به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

درس شبکه‌های عصبی و یادگیری عمیق

تمرین ششم

نام و نام خانوادگی	فربد سیاهکلی – سعید شکوفا
شماره دانشجویی	810198418 - 810198510
تاریخ ارسال گزارش	1401.11.07

فهرست

شکل‌ها

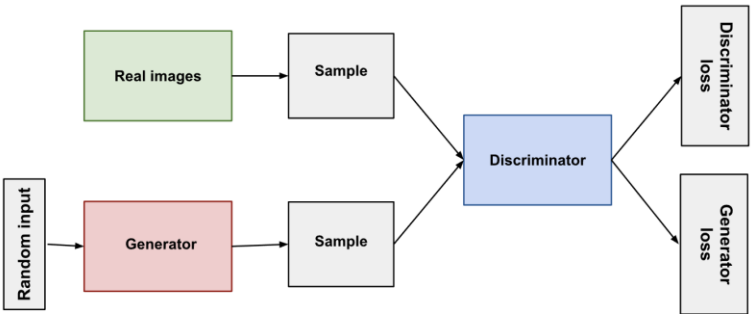
جدول‌ها

No table of figures entries found.

پاسخ 1. شبکه‌های مولد متخاصمی کانولوشنال عمیق

1- پیاده سازی مولد تصویر با استفاده از شبکه های مولد متخاصمی کانولوشنال عمیق

شماتیک کلی شبکه GAN:



شکل 1 شماتیک کلی شبکه **GAN** پیاده‌سازی شده

پیاده سازی: در این بخش دو شبکه Generator و Discriminator را بر اساس مقاله مرجع پیاده سازی کرده و سپس اقدام به آموزش شبکه می‌کنیم. معماری شبکه مولد:

```
Sequential(
  (0): Sequential(
    (0): ConvTranspose2d(100, 256, kernel_size=(3, 3), stride=(2, 2))
    (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
  )
  (1): Sequential(
    (0): ConvTranspose2d(256, 128, kernel_size=(3, 3), stride=(2, 2))
    (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
  )
  (2): Sequential(
    (0): ConvTranspose2d(128, 64, kernel_size=(3, 3), stride=(2, 2))
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
  )
  (3): Sequential(
    (0): ConvTranspose2d(64, 1, kernel_size=(4, 4), stride=(2, 2))
    (1): Tanh()
  )
)
```

شکل 2 معماری شبکه مولد

شبکه Discriminator:

```
Sequential(
  (0): Sequential(
    (0): Conv2d(1, 16, kernel_size=(4, 4), stride=(2, 2))
    (1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): LeakyReLU(negative_slope=0.2, inplace=True)
  )
  (1): Sequential(
    (0): Conv2d(16, 32, kernel_size=(4, 4), stride=(2, 2))
    (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): LeakyReLU(negative_slope=0.2, inplace=True)
  )
  (2): Sequential(
    (0): Conv2d(32, 1, kernel_size=(4, 4), stride=(3, 3))
  )
)
```

شکل 3 معماری شبکه **Discriminator**

برای محاسبه خطا ابتدا تصاویر واقعی را به شبکه داده و لیبل متناظر با آن را یک قرار می‌دهیم. سپس نویز تولید شده را به شبکه مولد داده و خروجی آن که تصاویر فیک هستند را به شبکه Discriminator داده و لیبل متناظر با آن را صفر می‌گذاریم. نهایتا از روی این خطاها شبکه‌ها را بهینه سازی می‌کنیم.

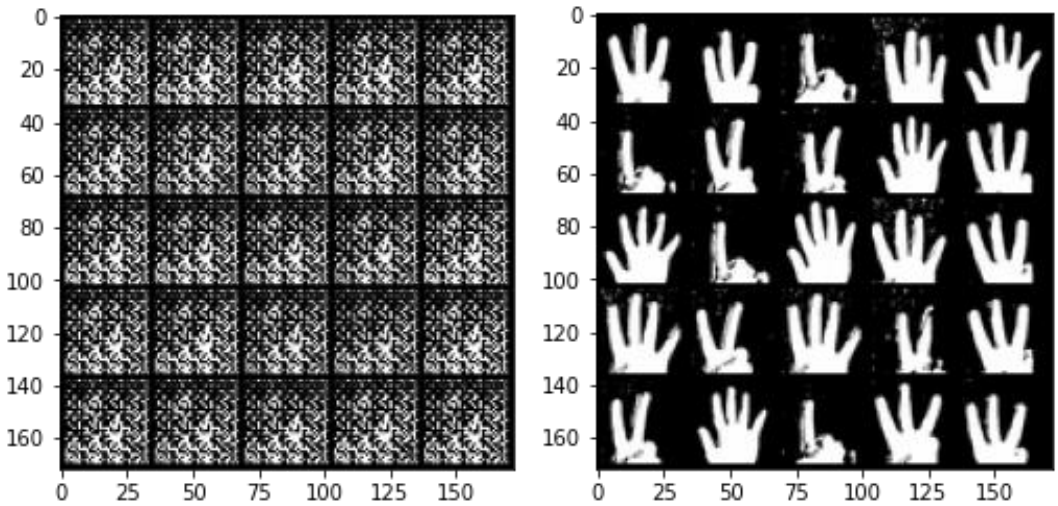
نکته قابل توجه آن است که شبکه مولد با توجه خطایی که Discriminator برای خروجی آن در نظر گرفته آموزش می‌گردد ولی شبکه Discriminator با ترکیب دو خطا آموزش می‌بیند. بدین منظور از تابع خطای BCE استفاده شده است تا خطای دسته بندی تصاویر توسط Discriminator بدست آید.

