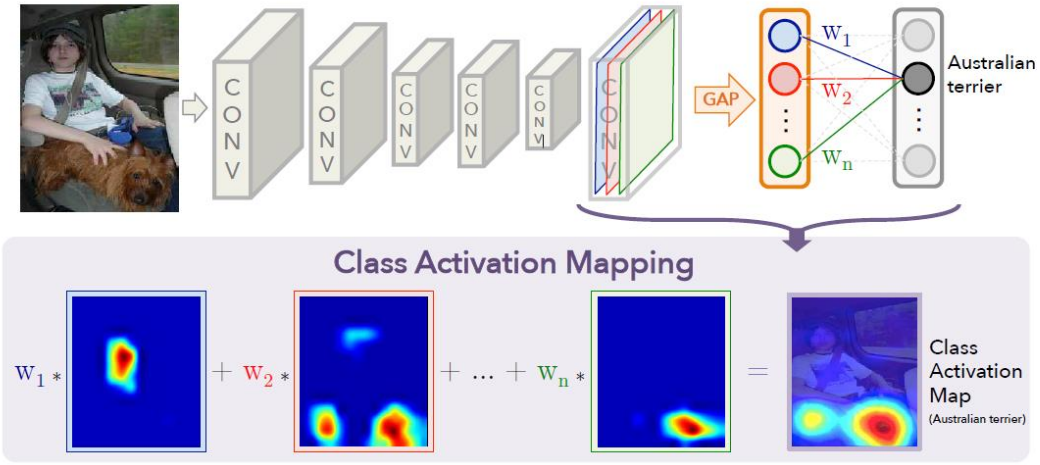
- پوشاندن مناطق تصویر برای شناسایی مناطقی که حداکثر فعال‌سازی را برای بومی‌سازی اشیا ایجاد می‌کنند.
- یادگیری چند نمونه با استفاده از ویژگی های CNN برای بومی سازی اشیا.
- انتقال نمایش‌های تصویر در سطح میانی

با این حال، این رویکردها در واقع توانایی محلی سازی را ارزیابی نمی‌کنند. از سوی دیگر، در حالی که این رویکردها نتایج امیدوارکننده‌ای را به همراه دارند، اما به صورت کلی آموزش داده نمی‌شوند و برای بومی سازی اشیاء به چندین پاس رو به جلو از یک شبکه نیاز دارند که این امر مقایسه‌ی آن‌ها را برای مجموعه داده‌های دنیای واقعی دشوار می‌کند. رویکرد معرفی شده در این مقاله، از انتها به انتها آموزش داده شده است و می‌تواند اشیاء را در یک پاس رو به جلو بومی‌سازی کند.

در این مقاله از class activation map استفاده شده است که به دلیل سادگی، آن را قابل‌حمل می‌کند و می‌توان برای مکان‌یابی سریع و دقیق در انواع وظایف بینایی ماشین اعمال کرد.

در ادامه روش تولید CAM با استفاده از GAP در CNNها شرح داده می‌شود. همانطور که در شکل زیر مشاهده می‌شود، CAM مناطق تصویر متمایز شده توسط CNN برای شناسایی آن دسته را نشان می‌دهد.

معماری CAM نیز در شکل زیر نشان داده شده است که در آن از معماری NIN و GoogleNet استفاده شده است. معماری به این صورت است که از تعداد زیادی لایه‌ی کانولوشن استفاده شده است و درست قبل از خروجی نهایی، لایه‌ی Softmax قرار داده شده است. GAP روی کانولوشن انجام می‌شود و ویژگی‌های به دست آمده به عنوان ویژگی‌هایی برای لایه‌ی Fully Connected، جهت تولید خروجی استفاده می‌شوند. با استفاده از این ساختار، می‌توان اهمیت مناطق تصویر را با بازتاب وزن لایه‌ی خروجی روی نقشه‌های وژگی کانولوشن تشخیص داد، تکنیکی که به آن CAM گفته می‌شود.



در ادامه توانایی محلی‌سازی CAM را هنگامی که بر روی مجموعه داده معیار ILSVRC 2014 آموزش می‌بینند، ارزیابی می‌شود.

