به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

# درس شبکه‌های عصبی و یادگیری عمیق

## تمرین ششم

| | | |
|---|---|---|
| پرسش ۱ | نام و نام خانوادگی | حمیدرضا نادی مقدم |
| | شماره دانشجویی | ۸۱۰۱۰۳۲۶۴ |
| پرسش ۲ | نام و نام خانوادگی | علی صفری |
| | شماره دانشجویی | ۸۱۰۲۰۲۱۵۳ |
| | مهلت ارسال پاسخ | **۱۴۰۳.۱۰.۲۸** |

# فهرست

# شکل‌ها

# جدول‌ها

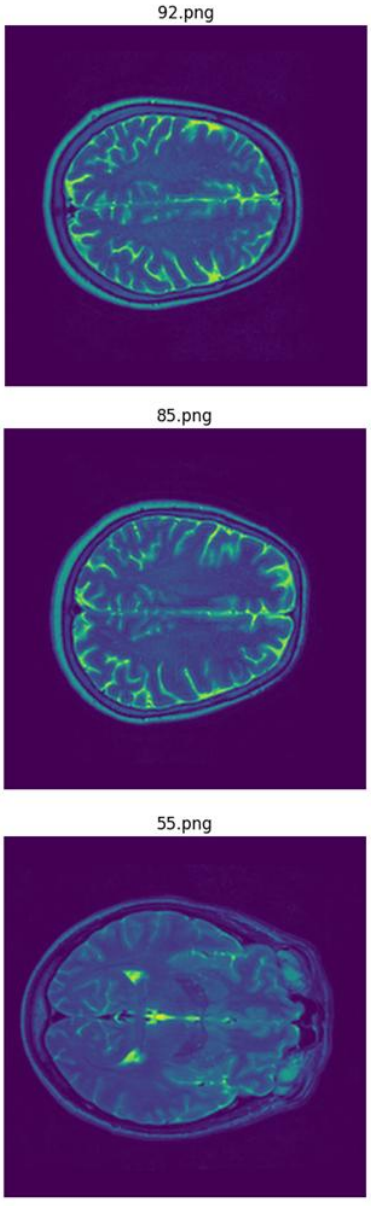
# مقدمه

برای پیاده‌سازی پروژه از بستر Google Colab به منظور کد نویسی و اجرا استفاده شده است. تمامی مراحل کد و اجرای آن در این گزارش به تفصیل شرح داده شده است.

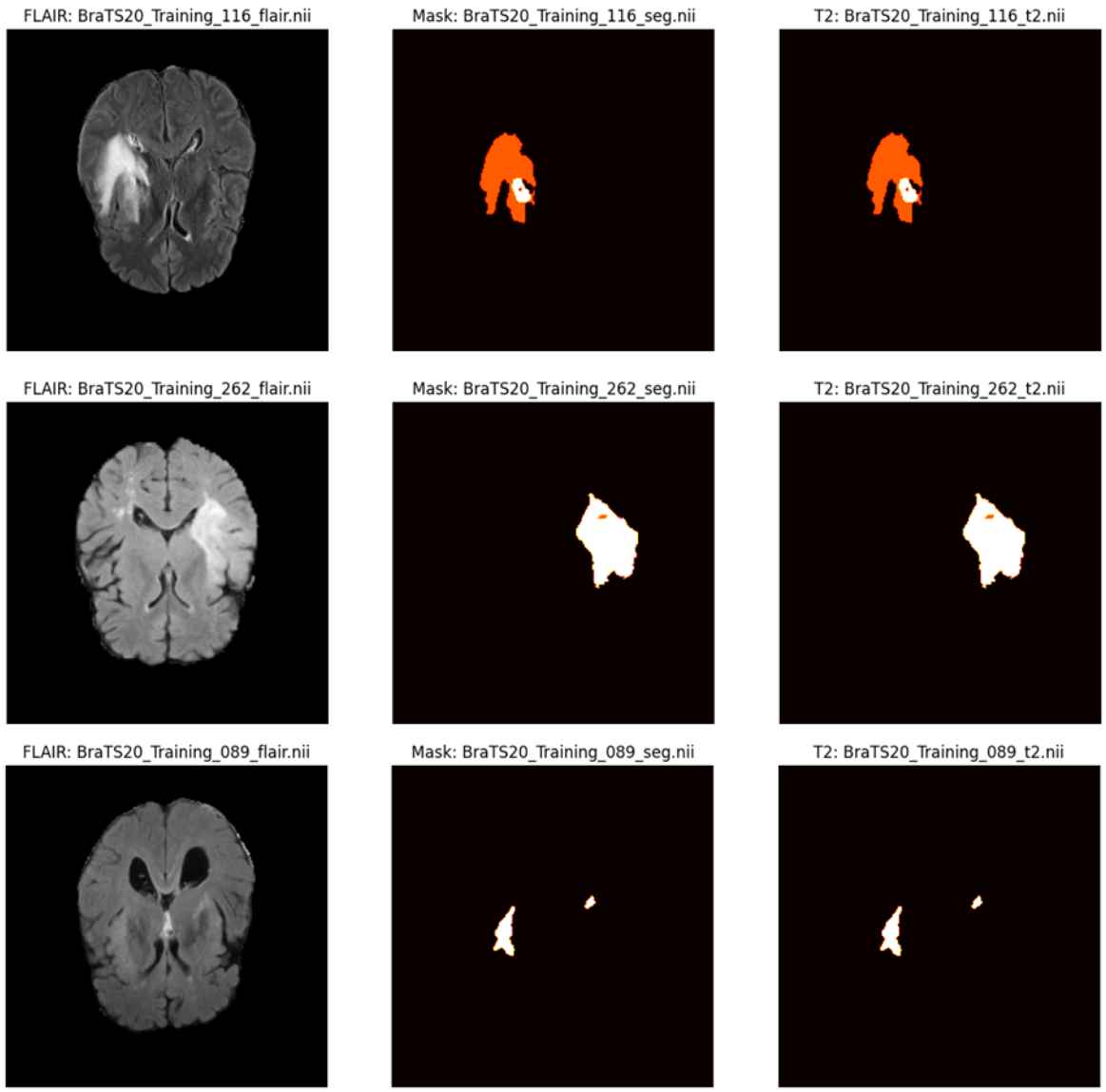
کد های نوشته شده همگی در پوشه‌ی Code و با پسوند ipynd. ذخیره شده است.

## پرسش ۱. طراحی و پیاده‌سازی Triplet VAE برای تشخیص تومور در MRI

ابتدا هر دو دیتاست را از Kaggle بارگذاری کردیم و سپس چند تصویر نمونه از دیتاست IXI و ماسک تومور BraTS به همراه تصویر سگمنت شده آن را نمایش داده‌ایم.