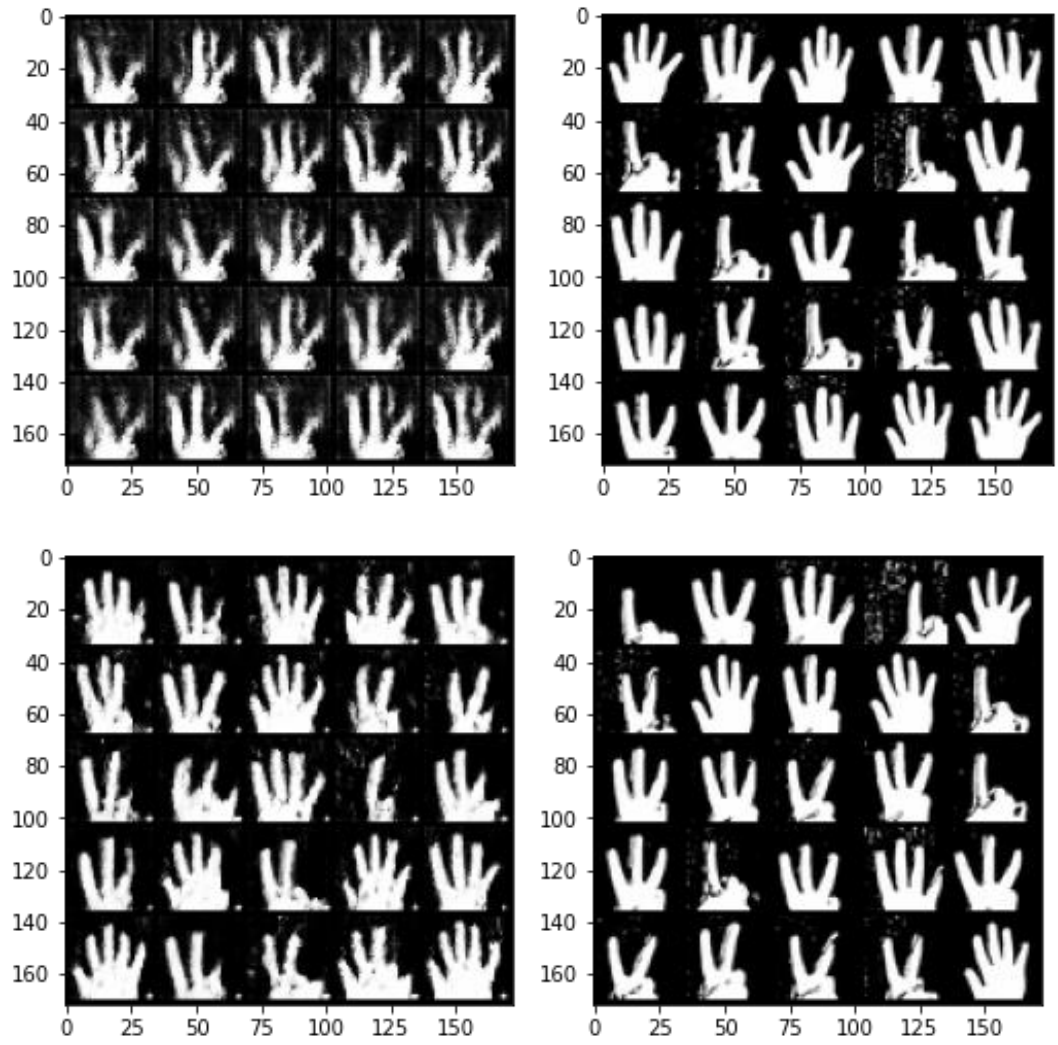
نهایتا از بهینه ساز Adam با learning rate=0.001 برای هر دو شبکه استفاده شده است.

```
criterion = nn.BCEWithLogitsLoss()
z_dim = 100
display_step = 100
batch_size = 128
lr = 0.0011
beta_1 = 0.5
beta_2 = 0.999

device = 'cuda' if torch.cuda.is_available() else 'cpu'

gen = Generator(z_dim,1,64).to(device)
gen_opt = torch.optim.Adam(gen.parameters(), lr=lr, betas=(beta_1, beta_2))

disc = Discriminator(1,16).to(device)
disc_opt = torch.optim.Adam(disc.parameters(), lr=lr, betas=(beta_1, beta_2))
```

شکل 4 تعریف بهینه ساز و تابع خطا

نتایج: خروجی شبکه مولد به ازای نویز ورودی:



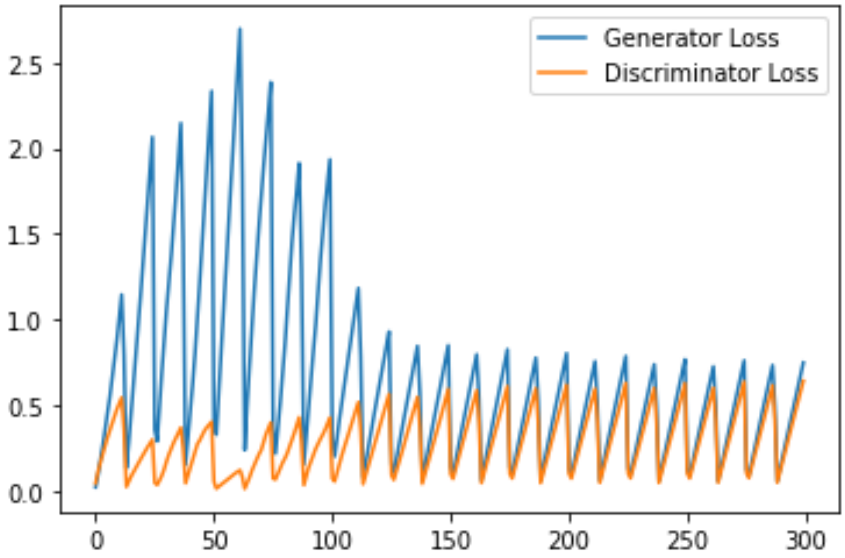


شکل 5 خروجی مولد در هنگام آموزش و ورود مولد به فضای دیتاست

نتایج خطا برای شبکه‌ها در ایپاک نهایی:

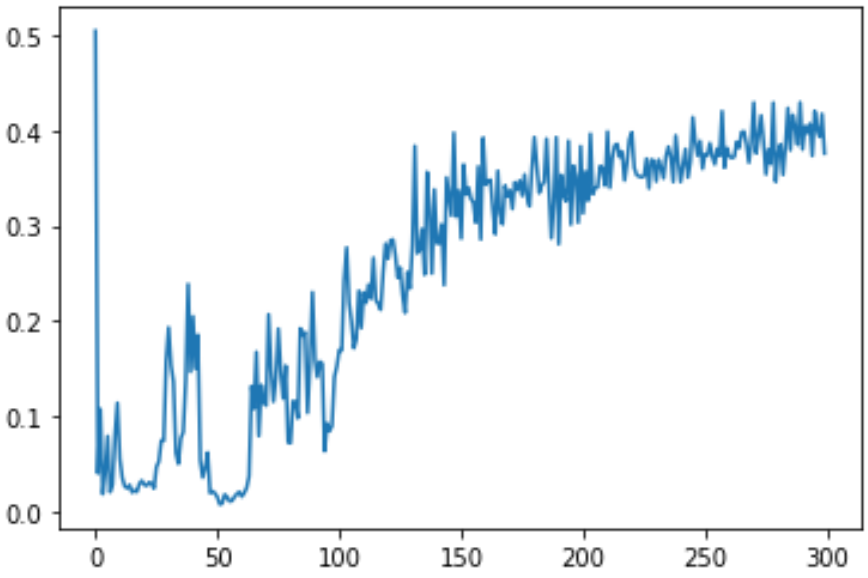
```
Step 2300:
Generator loss: 0.8009951579570767
Discriminator loss: 0.6750366014242172
```

نمودارهای خطا برای دو شبکه Gen و Discriminator:



شکل 6 نمودار خطا برای شبکه **DCGAN**

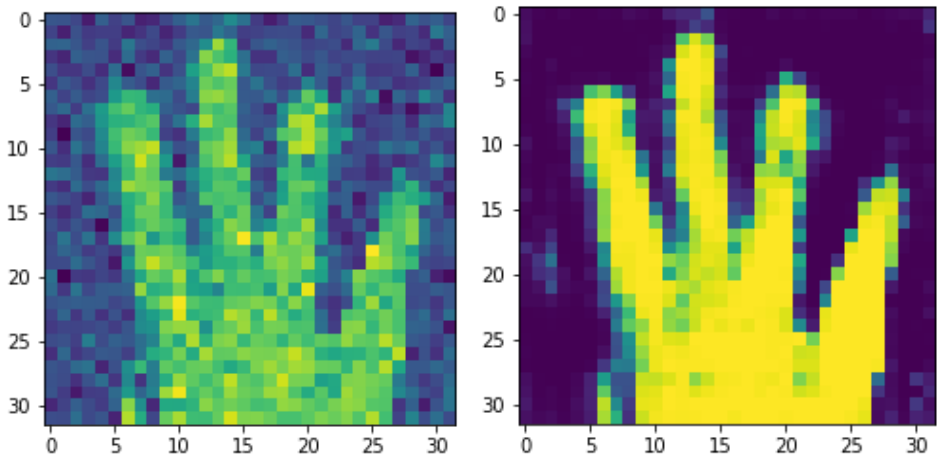
نمودار دقت شبکه Discriminator برای تشخیص تصاویر واقعی از فیک:



شکل 7 نمودار دقت برای **Discriminator** در **DCGAN**

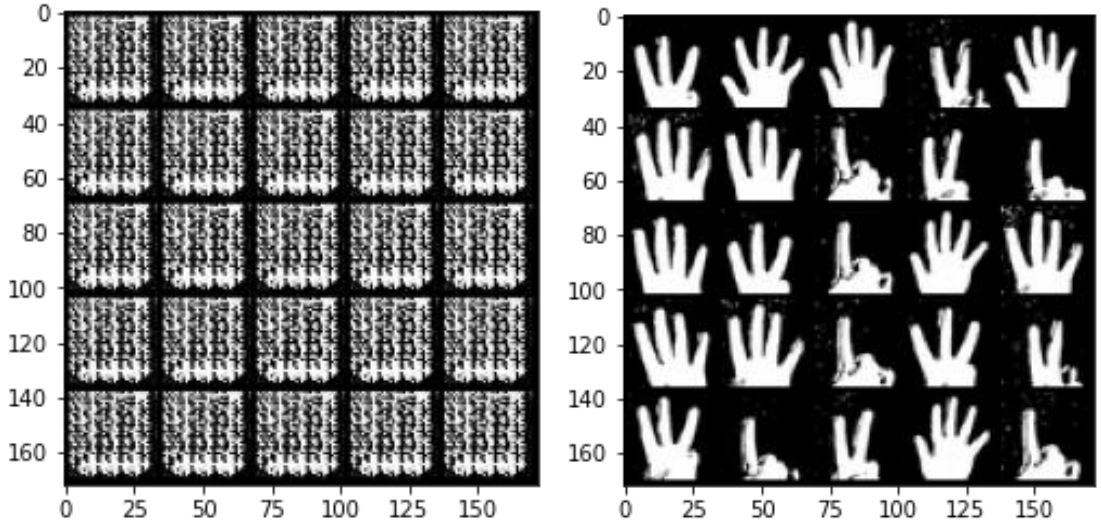
2- پایدارسازی شبکه

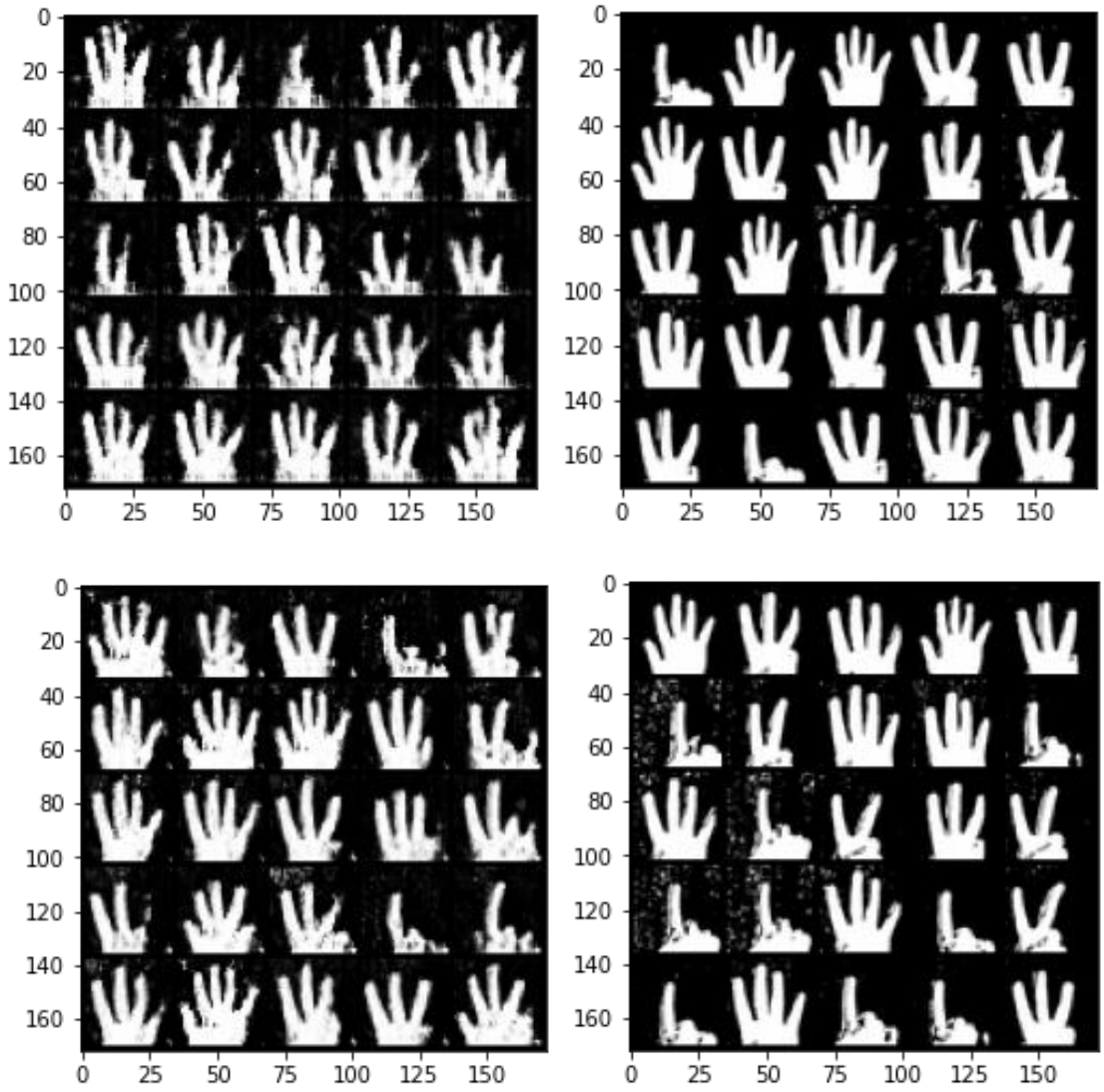
حال دو متود **One Sided Smoothing** و **Noise Addition** را پیاده می‌کنیم. در متود Noise Addition به تصاویر فیک یک نویز رندوم را اضافه می‌کنیم. تصاویر به صورت زیر خواهند شد.



شکل 8 اضافه کردن نویز به تصاویر مولد

حال برای متود One Sided Smoothing به جای لیبل 1 برای تصاویر واقعی، از لیبل 0.9 استفاده می‌کنیم. این متودها به Stability شبکه به هنگام آموزش کمک می‌کنند. نتایج آموزش پس از استفاده از این متودها به شرح زیر است:





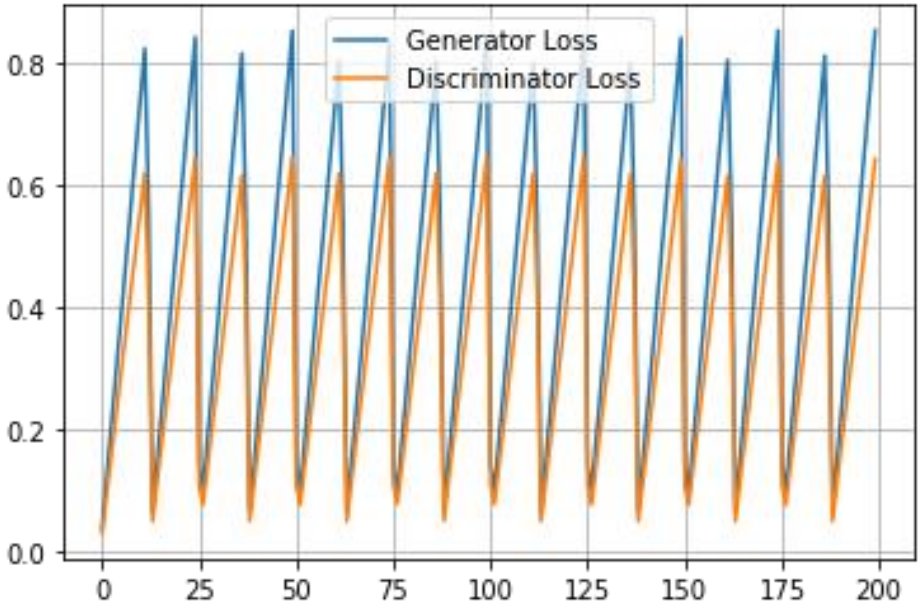
شکل 9 تصاویر خروجی مولد پس از اضافه کردن دو متود مطرح شده

نتایج خطای دو شبکه در ایپاک آخر:

```
Step 1500:
Generator loss: 0.8880451267957685
Discriminator loss: 0.6731136178970337
```