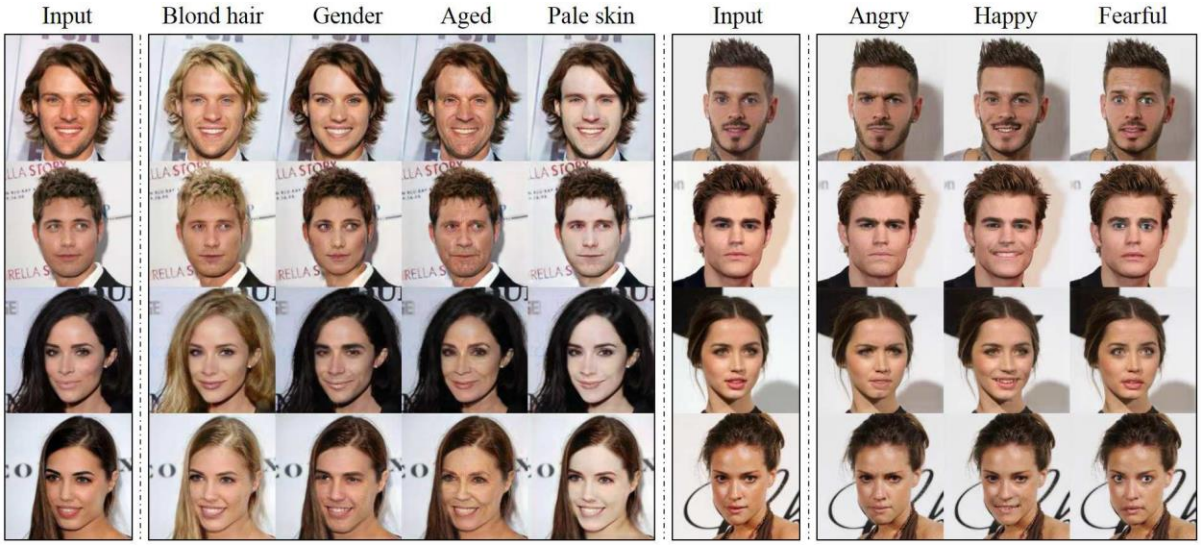
تاثیر استفاده از CAM بر روی انواع شبکه‌های کانولوشنی در ادامه بررسی می‌شود. به طور کلی، برای هر یک از شبکه‌های کانولوشنی، لایه‌های Fully Connected قبل از خروجی نهایی حذف شده و با GAP و سپس یک لایه‌ی Softmax جایگزین می‌شوند. حذف این لایه‌های Fully Connected تا حد بسیار خوبی پارامترهای شبکه را کاهش می‌دهد؛ البته علاوه بر آن، باعث کاهش عملکرد طبقه‌بندی نیز می‌شود. با توجه به بررسی‌های صورت گرفته، این نتیجه حاصل شد که توانایی محلی‌سازی شبکه‌ها زمانی بهبود می‌یابد که آخرین لایه کانولوشن قبل از GAP وضوح فضایی بالاتری داشته باشد که به آن mapping resolution گفته می‌شود. برای انجام این کار چندین لایه‌ی کانولوشن را از برخی شبکه‌ها حذف کرده تا وضوح افزایش یابد. به طول مثال برای AlexNet، لایه‌ی بعد از conv5 حذف شد (یعنی pool5 به prob). برای VGGnet، لایه‌های بعد از conv5-3 حذف گردید (یعنی pool5 به prob). برای GoogleNet، لایه‌های بعد از inceptione4 حذف گردیدند (یعنی pool4 به prob). همچنین برای هر یک از این دو شبکه یک لایه‌ی کانولوشن به اندازه ۳*۳ و به دنبال آن یک لایه GAP و یک لایه Softmax اضافه شد. در ادامه هر یک از شبکه‌ها بر روی تصاویر مدنظر آموزش داده شد و شبکه‌های نهایی AlexNet-GAP، VGGnet-GAP و GooLeNet-Gap ایجاد شدند.

نتیجه دریافتی بیان می‌کند که رویکرد استفاده شده به طور قابل توجهی به عملکرد طبقه‌بندی آسیب نمی‌زند. سپس نشان داده شد که رویکرد استفاده شده در محلی‌سازی شی با نظارت ضعیف، موثرتر است. نتیجه شد که در بیشتر موارد در هنگام حذف لایه‌های اضافی از شبکه‌های مختلف، یک افت یک الی دو درصدی در عملکرد آن‌ها مشاهده می‌شود که برای جبران می‌توان از دو لایه‌ی کانولوشن، قبل از GAP استفاده کرد. به این ترتیب شبکه‌ی AlexNet*-GAP پدید می‌آید که عملکردی مشابه AlexNet دارد. در نتیجه عملکرد طبقه‌بندی تا حدود خوبی برای شبکه‌های GAP حفظ می‌شود.