شکل ۱ نمونه تصاویر از دیتاست **IXI**




شکل ۲- نمونه تصاویر از دیتاست **BrsTS** با عکس اصلی و ماسک و **T2**

## ۲-۱. پیاده‌سازی یک VAE ساده

یک Variational Autoencoder (VAE) نوع خاصی از شبکه‌های عصبی مولد (Generative) است که برای مدل‌سازی داده‌های پیچیده استفاده می‌شود. هدف اصلی VAE یادگیری توزیع احتمال داده‌ها و تولید داده‌های جدید مشابه نمونه‌های اصلی است. در ادامه ایده کلی VAE توضیح داده شده است:

**قسمت Encoder و Decoder:**

Encoder: بخشی از VAE است که داده‌های ورودی (مثلاً تصاویر، متن یا داده‌های دیگر) را به یک فضای نهان (Latent Space) فشرده می‌کند. این فضای نهان به صورت معمولاً دو بردار میانگین و انحراف معیار توزیع نهان است.

Decoder: بخش دیگر VAE است که تلاش می‌کند از فضای نهان نمونه‌برداری کند و داده‌های ورودی اصلی را بازسازی کند. به عبارت دیگر، Decoder داده‌های جدیدی تولید می‌کند که از نظر آماری به داده‌های ورودی شباهت دارند.

**Latent Space (فضای نهان):**

فضای نهان جایی است که داده‌های ورودی به یک نمایش فشرده نگاشت می‌شوند. به جای اینکه داده‌ها را به نقاط خاصی در فضای نهان نگاشت کنیم (مانند Autoencoder های معمولی)، VAE داده‌ها را به یک توزیع احتمال (معمولاً یک توزیع گاوسی) نگاشت می‌کند. این ویژگی امکان نمونه‌برداری از فضای نهان و تولید داده‌های جدید را فراهم می‌کند.

**KL Divergence (واگرایی کولبک-لیبلر):**

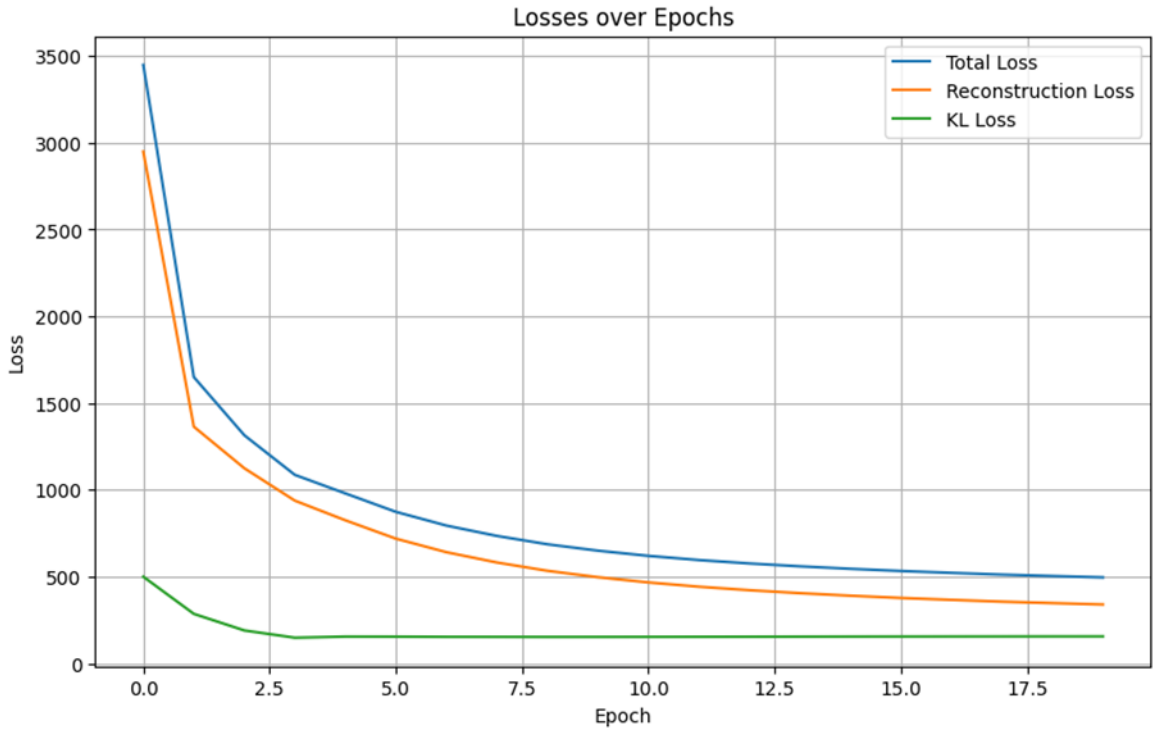
KL Divergence یک معیار برای اندازه‌گیری تفاوت بین دو توزیع احتمال است. در VAE از KL Divergence برای اطمینان از این استفاده می‌شود که توزیع نهان نزدیک به یک توزیع هدف باشد. تابع هزینه VAE از دو بخش تشکیل شده است: خطای بازسازی (Reconstruction Loss) که اندازه‌گیری تفاوت بین داده‌های اصلی و داده‌های بازسازی‌شده توسط Decoder است. و واگرایی KL که توزیع نهان را به توزیع گاوسی استاندارد نزدیک می‌کند.

خلاصه که در VAE داده‌های ورودی به کمک Encoder به یک توزیع احتمال نهان نگاشت می‌شوند. از این توزیع نمونه‌برداری شده و بردار نهان z تولید می‌شود. Z به کمک Decoder بازسازی می‌شود تا داده‌ای شبیه به x تولید شود. مدل با کمینه کردن ترکیبی از خطای بازسازی و KL Divergence آموزش می‌بیند.

در این مدل ما ابتدا تصاویر را برای ورودی به VAE آماده کردیم و انکدر و دیکدر را با ۴ لایه و ۳ لایه و مجموعه دو تابع هزینه KL Divergence و MSE Reconstruction Loss پیاده سازی کردیم. در این پیاده سازی ما ابعاد فضای نهان را برابر ۲۵۶ گرفته‌ایم و سایر پارامترها را نیز تنظیم کرده‌ایم.