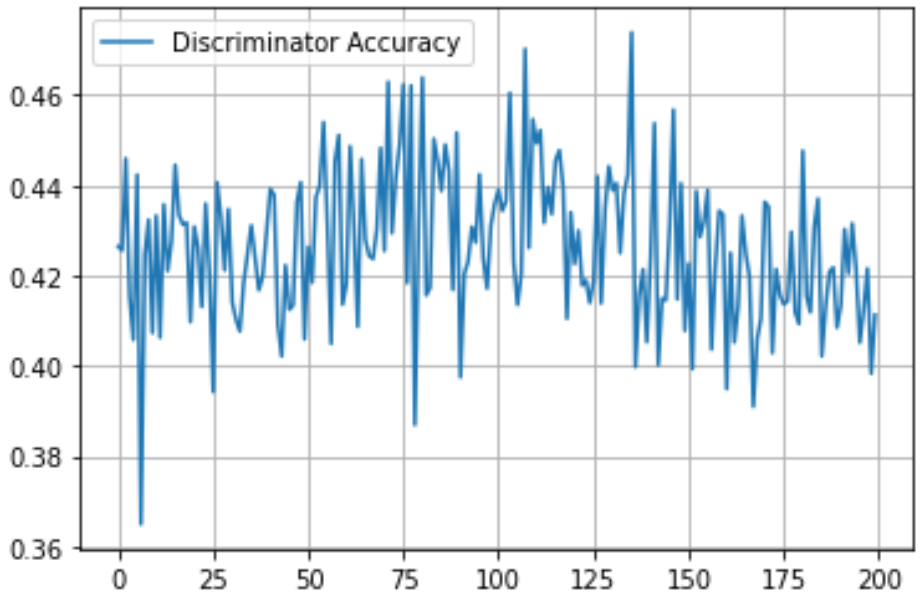
3- ارزیابی شبکه

نمودار Loss برای دو شبکه:



شکل 10 نمودار خطای دو شبکه برای ایپاک‌ها

نمودار Accuracy برای شبکه Discriminator:



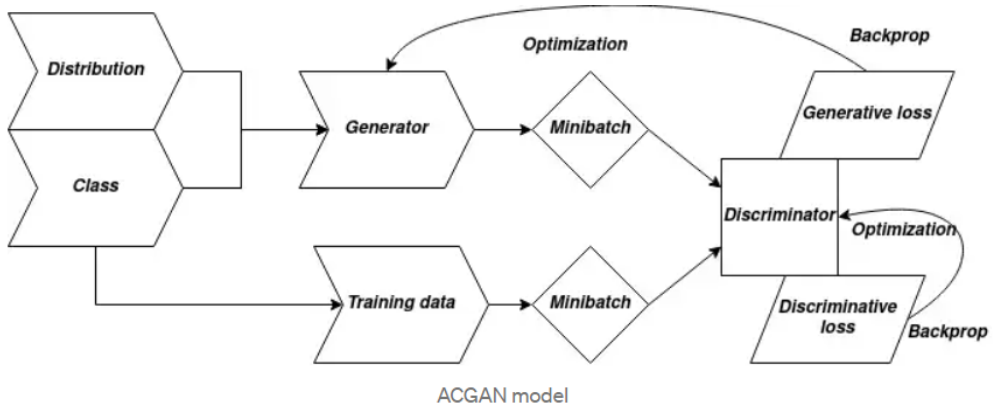
شکل 11 نمودار دقت برای شبکه **Discriminator**

پاسخ ۲ – شبکه‌های متخاصم مولد طبقه‌بند کمکی و Wasserstein

1- شبکه متخاصم مولد طبقه بند کمکی:

در این شبکه generator مانند شبکه GAN می باشد(فقط شماره کلاس را هم برای تشخیص طبقه بندی در Discriminator به آن می دهیم) و Discriminator دچار تفاوت شده است به این صورت که Discriminator علاوه بر تشخیصfake بودن تصویر باید کلاس تصویر را نیز تشخیص دهد و در نتیجه دو خروجی می دهد یکی برای تشخیص fake بودن و دیگری برای نوع طبقه ای که عکس در آن قرار دارد و backward بر روی میانگین این دو تابع خطا زده می شود که سبب می شود ترین دقیق تری انجام شود چرا که Discriminator قوی تر شده است و سبب می شود generator نیز قوی تر شود.

مدل شبکه:



شکل 12 دیاگرامی از شبکه مورد بررسی **ACGAN**

پیاده سازی: AC-GAN (Auxiliary Classifier GAN) گونه ای از GAN (شبکه متخاصم مولد) است که یک طبقه بندی کمکی را به ژنراتور معرفی می کند. این به تولید تصاویری کمک می کند که نه تنها واقع گرایانه هستند، بلکه به یک کلاس خاص نیز تعلق دارند. طبقه بندی کننده کمکی اطلاعات اضافی در مورد برچسب کلاس در اختیار جنراتور قرار می دهد که می تواند برای هدایت فرآیند تولید و تولید نمونه های متنوع و دقیق تر مورد استفاده قرار گیرد.

مدل Discriminator:

```
Discriminator(
  (conv_layer1): Conv2d(3, 16, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
  (conv_layer2): Sequential(
    (0): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
  (conv_layer3): Sequential(
    (0): Conv2d(32, 64, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
  (conv_layer4): Sequential(
    (0): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
  (conv_layer5): Sequential(
    (0): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
    (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
  (conv_layer6): Sequential(
    (0): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
  (aux_fc_layer7): Linear(in_features=8192, out_features=5, bias=True)
  (dis_fc_layer7): Linear(in_features=8192, out_features=1, bias=True)
  (sigmoid): Sigmoid()
  (softmax): Softmax(dim=-1)
)
```

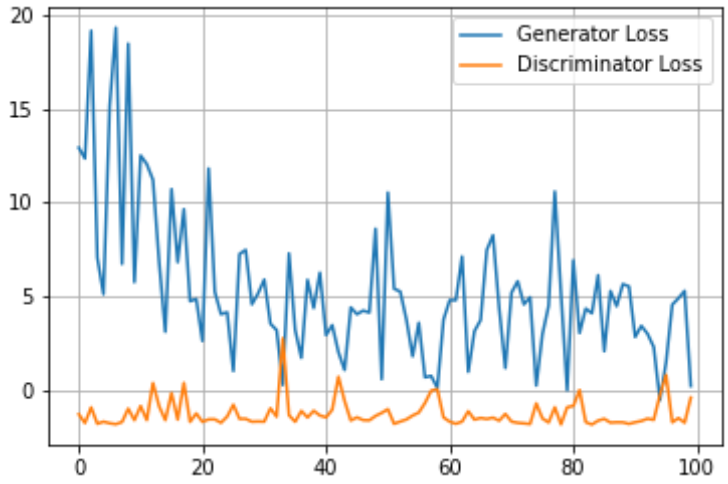
شکل 13 مدل **Discriminator** برای الگوریتم **ACGAN**