به صورت کلی این نتیجه در انتهای مقاله کسب شده است که CAM برای CNNها با GAP پیشنهاد می‌شود. این CNNهای آموزش دیده با طبقه‌بندی را قادی می‌سازد تا بدون استفاده از هیچ حاشیه‌نویسی، محلی‌سازی را یاد بگیرند. قسمت‌های تشخیص داده شده توسط CNN برجسته می‌شود. بنابراین CNNهای ادغام شده می‌توانند محلی‌سازی دقیق شئ را انجام دهند. علاوه بر این، این تکنیک‌ها به سایر وظایف تشخیص نیز تعمیم می‌یابد.

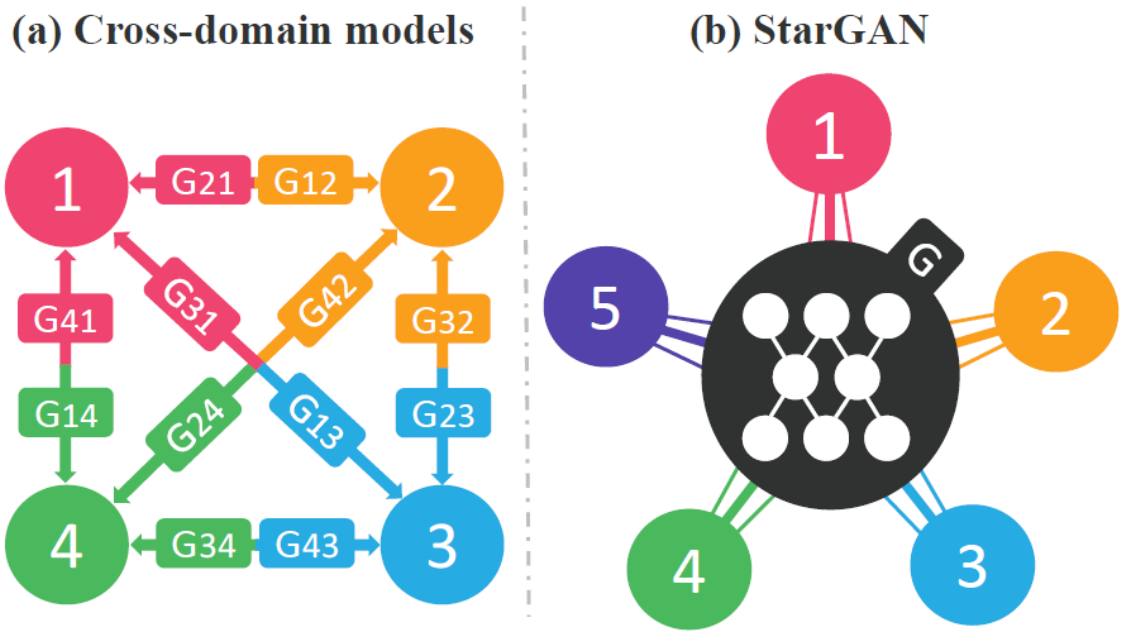
Problem 2

image-to-image یک جنبه‌ی خاص از تصویر داده شده را به دیگری تغییر می‌دهد. به طور مثال تغییر چهره‌ی یک فرد از خندان به اخم که در شکل زیر مشاهده می‌شود.