**پیاده سازی:**

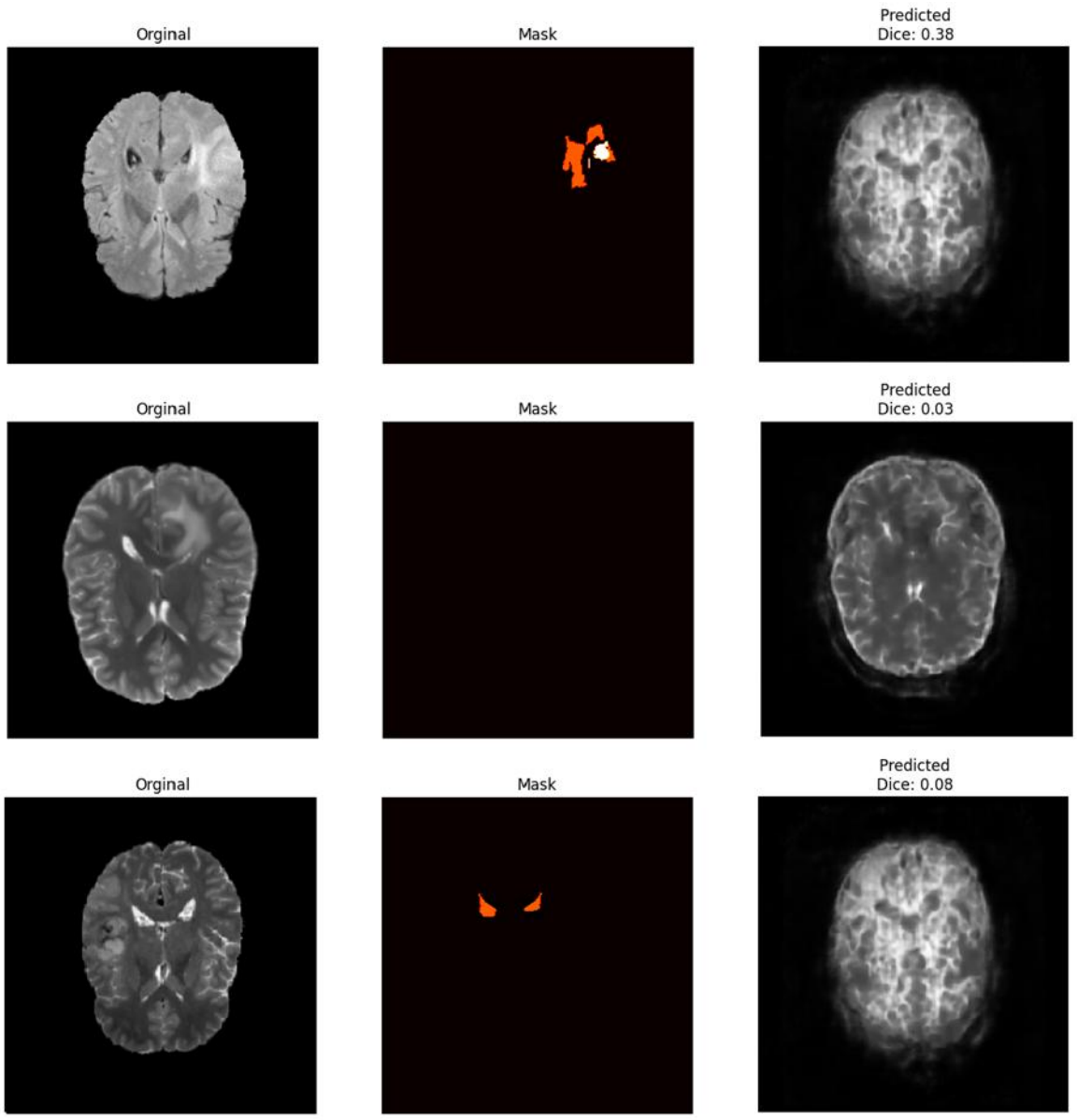
در پیاده سازی این مدل از شبکه عصبی VAE در قسمت Encoder که وظیفه استخراج ویژگی‌ها از ورودی (تصویر) و تولید دو بردار میانگین ($\mu$) و واریانس ($\sigma$) را دارد. این بخش از چند لایه کانولوشنی با فیلترهای افزایشی، نرمال‌سازی دسته‌ای (BatchNorm)، و تابع فعال‌ساز ReLU استفاده کرده‌ایم. و در قسمت Decoder که وظیفه بازسازی تصویر اولیه از بردار کدگذاری‌شده (بردار نهفته z) را دارد. این بخش نیز شامل لایه‌های کانولوشنی ترانهاده (Transposed Convolution) همراه با نرمال‌سازی دسته‌ای و فعال‌سازی ReLU است. خروجی نهایی از یک تابع tanh عبور کرده است. و در آخر VAE، دو بخش Encoder و Decoder را ترکیب می‌کند و با استفاده از روش نمونه‌گیری مجدد، بردار z را تولید می‌کند. و بعلاوه بر بازسازی، این مدل شامل یک تابع هزینه KL Divergence Loss بعلاوه MSE نیز می‌باشد. که هر سه به تفکیک در هر ایپاک چاپ داده شده‌اند و نمودار آن ترسیم شده است.




شکل ۳ - نمودار مقادیر **loss** برای هر سه معیار

**تست روی داده‌های BraTS:**

بعد از آموزش این مدل از VAE روی دیتاست سالم IXI با ۲۰ دوره و تست آن با چند اسلایس توموری از دیتاست BraTS که نمونه بازسازی را نمایش داده‌ایم. معیار Dice نیز پیاده‌سازی و برای هر تصویر نمایش داده شده است. تصاویر بازسازی با توجه به اینکه داده‌های تست را ندیده است قابل قبول است ولی تصویر ماسک را نمی تواند بازسازی کند. دلیل آن این است که روی داده‌های سالم آموزش دیده است و روی داده توموری تست انجام داده ایم که باعث می‌شود مدل درست تومورها را یاد نگیرد.




شکل ۴ - چند نمونه از تست باسازی عکس های مدل **VAE** ساده

### ۳-۱. پیاده سازی Tri-VAE

برای پیاده‌سازی مدل Tri-VAE با جزئیات مورد نظر، مراحل زیر را انجام می‌دهیم:

**آماده‌سازی ورودی‌ها:** سه دسته ورودی (Anchor, Positive, Negative) آماده می‌شوند. که Anchor و Positive تصاویر سالم بدون نویز و Negative تصاویر سالم همراه با نویز Coarse است.

**معماری مدل:** Encoder ورودی‌های که تصاویر هستند را دریافت می‌کند. و معماری شامل لایه‌های کانولوشن، نرمال‌کردن و تابع فعال ساز است. خروجی ها تعبیه‌های $E_a$، $E_p$ و $E_n$ است.

Decoder برای بازسازی تصاویر استفاده می‌شود. بازسازی در دو مقیاس Coarse Scale با خروجی ۳۲*۳۲ و Full Scale با خروجی ۲۵۶*۲۵۶ است. و برای اتصال Encoder و Decoder از Gated Cross Skip Connections (GCS) استفاده می‌کنیم.

**تابع هزینه:**

Triplet Loss:

$$\text{Loss}_{\text{triplet}} = \max\left(0, \left|\left|E_a - E_p\right|\right|^2 - \left|\left|E_a - E_n\right|\right|^2 + \text{margin}\right)$$

L1 Loss (Coarse): خطای بازسازی در مقیاس coarse برای Anchor، Positive، و Negative.

L1 Loss (Full): خطای بازسازی در مقیاس full فقط برای Negative.

KL Divergence Loss: اختلاف بین توزیع تعبیه‌های Anchor و Positive.

SSIM Loss: کیفیت بازسازی Negative در مقیاس full.