مدل Generator:

```
Generator(
  (fc_layer1): Linear(in_features=105, out_features=768, bias=True)
  (up_sample_layer2): Sequential(
    (0): ConvTranspose2d(768, 384, kernel_size=(4, 4), stride=(2, 2))
    (1): BatchNorm2d(384, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
  (up_sample_layer3): Sequential(
    (0): ConvTranspose2d(384, 192, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (1): BatchNorm2d(192, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
  (up_sample_layer4): Sequential(
    (0): ConvTranspose2d(192, 96, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (1): BatchNorm2d(96, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
  (up_sample_layer5): ConvTranspose2d(96, 3, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
  (tanh): Tanh()
)
```

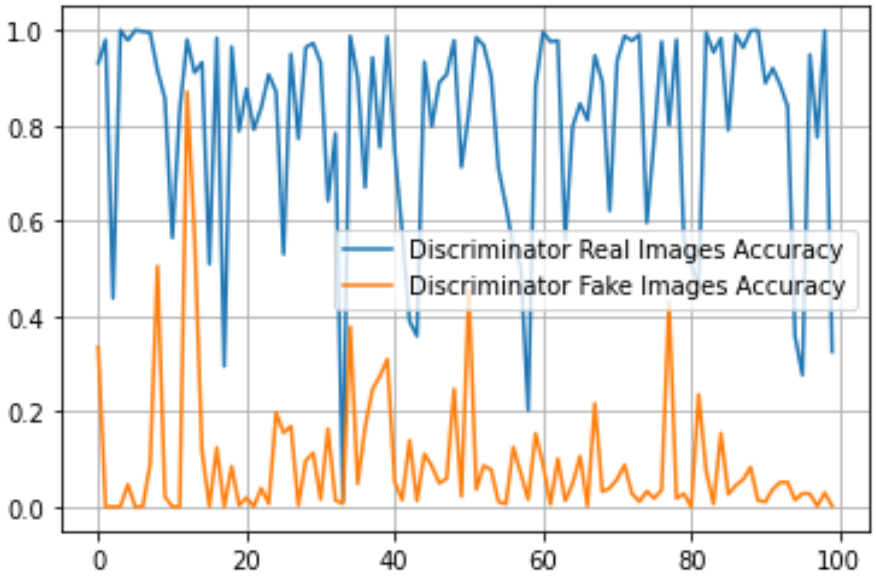
شکل 14 مدل **Discriminator** برای الگوریتم **ACGAN**

نتایج: نمودار خطای دو شبکه در معماری ACGAN:



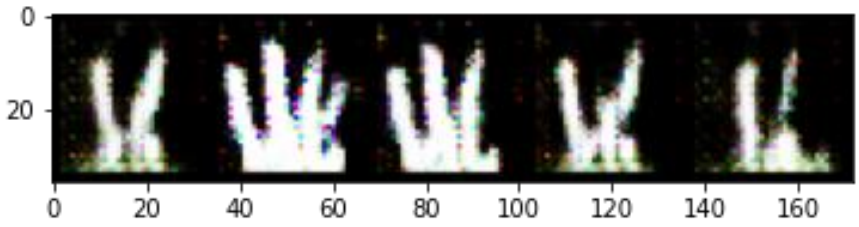
شکل 15 نمودار خطای دو شبکه در متود **ACGAN**

نمودار دقت شبکه Discriminator:



شکل 16 نمودار دقت شبکه **Discriminator** در متود **ACGAN**

خروجی شبکه برای 5 نویز متفاوت: مشاهده می‌شود که شبکه مولد خروجی‌های نسبتا واقع گرایانه‌ای از روی دیتاست بدست آورده است.



شکل 17 خروجی شبکه مولد پس از آموزش در متود **ACGAN**

همچنین خطای شبکه در ایپاک آخر به صورت زیر می باشد:

```
epoch 99: d_loss: -0.4262838363647461 , g_loss: 0.19416691362857819
```

2- شبکه متخاصم مولد Wasserstein

در این بخش، هدف ساخت یک Wasserstein GAN با Gradient Penalty است که برخی از مشکلات پایداری GAN هایی را که تا این لحظه استفاده کرده‌ایم را حل می کند. به طور خاص، از نوع خاصی از تابع loss به نام W-loss استفاده خواهیم کرد، تا از مجازات های گرادیان به جهت جلوگیری از فروپاشی state استفاده شود.

تفاوت Loss جدید: تابع loss جدید به این صورت محاسبه می شود:

- Critic Loss = [average critic score on real images] – [average critic score on fake images]
- Generator Loss = -[average critic score on fake images]

در این تابع علامت امتیاز اهمیتی ندارد و مادامی که امتیاز تصاویر واقعی کمتر از تصاویر تقلبی است این تابع عملیات جداسازی را انجام می دهد و واقعی بودن یا نبودن را تشخیص می دهد. در صورتی که در تابع لاس قبلی از BCE استفاده میکردیم و خروجی ما صفر و یکی بود که تفاوت آن ها را با یکدیگر مبنا قرار میدادیم.

پیاده سازی:

ابتدا مانند بخش قبل به تعیین مدل generator و discriminator پرداخته و سپس الگوریتم پنالتی گرادیان را پیاده سازی می‌کنیم. ما اینبار در الگوریتم WGAN-GP، دیگر از discriminator استفاده نمی کنیم که تصاویر جعلی و واقعی را به عنوان 0 و 1 طبقه بندی کند، بلکه از منتقدی استفاده می‌کنیم که تصاویر را با اعداد واقعی امتیاز می دهد.

محاسبه پنالتی گرادیان به دو بخش تقسیم می‌گردد.