باتوجه به داده‌های آموزشی، این مدل‌ها یاد میگیرند که تصاویر را از یک دامنه به دامنه دیگر ترجمه کنند. Attribute را به عنوان یک ویژگی معنادار ذاتی در یک تصویر در نظر می‌گیریم (مانند رنگ مو، جنسیت یا سن) و مقدار ویژگی را به عنوان مقدار خاصی از یک ویژگی درنظر می‌گیریم (مانند سیاه، بور و قهوه‌ای یا مردان و زنان برای ویژگی جنسیت). مجموعه‌ای از تصاویر را نیز دامنه می‌نامیم. مثلا تصاویر زنان می‌تواند یک حوزه را نشان دهد و تصاویر مردان نیز حوزه دیگری را. در این مقاله دو مجموعه داده‌ی CelebA شامل ۴۰ برچسب مربوط به ویژگی‌هایی مانند رنگ مو، جنسیت و سن و RaFD دارای ۸ برچسب برای حالات چهره بررسی می‌شود. میخواهیم image-to-image چند دامنه انجام دهیم، به عبارتی تصاویر را باتوجه به ویژگی‌های چندین دامنه تغییر دهیم. در شکل بالا نشان داده شده است که یک تصویر را می‌توان براساس هر یک از چهار حوزه "Blond hair"، "Gender"، "Aged" و "Pale skin" ترجمه کرد. می‌توان چندین دامنه از مجموعه داده‌های مختلف را نیز آموزش داد، مانند آموزش مشترک تصاویر CelebA و RaFD برای تغییر حالت چهره یک تصویر CelebA با استفاده از ویژگی‌های یادگرفته شده توسط آموزش روی داده‌های RaFD. البته تاکنون مدل‌هایی که در این زمینه‌ی آموزش چندین دامنه وجود دارند، کارایی خوبی ندارند و به عبارتی به دلیل تعداد نگاشت‌های بسیار زیادی که دارا هستند، ناکارآمد هستند. این مولدها نمی‌توانند به طور کامل از داده‌های آموزشی استفاده کنند و این موجب پایین آمدن کیفیت تصاویر تولید شده می‌شود. علاوه بر این، آن‌ها قادر به آموزش مشترک دامنه‌ها از مجموعه داده‌های مختلف نیز نیستند.

در این مقاله راه حلی تحت عنوان StarGAN را پیشنهاد می‌کنیم که یک شبکه متخاصم مولد است که قادر به یادگیری نقشه‌برداری در میان دامنه‌های متعدد است. همانطور که در شکل زیر نشان داده شده است، مدل ما داده‌های آموزشی چندین حوزه را دریافت می‌کند و نگاشت بین تمام دامنه‌های موجود را تنها با استفاده از یک مولد یاد می‌گیرد. این مدل هم تصویر و هم اطلاعات دامنه را به عنوان ورودی می‌گیرد و یاد میگیرد که تصویر ورودی را به دامنه‌ی مربوطه ترجمه کند. بردار ماسک تعریف می‌شود که آموزش مشترک بین دامنه‌های مجموعه داده‌های مختلف را امکان‌پذیر می‌کند. این مدل labelهای ناشناخته را نادیده گرفته و بر آنچه ارائه شده است تمرکز می‌کند.



شبکه‌های تخاصم مولد (GANs)، نتایج قابل توجهی را در حوزه‌ی بینایی کامپیوتر، تولید تصویر، ترجمه تصویر و ... نشان داده اند. GAN از دو ماژول تشکیل شده است. یک ماژول تفکیک‌کننده و یک مولد. ماژول تفکیک‌کننده جهت یادگیری نمونه‌های واقعی از جعلی و ماژول مولد جهت تولید نمونه‌های جعلی عمل می‌کند.

اخیرا نتایج چشمگیری در حوزه‌ی ترجمه image-to-image بدست آمده است. CycleGAN و DiscoGAN ویژگی‌های کلیدی را بین ورودی و تصویر ترجمه شده حفظ می‌کنند. با این حال، همه این چارچوب‌ها تنها قادر به یادگیری روابط بین دو حوزه مختلف در یک زمان هستند. رویکردهای آن‌ها مقیاس‌پذیری محدودی در مدیریت چندین دامنه دارد، زیرا مدل‌های مختلف باید برای هر جفت دامنه آموزش داده شوند. برخلاف رویکردهای فوق، چارچوب ما می‌تواند روابط بین چندین حوزه را تنها با استفاده از یک مدل بیاموزد.

StarGAN میتواند به طور همزمان مجموعه داده‌های متعددی را شامل شود که انواع مختلف labelها را شامل می‌شود. StarGAN می‌تواند تمام labelها را کنترل کند. البته این مسئله نیز وجود دارد که اطلاعات labelها فقط تا حدی برای هر مجموعه داده شناخته شده است که این باعث ایجاد مشکل به هنگام بازسازی تصویر ورودی از تصویر ترجمه شده می‌شود.

Mask Vector: برای کاهش مشکل اشاره شده، بردار Mask معرفی می‌شود که به StarGAN اجازه می‌دهد labelهای نامشخص را نادیده بگیرد و بر روی label ارائه شده توسط مجموعه داده تمرکز کند. StarGAN مقدار labelهای ناشناخته را برابر با صفر قرار میدهد.