## ۴-۱. ارزیابی در دیتاست BraTS (دو بعدی)

با توجه به مقاله و آموزش روی دیتاست سالم، برای تست با استفاده از دیتاست BrasTS که تومورهای مغزی را دارند باید عکس‌های باسازی کند و سپس قسمت آنومالی آن را تشخیص دهد. لذا بعد از آموزش مدل، ۱۰ بیمار از دیتاست BraTS انتخاب شدند. به منظور تمرکز بر بخش‌های آنومالی، اسلایس‌های دوبعدی استخراج و برای ارزیابی مدل Tri-VAE استفاده شدند. پس از اعمال مدل Tri-VAE بر روی تصاویر ورودی، تصاویر بازسازی‌شده دریافت شدند. برای ارزیابی دقت مدل در بازسازی، خطای بازسازی $\| x - \hat{x} \|$ محاسبه شد که در آن $x$ تصویر ورودی و $\hat{x}$ تصویر بازسازی‌شده است. این محاسبه به ما امکان می‌دهد تا میزان اختلاف بین تصویر ورودی و تصویر بازسازی‌شده توسط مدل را اندازه‌گیری کنیم. و همچنین از یک Threshold (مقدار آستانه)، مانند ۰.۱، برای کاهش نویز و بهبود دقت مدل استفاده شد. برای ارزیابی دقت پیش‌بینی‌های مدل، از معیار Dice نیز استفاده کردیم.

برای ارزیابی کیفی نتایج، چند نمونه از تصاویر ورودی، پیش‌بینی‌های آنومالی و ماسک‌های واقعی نمایش داده شد. این تصاویر به ما کمک می‌کنند تا عملکرد مدل را مشاهده کنیم.

مدل VAE دارای معماری ساده‌تر نسبت به Tri-VAE است و قادر به بازسازی تصاویر مغزی است. و دارای ویژگی‌های مناسبی برای داده‌های پزشکی می‌باشد.

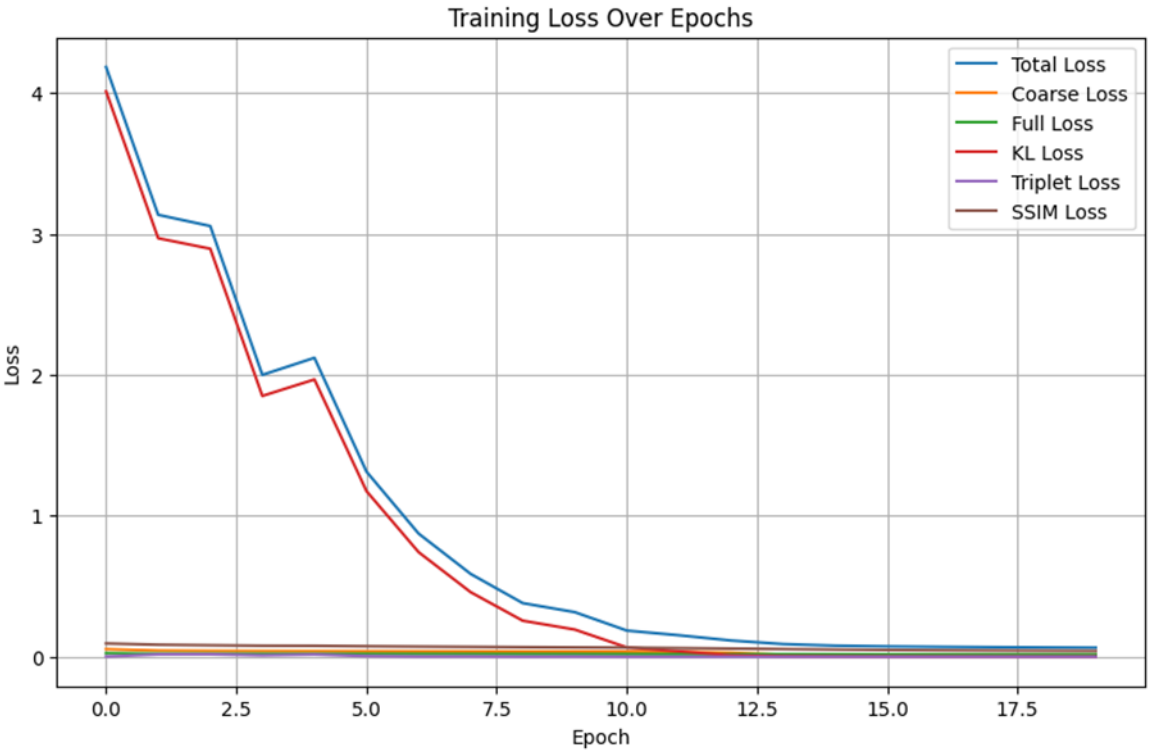
مدل Tri-VAE نسخه‌ای پیشرفته‌تر از VAE است که به‌طور خاص برای پردازش تصاویر پیچیده‌تر و سه‌بعدی (مانند تصاویر MRI مغز) طراحی شده است. این مدل شامل چندین مولفه و بخش برای یادگیری ویژگی‌ها در مقیاس‌های مختلف است. در نتیجه، Tri-VAE قادر به یادگیری ویژگی‌های پیچیده‌تری است که ممکن است در مدل ساده VAE قابل شبیه‌سازی نباشند. از ویژگی‌های این مدل می‌توان به معماری پیچیده‌تر و شامل بخش‌های متعددی برای پردازش بهتر داده‌های سه‌بعدی و ویژگی‌های چند مقیاسی است و عملکرد بهتری در بازسازی تصاویر و پیش‌بینی نواحی آنومالی، به‌ویژه در مقیاس‌های پیچیده‌تر دارد. و نسبت به مدل VAE بیشتر برای داده‌های پزشکی پیچیده مانند تصاویر مغزی مناسب است.

مدل VAE معمولاً قادر است تصاویر مغزی را بازسازی کند، اما در بخش‌هایی که ویژگی‌های آنومالی یا تومور مغزی وجود دارند، دقت خوبی ندارد. و در تشخیص مرزهای دقیق تومورها یا نواحی آنومالی با مشکل مواجه می‌شود، زیرا معماری ساده‌تری دارد. در تصاویر با نویز زیاد یا پیچیدگی‌های ساختاری بیشتر، بازسازی‌ها ممکن است کیفیت مطلوب را نداشته باشند.

در مقایسه با VAE مدل Tri-VAE به دلیل داشتن معماری پیچیده‌تر، قادر است تصاویر مغزی را با دقت بالاتری بازسازی کند و در شبیه‌سازی نواحی آنومالی (تومورها) بهتر عمل کند. این مدل در شبیه‌سازی مرزهای تومورها کمی بهتر عمل می‌کند و می‌تواند ویژگی‌های پیچیده‌تری از تصاویر MRI را شبیه‌سازی کند. از آنجا که Tri-VAE معماری چند مقیاسی دارد، قادر است با جزئیات بیشتری ویژگی‌های آنومالی‌های کوچک‌تر را شبیه‌سازی کند که ممکن است در VAE آموزش نگیرد.