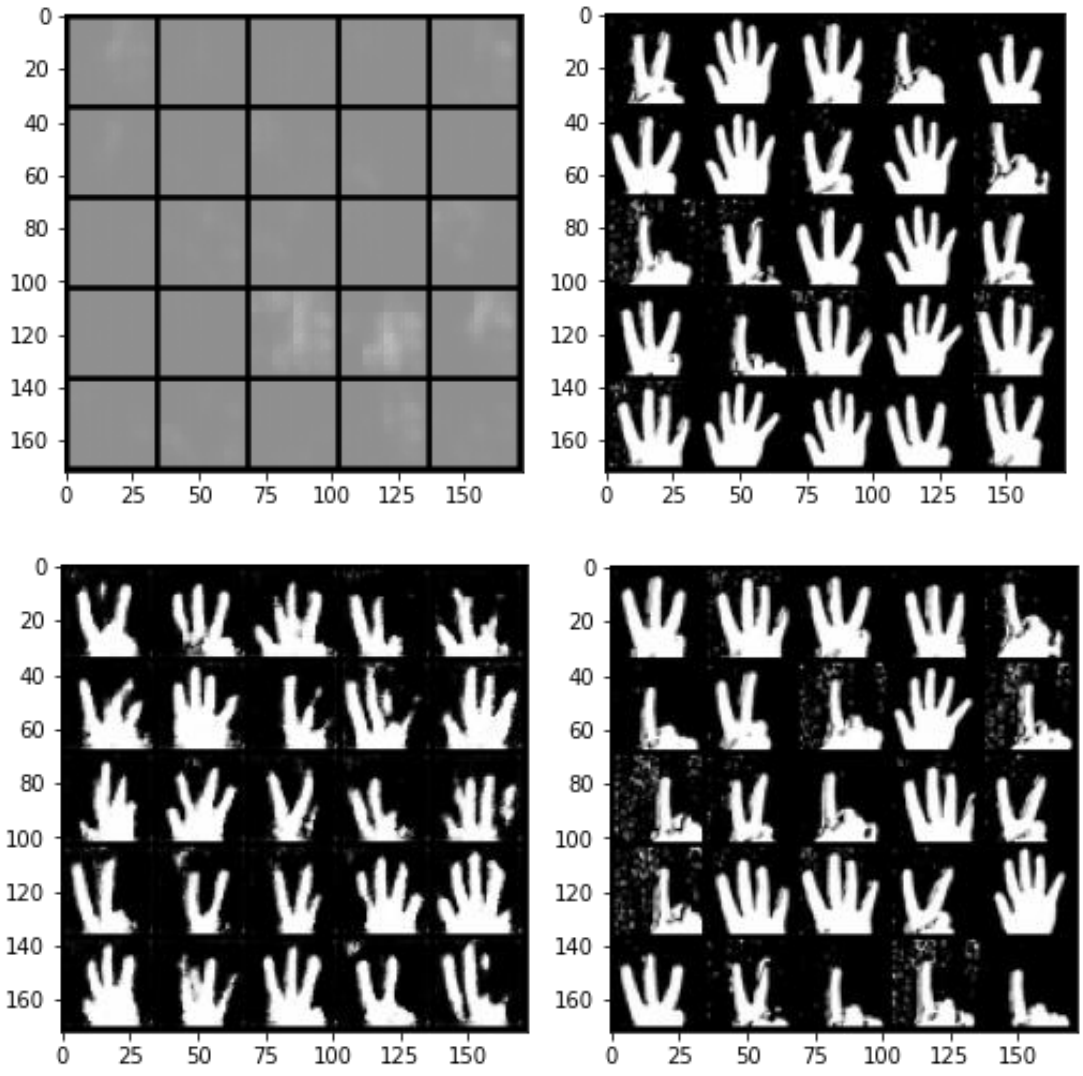
- محاسبه گرادیان با توجه به تصویر ورودی
- محاسبه پنالتی بر حسب گرادیان‌های شبکه

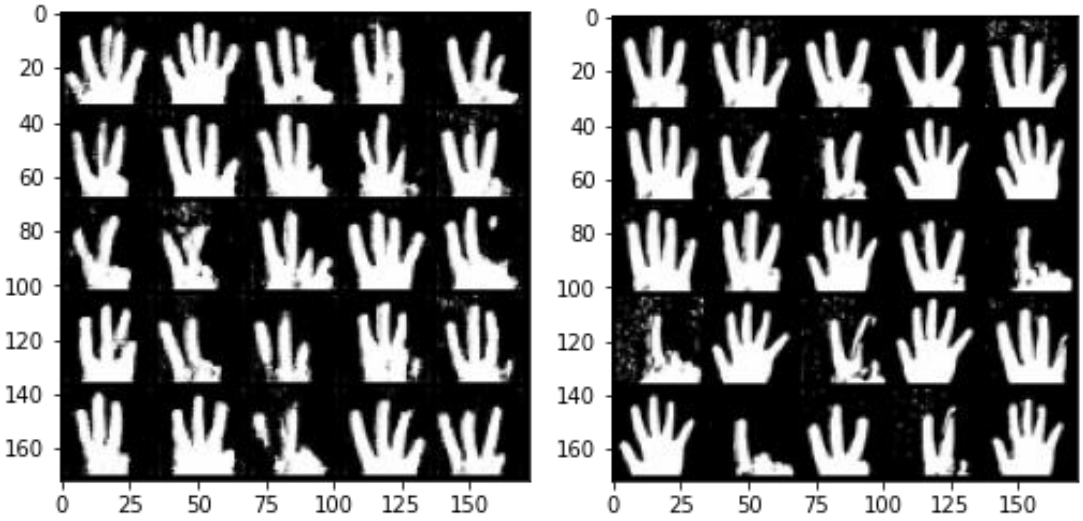
گرادیان ابتدا با ایجاد یک تصویر ترکیبی محاسبه می شود. این کار با جمع وزن دار تصویر جعلی و واقعی با استفاده از یک اپسیلون انجام می شود. هنگامی که تصویر تلفیقی ایجاد می‌گردد، می‌توان خروجی منتقد را برای تصویر دریافت کرد. در نهایت، گرادیان امتیاز منتقد را برای تصاویر ترکیبی (خروجی) با توجه به پیکسل های تصاویر ترکیبی (ورودی) محاسبه می‌شود. حال برای محاسبه پنالتی گرادیان ابتدا، بزرگی گرادیان هر تصویر را محاسبه کرده. به بزرگی یک گرادیان، norm نیز گفته می شود. سپس، جریمه