Training Strategy: هنگام آموزش StarGAN با مجموعه داده‌های متعدد از labelدامنه تعریف شده در معادله استفاده می‌کنیم. با انجام این کار، مولد یاد می‌گیرد که labelهای نامشخص که مقدارشان برابر با صفر است را نادیده گرفته و بر روی labelهای شناخته شده تمرکز کند. از طرف دیگر، تفکیک‌کننده را برای تولید توزیع احتمال روی labelها برتی همه مجموعه داده‌ها گسترش می‌دهیم. سپس مدل را در یک محیط یادگیری multi-task آموزش می‌دهیم.

در این میان از DIAT و CycleGAN به عنوان مدل‌های پایه استفاده می‌شود که هر دو ترجمه image-to-image را در دو حوزه‌ی مختلف انجام می‌دهند. CycleGAN از یک adversarial loss برای نگاشت بین دو دامنه مختلف X و Y استفاده می‌کند. این روش به دو مولد و دو تفکیک‌کننده برای هر جفت دامنه نیاز دارد.

ابتدا روش پیشنهادی را با مدل‌های پایه مقایسه می‌کنیم.

نتایج تجربی حاصل بر روی مجموعه داده‌ی CelebA در جدول زیر نشان داده شده است. همانطور که مشاهده می‌شود، بهترین انتقال ویژگی‌ها مربوط به StarGAN است.

Method	Hair color	Gender	Aged
DIAT	9.3%	31.4%	6.9%
CycleGAN	20.0%	16.6%	13.3%
IcGAN	4.5%	12.9%	9.2%
StarGAN	**66.2%**	**39.1%**	**70.6%**

نتایج حاصل بر روی مجموعه داده‌ی RaFD نیز نشان می‌دهد که StarGAN طبیعی‌ترین حالات را ایجاد می‌کند، درحالی که به درستی هویت شخصی و ویژگی‌های چهره ورودی را حفظ می‌کند. درحالی که دو مدل پایه‌ی دیگر، علی‌رغم حفظ هویت ورودی، نتایج آن‌ها تار نشان داده می‌شود.

همانطور که در جدول زیر نشان داده شده است، مدل ما به کمترین خطای طبقه‌بندی دست می‌یابد که نشان دهده‌ی این موضوع است که این مدل واقعی‌ترین حلات چهره را در بین تمام روش‌های دیگر تولید می‌کند.

Method	Classification error	# of parameters
DIAT	4.10	52.6M × 7
CycleGAN	5.99	52.6M × 14
IcGAN	8.07	67.8M × 1
StarGAN	**2.12**	**53.2M × 1**
Real images	0.45	-

در ستون اخر جدول بالا مشاهده می‌شود که تعداد پارامتر های مورد نیاز در فرایند یادگیری در مدل StarGAN از مدل‌های دیگر کمتر است که این نیز یکی از مزیت‌های مهم این مدل به شمار می‌رود.

در ادامه نیز نتایج تجربی حاصل در استفاده از دو مجموعه داده‌ی CelebA و RaFD به طور همزمان بررسی می‌شود. نشان می‌دهیم که مدل StarGAN نه تنها از چندین دامنه در یک مجموعه داده، بلکه از مجموعه داده‌های متعدد نیز می‌تواند یاد بگیرد. مدل را به صورت مشترک بر روی مجموعه داده‌های CelebA و RaFD با استفاده از Mask Vector آموزش می‌دهیم. برای تمایز بین مدلی که فقط بر روی RaFD آموزش داده شده و مدلی که در هر دو CelebA و RaFD آموزش داده شده است، اولی را به عنوان StarGAN-SNG (تک) و دومی را به عنوان StarGAN-JNT (مشترک) نشان می‌دهیم.

همانطور که در شکل زیر نشان داده شده است، StarGAN-JNT عبارات احساسی را با کیفیت بصری بالا به نمایش می گذارد، در حالی که StarGAN-SNG تصاویر معقول اما تار با پس زمینه خاکستری ایجاد می‌کند. این تفاوت به این دلیل است که StarGAN-JNT ترجمه تصاویر CelebA را در طول آموزش می آموزد اما StarGAN-SNG را نمی‌آموزد. به عبارت دیگر، StarGAN-JNT می‌تواند از هر دو مجموعه داده برای بهبود وظایف مشترک سطح پایین مانند تشخیص و تقسیم‌بندی نقاط کلید چهره استفاده کند.