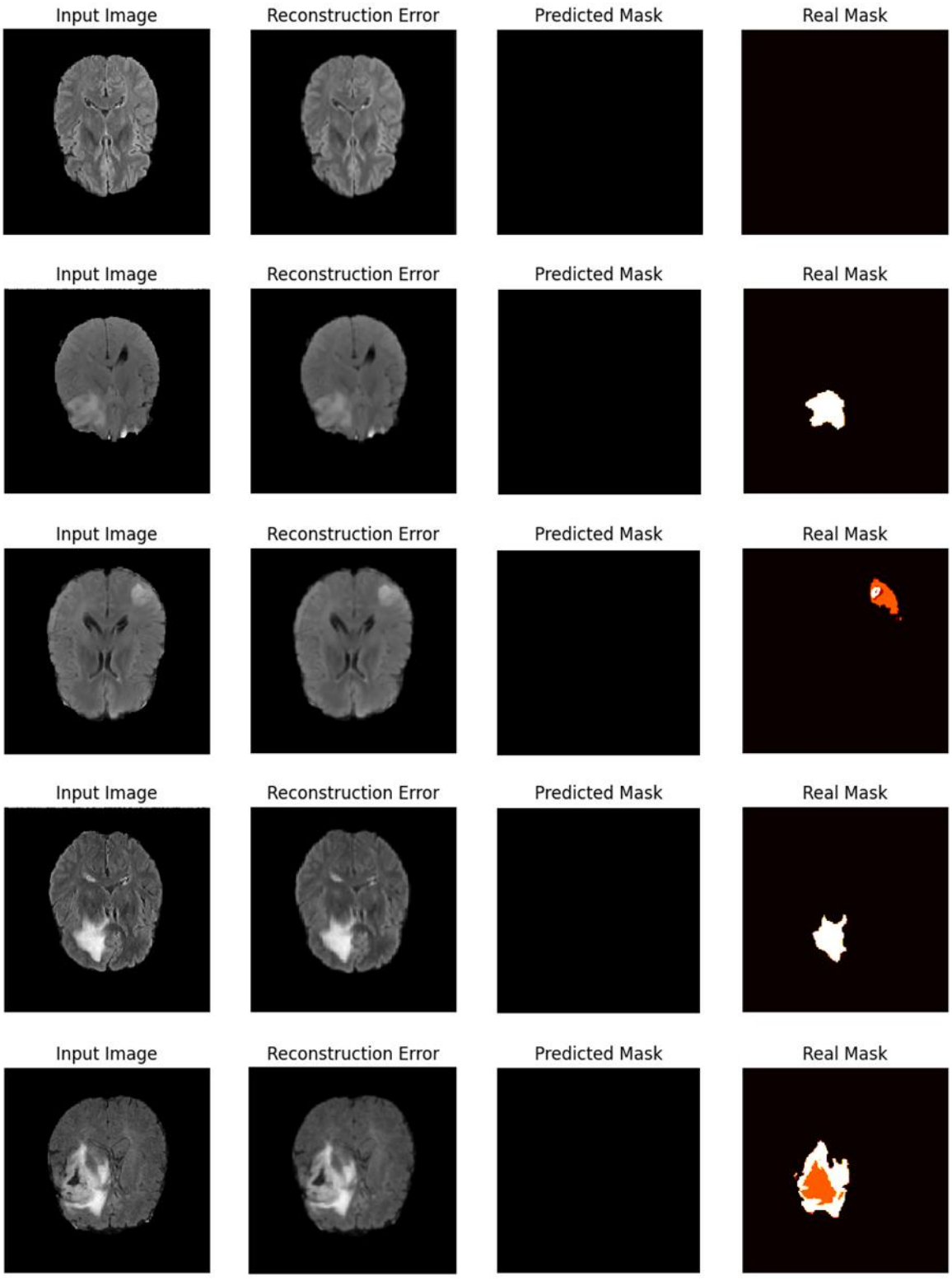
مدل VAE برای پروژه‌های ساده‌تر استفاده می‌شود و دارای سرعت بالاتر و هزینه‌های پردازش کمتری دارد، ولی در بازسازی دقیق تصاویر پیچیده‌تر و پیش‌بینی نواحی آنومالی دقت کمتری دارد. و مدل Tri-VAE به دلیل معماری پیچیده‌تر و قابلیت‌های پردازش چند مقیاسی، عملکرد بهتری در بازسازی تصاویر و پیش‌بینی نواحی آنومالی، به‌ویژه در تصاویر مغزی پیچیده و با جزئیات بالا، دارد. این مدل قادر است نواحی آنومالی دقیق‌تری را شبیه‌سازی کرده و در مقایسه با VAE، نتایج بهتری به‌دست آورد.

با وجود اینکه تعداد ایپاک‌ها در مدل Tri-VAE محدود به ۲۰ ایپاک بود و پیش‌پردازش‌ها در این پروژه ساده نگه‌داشته شدند، نتایج نشان‌دهنده‌ی عملکرد قابل‌قبول مدل در بازسازی تصاویر و پیش‌بینی نواحی آنومالی بود. با این حال، دقت مدل در مقایسه با نتایج مقاله‌های مشابه که از پردازش‌های سه‌بعدی پیچیده‌تر و تعداد ایپاک‌های بیشتر استفاده می‌کنند، پایین‌تر بود. این امر به‌طور عمده به دلیل محدودیت‌های تعداد ایپاک‌ها و نداشتن پس‌پردازش‌های پیچیده‌تر است.




شکل ۵ - نمودار **Loss** های مدل **Tri-VAE**

در این پروژه، مدل Tri-VAE در مقایسه با مدل ساده VAE عملکرد بهتری در بازسازی تصاویر و پیش‌بینی نواحی آنومالی داشت. با این حال، برای بهبود دقت مدل و نزدیک‌تر شدن به نتایج مقاله‌ها، استفاده از تعداد اپاک‌های بیشتر و اعمال پیش‌پردازش‌های پیچیده‌تر همچنان ضروری است. در نهایت، تحلیل دقیق نتایج و ارزیابی مدل از اهمیت بالایی برخوردار است و ساختار طراحی مدل باید بر اساس نیازهای خاص پروژه تنظیم شود. در بازسازی تصاویر نسبت به مدل VAE ساده خیلی بهتر کار کرده است ولی در تشخیص تومور همچنان مشکل دارد و البته در بعضی از موارد تشخیص آن به نسبت بیشتر از تصاویر در گزارش بود که متاسفانه چون به صورت رندوم عکس انتخاب شده است و بعد از چند اجرای مجدد نتوانستم خروجی بگیرم که تصاویر به ماسک بهتر را نمایش بدهد.




شکل ۶- نمونه ای از تصاویر برای تست مدل **Tri-VAE**

## ۵-۱. بخش امتیازی

پردازش داده‌های تصویربرداری پزشکی با مدل Tri-VAE، حذف نویز، و مقایسه دو روش Coarse Noise و Simplex Noise بر اساس شاخص Dice.

مراحل اصلی:

تولید داده مصنوعی: حجم سه‌بعدی باینری به‌عنوان Ground Truth.