را با به توان رساندن فاصله بین هر بزرگی و norm ایده‌آل 1 و گرفتن میانگین توان دوی فاصله‌ها محاسبه می‌شود. برای ژنراتور، loss با به حداکثر رساندن پیش‌بینی منتقد بر روی تصاویر جعلی generator محاسبه می‌شود. این آرگومان برای همه تصاویر جعلی در batch یک امتیاز دارد، اما ما از میانگین آنها استفاده خواهیم کرد. برای منتقد، loss با به حداکثر رساندن فاصله بین پیش‌بینی‌های منتقد روی تصاویر واقعی و پیش‌بینی‌های روی تصاویر جعلی محاسبه می‌شود و در عین حال یک جریمه گرادیان نیز اضافه می‌شود. جریمه گرادیان بر اساس پارامتر λ سنجیده می شود. آرگومان ها همان نمرات همه تصاویر در batch هستند و ما از میانگین آنها استفاده خواهیم کرد.

نتایج:

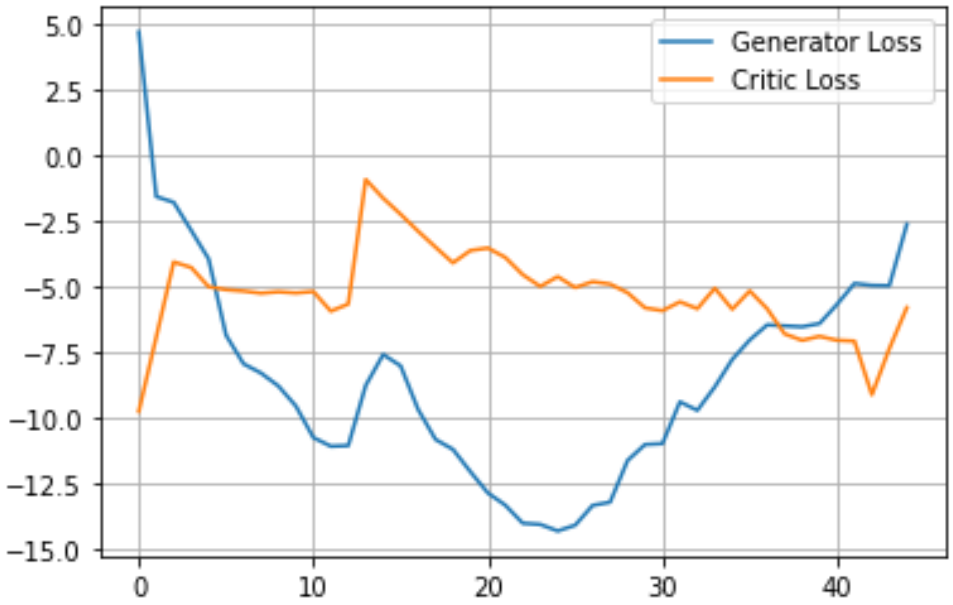
خروجی شبکه Generator به همراه Batch از عکس‌های واقعی به شرح زیر است:





شکل 18 تصویر شبکه مولد در کنار دیتاست

نمودار خطای دو شبکه Generator و Critic:

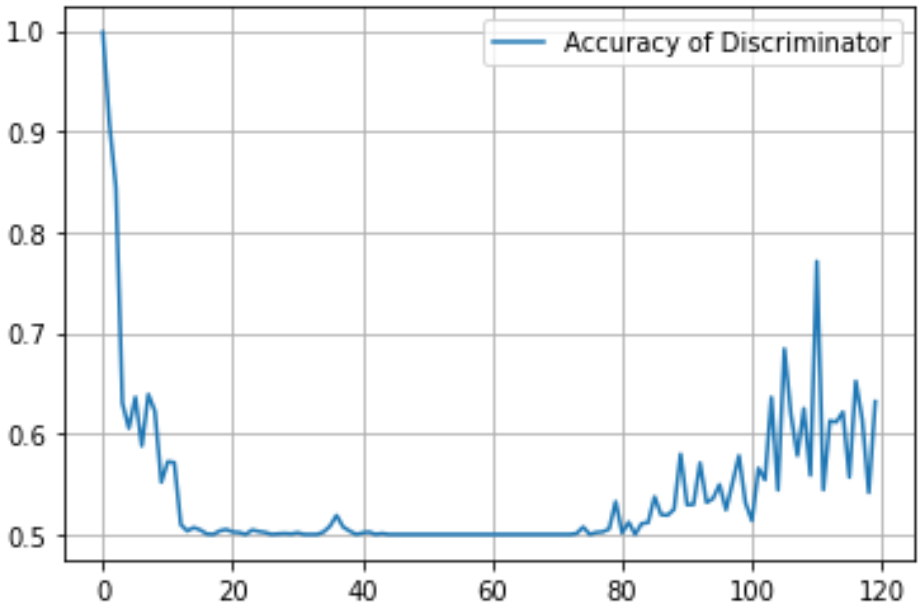


شکل 19 نمودار خطای دو شبکه

مقادیر مورد بررسی در ایپاک آخر:

```
Step 900:
Generator loss: -4.666675052791834
Critic loss: -7.284151717126374
Accuracy of critic 0.50390625
```

نمودار تغییرات Accuracy:



شکل 20 نمودار دقت برای شبکه **Discriminator**

مشاهده می‌شود که در هنگام آموزش، شبکه Generator با پیشرفت وزن‌های خود، از بالا رفتن دقت شبکه Discriminator جلوگیری کرده و نمودار آن نشان از کارکرد الگوریتم MinMax دارد.