بنابراین در این مقاله StarGAN را پیشنهاد می‌کنیم که یک مدل ترجمه image-to-image مقیاس‌پذیر در حوزه‌های مختلف است. تصاویر تولید شده در StarGAN دارای کیفیت بصری بالاتری هستند.

Problem 3

در این قسمت بعد از import کردن کتابخانه ها ، دیتای (mnist) را دانلود کرده و پس از reshape کردن عکس ها و تبدیل مقادیر پیکسلش به ۰ و ۱ به ادامه کار میپردازیم.

```
import glob
import imageio
import matplotlib.pyplot as plt
import numpy as np
import os
import PIL
from tensorflow.keras import layers
import time

from IPython import display
```

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

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

```
np.shape(train_images[1]), np.shape(train_labels[1])
```

در این قسمت دیتا را شافل کرده و همچنین پارامتر های train (batch size , buffersize) را تعیین می کنیم .

```
BUFFER_SIZE = 60000
BATCH_SIZE = 256
```

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

```python
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)
```

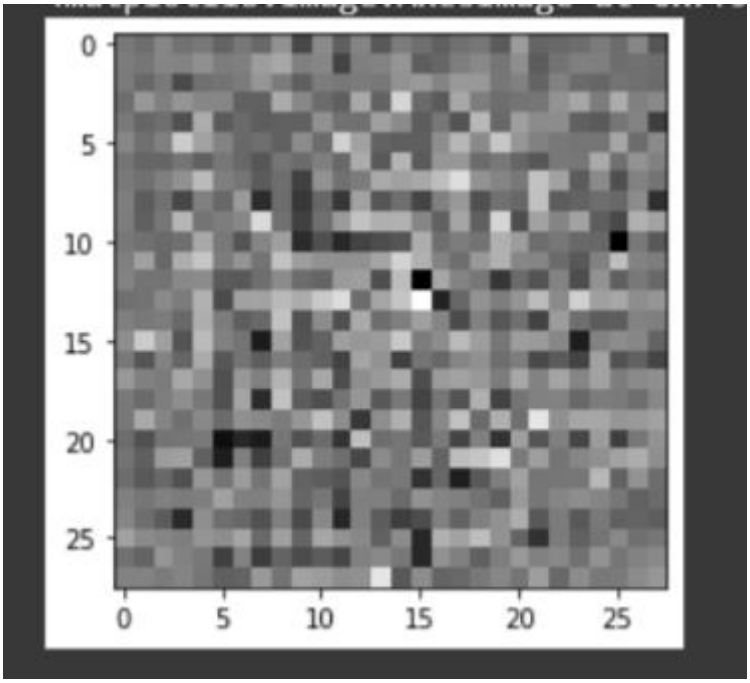
تابع generator از لایه‌های tf.keras.layers.Conv2DTranspos (upsampling) برای تولید تصویر از یکنویز تصادفی استفاده می‌کند. با یک لایه dense شروع می کند که این نویز را به عنوان ورودی می گیرد، سپس چندین بار نمونه برداری می کند تا به اندازه تصویر دلخواه ۲۸*۲۸*۱ برسد. فعال ساز هر لایه با tf.keras.layers.LeakyReLU است به جز لایه خروجی که از tanh استفاده می کند.

```python
generator = make_generator_model()

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

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

در این مرحله از تابع generator استفاده می کنیم و با پاس دادن یک نویز ، تصویری را می سازیم



تابع discriminator یک تابع cnn باری کلاس بندی تصاویر است که معماری آن به شکل زیر است.

```
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
```

از این تابع برای طبقه بندی تصویر ساخته شده در مرحله قبل (real or fake) استفاده میکنیم. چنانچه تصویر ساخته شده realاشد ، مقدار برگردانده شده مثبت است و در غیر این صورت مقداری منفی را بر میگرداند(برای تصویر fake)

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

در ادامه تابع loss اصلی را تعریف میکنیم که از نوع binarycrossentropy می باشد .

```
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
```

این تابع (discriminator loss) نشان میدهد که تمایزکننده چقدر می‌تواند تصاویر واقعی را از تقلبی تشخیص دهد. پیش‌بینی‌های discriminator روی تصاویر واقعی را با آرایه‌ای از ۱ها و پیش‌بینی‌های discriminator در تصاویر جعلی (تولید شده) را با آرایه‌ای از ۰ ها مقایسه می‌کند.

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

این تابعloss نشان می دهد که بخش generator چقدر توانسته است discriminator را فریب دهد. بطور شهودی، اگر generator به خوبی عمل کند، discriminator تصاویر جعلی را به عنوان واقعی (یا ۱) طبقه بندی می کند.

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

Optimizer مربوط به generator و discriminator جدا گانه تنظیم شده اند چرا که میبایست جداگانه train شوند .