Tri-VAE: پردازش اسلایس‌ها و ترکیب به حجم سه‌بعدی.

حذف نویز: فیلتر Median سه‌بعدی و حذف کامپوننت‌های کوچک.

نویز Simplex: اعمال نویز Simplex و مقایسه با Coarse Noise.

محاسبه Dice: ارزیابی عملکرد برای هر دو روش.

این روش را تا حدودی انجام شد ولی به دلیل کمبود سخت افزار محاسباتی آن را نتوانسیم اجرا کنیم و خطای آن را برطرف کنیم و به یک کلیتی از آن بسنده کردیم.

# پرسش۲. AdvGAN

## ۲-۱. آشنایی با حملات خصمانه و معماری AdvGAN

### ۲-۱-۱. توضیح FGSM و PGD و مقایسه با AdvGAN

۱. **روش(Fast Gradient Sign Method) FGSM** :

- **شرح FGSM** :یک حمله **جعبه سفید** است که از گرادیان تابع خطا نسبت به ورودی داده‌ها استفاده می‌کند تا تغییرات کوچک و هدفمند به ورودی وارد کند. این تغییرات به گونه‌ای اعمال می‌شوند که مدل نتواند پیش‌بینی صحیحی انجام دهد. در این روش، حمله به طور مستقیم به پارامترهای مدل وابسته است.
- **فرمول**

$$x' = x + \epsilon \cdot \text{sign}(\nabla_x J(\theta, x, y))$$

  - در اینجا:
    - $x$ ورودی اصلی
    - $x'$ نمونه خصمانه
    - $J(\theta, x, y)$ تابع خطا(loss function)
    - $\epsilon$ میزان تغییرات
- **نوع حمله :جعبه سفید** (White-box attack)
- **مزایا:**
  - ساده و سریع برای ارزیابی‌های اولیه
  - به راحتی قابل پیاده‌سازی
- **محدودیت‌ها:**
  - به دلیل سادگی، برای حملات پیچیده و مدل‌های مقاوم ناکارآمد است.
  - معمولاً نمی‌تواند در برابر روش‌های دفاعی مانند آموزش خصمانه مقاومت کند.

۲. **روش PGD (Projected Gradient Descent)** :

- **شرح PGD** :یک نسخه پیشرفته‌تر و تکراری از **FGSM** است که در هر مرحله تغییرات کوچکی را اعمال کرده و سپس نتیجه را به محدوده‌ای خاص برمی‌گرداند. این کار باعث می‌شود که حمله بیشتر به ساختار ورودی بپردازد و اثرات آن تدریجاً بیشتر شود.
- **فرمول**

$$x_{t+1}' = \text{Clip}_{x,\epsilon}(x_t + \alpha \cdot \text{sign}(\nabla_x J(\theta, x_t, y)))$$

  - در اینجا:
    - $x_t$ ورودی در مرحله t
    - $\alpha$ اندازه گام در هر مرحله
    - $\text{Clip}_{x,\epsilon}$ اطمینان حاصل می‌کند که تغییرات در محدوده تعریف شده باقی بمانند.
- **نوع حمله** : **جعبه سفید** (White-box attack)
- **مزایا**:
  - نرخ موفقیت بالاتر نسبت به **FGSM** به دلیل استفاده از تکرار در مراحل مختلف
  - مقاوم‌تر در برابر مدل‌های پیچیده و دفاع‌های اولیه
- **محدودیت‌ها**:
  - زمان‌بر و نیازمند محاسبات بیشتر به دلیل تکرار در هر مرحله
  - در حملات زمان‌بر و پیچیده می‌تواند ناکارآمد باشد

۳. **مدل AdvGAN (Adversarial Generative Adversarial Networks)** :

- **شرح AdvGAN** :از معماری شبکه‌های مولد متخاصم (GAN) برای تولید نمونه‌های خصمانه استفاده می‌کند. این مدل یک تولید کننده (Generator) و یک تمایزگر (Discrimintor) دارد که به طور همزمان تلاش می‌کنند تا نمونه‌هایی تولید کنند که برای مدل هدف فریبنده باشند، در حالی که کیفیت بصری بالا و شباهت زیادی به نمونه‌های اصلی دارند.

- **فرمول**

$$L=L^{f}_{adv}+\alpha L_{GAN}+\beta L_{hinge}$$

  - در اینجا:
    - Ladv تابع خطای حمله
    - LGAN تابع خطای GAN برای حفظ کیفیت بصری
    - Lhinge تابعی برای محدود کردن اندازه تغییرات

در بخش ۲-۱-۳ به تفصیل در رابطه با تابع هزینه و کارکرد هر بخش توضیح داده شده است.

- **نوع حمله :جعبه سفید** و **جعبه سیاه** (White-box and Black-box attacks)
- **مزایا:**

  - **کیفیت بصری بالا** :نمونه‌های خصمانه تولید شده توسط **AdvGAN**بسیار شبیه به تصاویر اصلی هستند و تشخیص آن‌ها دشوار است.
  - **کارایی بالا** :پس از آموزش Generator، می‌توان نمونه‌های خصمانه را به سرعت تولید کرد بدون اینکه نیازی به دسترسی به مدل هدف باشد.
  - **مقاومت در برابر دفاع‌ها** :این روش می‌تواند در برابر روش‌های دفاعی مختلف، مانند آموزش خصمانه، عملکرد بهتری داشته باشد.
  - **حملات جعبه سیاه AdvGAN** :قادر است حملات را در شرایط **جعبه سیاه** نیز انجام دهد، که برای بسیاری از روش‌های دیگر مانند **FGSM**و **PGD** چالش‌برانگیز است.

- **محدودیت‌ها:**

  - نیاز به آموزش اولیه Generator
  - پیچیدگی بیشتر در مقایسه با **FGSM**و **PGD**

**مقایسه AdvGAN با FGSM وPGD**

۱. **حملات جعبه سفید:**

- **FGSM**و **PGD**تنها در شرایط **جعبه سفید** به خوبی کار می‌کنند، جایی که دسترسی کامل به پارامترهای مدل وجود دارد. این روش‌ها در تولید نمونه‌های خصمانه برای مدل‌های خاص موفق هستند، اما به دلیل سادگی، نمی‌توانند به راحتی مدل‌های پیچیده را تحت تاثیر قرار دهند.
- **AdvGAN**نیز در **جعبه سفید** می‌تواند به خوبی عمل کند و با استفاده از شبکه‌های مولد متخاصم، نمونه‌های خصمانه‌ای با کیفیت بصری بالا تولید کند که برای مدل‌های مقاوم نیز کارایی دارند.

۲. **حملات جعبه سیاه:**

- در حملات **جعبه سیاه**، جایی که اطلاعاتی از مدل هدف در دسترس نیست، **FGSM**و **PGD**معمولاً نمی‌توانند به خوبی عمل کنند. این حملات نیازمند استفاده از **انتقال** یا استفاده از مدل‌های کمکی هستند.
- **AdvGAN**با استفاده از روش **مدل‌سازی مجدد** (distillation)، قادر است حملات **جعبه سیاه** را به طور مؤثر انجام دهد. این ویژگی باعث می‌شود که **AdvGAN**نسبت به **FGSM**و **PGD**در این زمینه مزیت داشته باشد.

۳. **کیفیت بصری:**

- **FGSM**و **PGD**به دلیل سادگی و استفاده از تغییرات مستقیم در داده‌ها، معمولاً باعث می‌شوند نمونه‌های خصمانه با تغییرات آشکار در تصویر ایجاد شوند.
- **AdvGAN**با استفاده از معماری **GAN**می‌تواند تغییرات بسیار ظریف‌تری اعمال کند که به راحتی قابل تشخیص برای انسان‌ها نیستند و تصویر همچنان طبیعی باقی می‌ماند.

۴. **زمان تولید:**

- **FGSM**به دلیل سادگی خود بسیار سریع است و زمان کمی برای تولید نمونه‌های خصمانه نیاز دارد.
- **PGD**به دلیل تکرار مراحل، زمان بیشتری نسبت به **FGSM**می‌برد.
- **AdvGAN**پس از آموزش، قادر است نمونه‌های خصمانه را به سرعت تولید کند، که این ویژگی آن را برای حملات سریع و گسترده مناسب می‌سازد.

### ۲-۱-۲. تفاوت‌های کلیدی AdvGAN و GAN

- **چگونه AdvGAN از گرادیان‌ها یا خروجی‌های مدل هدف در زمان آموزش استفاده می‌کند؟**

- **AdvGAN** از یک **مدل هدف** (Target Model) در فرآیند آموزش استفاده می‌کند. این مدل هدف می‌تواند یک مدل طبقه‌بندی (مثل مدل‌های شناسایی تصویر) باشد. در طول آموزش **AdvGAN**، **Generator** و **Discriminator** معمولاً همانطور که در **GAN ساده** انجام می‌شود، آموزش داده می‌شوند، اما **AdvGAN**از اطلاعات بیشتری استفاده می‌کند.
- در این روش، علاوه بر **Discriminator** که به **Generator** کمک می‌کند تا نمونه‌هایی تولید کند که به نظر واقعی بیایند، **AdvGAN** از **گرادیان‌ها** یا **خروجی‌های مدل هدف** برای هدایت **Generator** به سمت تولید **نمونه‌های خصمانه** استفاده می‌کند.
- در واقع، **Generator** از **گرادیان‌های مدل هدف** استفاده می‌کند تا تغییراتی در ورودی‌ها اعمال کند که باعث می‌شود مدل هدف دچار خطا شود و نتواند پیش‌بینی درستی انجام دهد. این به معنای این است که **Generator** نه تنها باید **Discriminator** را فریب بدهد، بلکه باید **مدل هدف** را نیز فریب دهد و آن را به اشتباه بیاندازد.
- به طور خاص، در هر مرحله از آموزش، **AdvGAN** به **Generator** این اطلاعات را می‌دهد که چگونه باید تغییرات را اعمال کند تا مدل هدف اشتباهات بیشتری داشته باشد. به عبارت دیگر، **AdvGAN**از **خروجی‌ها** و **گرادیان‌ها**ی مدل هدف برای اصلاح **Generator**و تولید نمونه‌هایی استفاده می‌کند که **نه تنها واقعی به نظر می‌آیند، بلکه مدل هدف را نیز به اشتباه می‌اندازند.**

- **چگونه AdvGAN نمونه‌های متخاصم تولید می‌کند و چگونه این مدل قادر است همزمان وفاداری بصری به تصویر اصلی و قابلیت حمله به مدل را حفظ کند؟**

- **AdvGAN** به طور همزمان دو هدف را دنبال می‌کند:

۱. **حفظ وفاداری بصری به تصویر اصلی** :این هدف از طریق **Discriminator** انجام می‌شود. **Discriminator** به **Generator**کمک می‌کند تا تغییرات اعمال‌شده به تصویر

اصلی آنقدر کوچک و نامحسوس باشد که تصویر تولیدشده تقریباً مشابه تصویر اصلی باقی بماند. به این ترتیب، **AdvGAN** اطمینان حاصل می‌کند که نمونه‌های تولیدشده از نظر بصری قابل تشخیص نباشند و برای انسان‌ها غیرقابل شناسایی باشند.

۲. **حمله به مدل هدف** :در عین حال، **Generator** با استفاده از **گرادیان‌های مدل هدف** به گونه‌ای آموزش می‌بیند که نمونه‌های تولیدشده به گونه‌ای طراحی شوند که مدل هدف نتواند آن‌ها را به درستی شبیه‌سازی کند. این یعنی تصویر تولید شده باید باعث اشتباه مدل هدف شود، یعنی مدل هدف باید پیش‌بینی نادرستی انجام دهد.

- **چگونگی ایجاد نمونه‌های متخاصم**:
  - در ابتدا، **Generator** یک تصویر **اصلی** را می‌گیرد و یک **Perturbation** (تغییرات کوچک) به آن اضافه می‌کند.
  - سپس این تصویر تغییر یافته به **Discriminator**داده می‌شود که ارزیابی می‌کند که آیا این تصویر از نظر بصری شبیه به نمونه واقعی است یا خیر.
  - در کنار این، تصویر به **مدل هدف** داده می‌شود و بررسی می‌شود که آیا این تصویر فریب‌دهنده است و مدل هدف نمی‌تواند آن را به درستی دسته‌بندی کند.
  - هدف **Generator**این است که تصویر تولیدی نه تنها **به نظر واقعی** بیاید (از نظر **Discriminator**)، بلکه در عین حال **مدل هدف را فریب دهد** (از نظر **مدل هدف**).
- **چگونه وفاداری بصری و حمله به مدل همزمان حفظ می‌شود**:
  - در **AdvGAN**، دو نوع **loss**به صورت همزمان وجود دارد:

۱. **loss**برای ارزیابی مشابهت بصری: این قسمت از فرآیند به **Discriminator** مربوط می‌شود و به **Generator**کمک می‌کند که تصویر تولید شده به نظر واقعی برسد.

۲. **loss**برای ارزیابی موفقیت حمله: این **loss**از **مدل هدف** به دست می‌آید و به **Generator**کمک می‌کند که نمونه تولیدی باعث اشتباه مدل هدف شود.

  - ترکیب این دو **loss**باعث می‌شود که **AdvGAN**به صورت همزمان **وفاداری بصری** و **موفقیت در حمله** را حفظ کند. در واقع، **Generator**تلاش می‌کند که تصویر تولیدی

تا حد ممکن به تصویر اصلی شبیه باشد با کمک **Discriminator**، اما در عین حال باید مدل هدف را فریب دهد با کمک **مدل هدف**.

- **در نتیجه، AdvGAN**قادر است نمونه‌های متخاصم تولید کند که نه تنها به طور مؤثری مدل هدف را فریب دهند، بلکه از نظر بصری هم شبیه به تصاویر اصلی باقی بمانند. این ویژگی آن را از روش‌های دیگر متمایز می‌کند که معمولاً نیاز به توازن بین حمله مؤثر و حفظ کیفیت بصری ندارند.