```
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)
```

در این بخش تابع مربوط به ذخیره سازی چک پوینت ها برای استفاده در صورتی که در training طولانی به مشکل بخوریم پرداخته شده است

```
EPOCHS = 10
noise_dim = 100
num_examples_to_generate = 16
seed = tf.random.normal([num_examples_to_generate, noise_dim])
```

```
@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))
```

این قسمت مربوط به پارامتر های training loop می باشد .

در واقع یک ورودی نویز را به تابع generator میدهیم تا تصویری بسازد و سپس تصویر ساخته شده را با کمکdiscriminator با تصویر دادگان مقایسه کرده و پس از به دست آوردن loss شروع به آپدیت کردن دو تابع generator و discriminator می کنیم .

```
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)
```

در تابع train هم به تعداد bach size عکس های دادگان را جدا و با تابع train_step که در قسمت قبل توضیح دادیم شبکه ها را train می کنیم و در هر پانزده ایپاک یک بار چک پوینت ها و اطلاعات را ذخیره می کنیم .

```
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 انتها نیز از تابع generate and save برای prediction و نشان دادن و ذخیره کردن نتایج به دست آمده استفاده می کنیم .

منبع: Deep Convolutional Generative Adversarial Network | TensorFlow Core

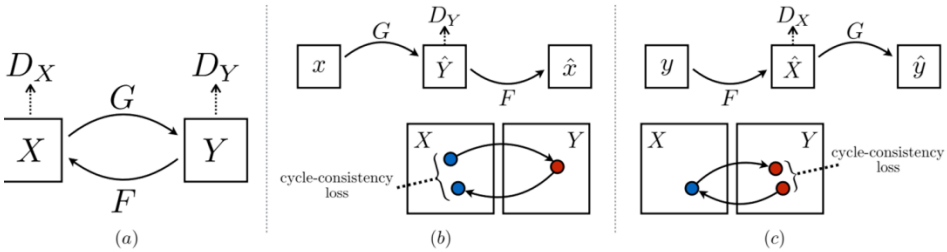
Problem 4

CycleGAN

$$\mathcal{L}_{\text{GAN}}(G, D_Y, X, Y) = \mathbb{E}_{y \sim p_{\text{data}}(y)}[\log D_Y(y)] + \mathbb{E}_{x \sim p_{\text{data}}(x)}[\log(1 - D_Y(G(x))]$$

$$\mathcal{L}_{\text{cyc}}(G, F) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\|F(G(x)) - x\|_1] + \mathbb{E}_{y \sim p_{\text{data}}(y)}[\|G(F(y)) - y\|_1]$$

$$\mathcal{L}(G, F, D_X, D_Y) = \mathcal{L}_{\text{GAN}}(G, D_Y, X, Y) + \mathcal{L}_{\text{GAN}}(F, D_X, Y, X) + \lambda \mathcal{L}_{\text{cyc}}(G, F)$$



در واقع در cycle –consistency loss در نظر میگیریم که نتیجه نهایی چقدر به ورودی نزدیک است (به طور مثال یک جمله را از انگلیسی به فرانسه ترجمه می کنیم و سپس از فرانسه به انگلیسی ترجمه می کنیم ، میزان تفاوت نتیجه حاصل با جمله اولیه همان cycle consistency loss است .

- Image X is passed via generator G that yields generated image $\hat{Y}$.
- Generated image $\hat{Y}$ is passed via generator F that yields cycled image $\hat{X}$.
- Mean absolute error is calculated between X and $\hat{X}$.

$$forward\ cycle\ consistency\ loss : X -> G(X) -> F(G(X)) \sim \hat{X}$$

$$backward\ cycle\ consistency\ loss : Y -> F(Y) -> G(F(Y)) \sim \hat{Y}$$

منبع: https://www.tensorflow.org/tutorials/generative/cyclegan

Problem 5

GAN از مولد و متمایزکننده تشکیل شده است ماموریت مولد، تولید داده‌های جعلی و متمایزکننده به دنبال طبقه‌بندی داده‌های واقعی از داده‌های جعلی است.

در طول آموزش، هر یک از این مدل‌ها در کار خود مهارت بیشتری پیدا می‌کنند و حالت توقف زمانی است که در آن هیچ‌یک از مدل‌ها بر دیگری غلبه نمی‌کند. مولد ویژگی‌های داده‌های آموزشی را برای تولید تصاویر واقعی می‌آموزد. متمایزکننده در حال آموزش خود بر روی تصاویر جعلی برچسب‌دار و تصاویر واقعی است تا آن‌ها را طبقه‌بندی کند.

علی‌رغم قدرتمندی زیاد GANها، آن‌ها بسیار ناپایدار نیز هستند. چندین سناریو خرابی وجود دارد که ممکن است یک GAN خود را در آن‌ها گیر کرده باشد. دو نوع رایج خرابی عبارتند از convergence failure (عدم تولید خروجی های با کیفیت خوب) و mode collapse (ناتوانی در تولید خروجی های مختلف با ظاهر متفاوت). در قسمت الف مورد دوم و در قسمت ب مورد اول بررسی می‌شود.

- **الف) Mode Collapse**

یک GAN خوب آموزش دیده می‌تواند خروجی‌های متنوعی تولید کند. هنگام تولید تصاویری از صورت انسان، می‌خواهید که ژنراتور دسته‌هایی از چهره‌های متفاوت با ویژگی‌های متفاوت ایجاد کند. mode collapse زمانی اتفاق می‌افتد که ژنراتور تنها بتواند یک نوع خروجی یا مجموعه کوچکی از خروجی‌ها را تولید کند. این ممکن است به دلیل مشکلاتی در آموزش اتفاق بیفتد، مثلاً مولد یک نوع داده را پیدا می‌کند که به راحتی قادر است متمایزکننده را فریب دهد و بنابراین به تولید آن نوع ادامه می‌دهد. از آنجایی که هیچ انگیزه‌ای برای ژنراتور وجود ندارد که همه چیز را تغییر دهد، کل سیستم در آن یک خروجی بیش از حد بهینه می‌شود. هیچ راه درستی برای اندازه گیری فروپاشی مدل وجود ندارد. اقدامات کیفی مانند نگاه دستی به تصاویر تنها در صورتی کار می‌کند که بدیهی باشد و ممکن است برای موارد پیچیده‌تر یا مقادیر انبوه داده شکست بخورد. سایر معیارهای کمی، مانند Inception Score (IS) یا Frechet Inception Distance (FID) بر مدل‌های از پیش آموزش دیده با مجموعه خاصی از کلاس‌های شی تکیه دارند.
در شکل‌های زیر نمونه‌ای Mode Collapse نشان داده شده است.

- **ب) Convergence failure**

یکی از رایج‌ترین شکست‌ها در هنگام آموزش GAN، Convergence failure است. یک شبکه عصبی زمانی که مسئله‌ی از دست دادن مدل در طول فرایند آموزش در آن حل نمی‌شود، همگرا نمی‌شود. در مورد GAN نیز این عدم همگرایی که عدم تعادل بین تمایزکننده و مولد اشاره دارد.

راه محتمل برای شناسایی این نوع شکست این است که ضرر برای متمایزکننده به صفر یا نزدیک به صفر رسیده باشد. در برخی موارد، تلفات ژنراتور نیز ممکن است افزایش یابد و در همان دوره به افزایش ادامه دهد. این نوع شکست ممکن است در ابتدای روند اتفاق بیفتد و در طول روند ادامه یابد، در این مرحله باید روند را متوقف کنید. برای برخی از GANهای ناپایدار، ممکن است GAN برای تعدادی از به‌روزرسانی‌های دسته‌ای یا حتی تعدادی از دوره‌ها در این حالت شکست قرار بگیرد و سپس بازیابی شود.

راه‌های زیادی برای آسیب رساندن به GAN پایدار برای دستیابی به شکست هم‌گرایی وجود دارد، مانند تغییر یک یا هر دو مدل برای داشتن ظرفیت ناکافی، تغییر الگوریتم بهینه‌سازی Adam به بیش از حد تهاجمی، و استفاده از اندازه‌های هسته بسیار بزرگ یا بسیار کوچک در مدل‌ها.

منبع:

https://wandb.ai/authors/DCGAN-ndb-test/reports/Measuring-Mode-Collapse-in-GANs--VmlldzoxNzg5MDk

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