### ۲-۱-۳. توضیح سه تابع هزینه استفاده شده در AdvGAN

نمونه متن

**1. GAN Loss ($L_{GAN}$)**

این تابع ضرر به‌طور عمده برای اطمینان از واقعی بودن بصری نمونه‌های adversarial تولید شده استفاده می‌شود و اطمینان می‌دهد که این نمونه‌ها از داده‌های واقعی قابل تمیز دادن نباشند. این تابع ضرر به تولیدperturbation هایی کمک می‌کند که به‌طور طبیعی با داده‌های اصلی هماهنگ باشند و از نظر بصری مشابه آنها باشند.

**رابطه ریاضی:**

$$L_{GAN}=E_x[\log D(x)]+E_x[\log(1-D(x+G(x)))]$$

جایی که:

- $x$داده واقعی از کلاس اصلی است.
- $G(x)$ perturbation تولید شده توسط شبکه Generator برای ورودی $x$ است.
- $x+G(x)$ نمونه adversarial است.
- $D(x)$خروجی شبکه Discriminator زمانی است که تصویر اصلی $x$ به آن داده می‌شود و احتمال اینکه $x$ داده واقعی باشد را پیش‌بینی می‌کند.
- $D(x+G(x))$پیش‌بینی احتمال Discriminator است که آیا نمونه $x+G(x)$ adversarial واقعی است.

**توضیح:**

این تابع ضرر کمک می‌کند تا Generator نمونه‌های adversarial تولید کند که از نظر بصری مشابه داده‌های واقعی باشند و Discriminator را فریب دهند. به این ترتیب، Generatorبه تولیدperturbation هایی هدایت می‌شود که به‌طور طبیعی با داده‌های اصلی هم‌راستا هستند و حملات adversarial مؤثری تولید می‌کند.

**2. Adversarial Loss ($L^f_{adv}$)**

این تابع ضرر به‌طور خاص برای ایجاد نمونه‌های adversarial طراحی شده است که مدل هدف fرا به اشتباه بیندازند. این تابع ضرر به این صورت عمل می‌کند که اطمینان می‌دهد نمونه‌های adversarial به درستی مدل را گمراه می‌کنند، خواه هدف حمله هدفمند باشد یا حمله غیرهدفمند.

**رابطه ریاضی:**

$$L^f_{adv}=Ex[Lf(x+G(x),t)]$$

جایی که:

- tکلاس هدف برای حمله هدفمند است.
- $x+G(x)$ نمونه adversarial است.
- fمدل هدف است.
- $Lf(x+G(x),t)$ تابع ضرر استفاده شده توسط مدل fاست (مثلاً ضرر کراس-انتروپی) که فاصله بین پیش‌بینی مدل برای $x+G(x)$ و کلاس هدف t را اندازه‌گیری می‌کند.

**توضیح:**

این تابع ضرر اطمینان می‌دهد که نمونه‌های adversarial تولید شده به درستی مدل هدف را گمراه کنند. در حملات هدفمند، این ضرر باعث می‌شود که پیش‌بینی مدل به کلاس هدف نزدیک‌تر شود. در حملات غیرهدفمند، هدف این است که مدل بیشتر دچار اشتباه شود. این ویژگی برای موفقیت حملات adversarialضروری است.

**3. Hinge Loss ($L_{hinge}$)**

این تابع ضرر به‌طور خاص برای تنظیم اندازه perturbation استفاده می‌شود و اطمینان می‌دهد که نمونه‌های adversarial مشابه تصاویر اصلی باقی بمانند. این تابع ضرر به Generator کمک می‌کند تا محدودیت‌هایی برای اندازه perturbation تعیین کند تا نمونه‌های adversarial بزرگ یا آشکار نشوند.

**رابطه ریاضی:**

$$L_{hinge}=E_x[\max(0,\|G(x)\|_2-c)]$$

جایی که:

- $G(x)$ perturbation تولید شده توسط Generator برای ورودی $x$است.
- $c$یک حد مشخص شده توسط کاربر برای norm-2L از perturbation است.

**توضیح:**

این تابع ضرر وقتی که اندازه $G(x)$ perturbation از حد مجاز $c$ بیشتر شود،Generator را جریمه می‌کند. با این کار، Generator هدایت می‌شود تاperturbation هایی تولید کند که از نظر بصری شبیه به داده‌های اصلی باقی بمانند و حملات را کمتر قابل شناسایی کنند.

**تابع ضرر کلی برایAdvGAN**

تابع ضرر نهایی در AdvGAN ترکیبی از این سه تابع ضرر است که با پارامترهای $\alpha$و $\beta$وزن‌دهی شده‌اند:

$$L=L^f_{adv}+\alpha L_{GAN}+\beta L_{hinge}$$

جایی که:

- $\alpha$و $\beta$کنترل می‌کنند که هر کدام از این ضررها چقدر اهمیت دارند.
- تابع ضرر کلی با هدف تولید نمونه‌های adversarial بهینه‌سازی می‌شود که نه تنها از نظر بصری واقعی و مشابه داده‌های اصلی هستند، بلکه مدل هدف را به‌طور مؤثر فریب می‌دهند و perturbationها در حد قابل قبولی باقی می‌مانند.

**نقش هر تابع ضرر درAdvGAN**

- **GAN Loss**: این تابع ضرر به Generator کمک می‌کند تا نمونه‌های adversarial تولید کند که از نظر بصری مشابه داده‌های واقعی باشند و به طور مؤثر Discriminator را فریب دهند.
- **Adversarial Loss**: این تابع ضرر به Generator کمک می‌کند تا نمونه‌های adversarial تولید کند که مدل هدف را به اشتباه بیندازند و حمله را موفقیت‌آمیز کند.
- **Hinge Loss**: این تابع ضرر اندازهperturbation ها را کنترل می‌کند و از این طریق اطمینان می‌دهد که نمونه‌های adversarial از داده‌های اصلی به‌طور عمده مشابه باقی بمانند و حملات از نظر بصری قابل شناسایی نباشند.

با بهینه‌سازی این تابع ضرر ترکیبی، AdvGANنمونه‌های adversarial تولید می‌کند که نه تنها از نظر بصری واقعی و مشابه داده‌های اصلی هستند، بلکه مدل هدف را به‌طور مؤثر فریب می‌دهند وperturbation ها در حد قابل قبولی باقی می‌مانند. این ویژگی‌ها باعث می‌شود که حملات adversarial تولید شده توسط AdvGANنه تنها مؤثر بلکه غیرقابل شناسایی و پایدار باشند.

### ۲-۱-۴. تفاوت حمله جعبه سفید و جعبه سیاه

۱. **حمله جعبه سفید(White-box Attack)** :

- **تعریف** :در حملات جعبه سفید، مهاجم دسترسی کامل به مدل هدف دارد. این دسترسی شامل معماری مدل، پارامترها، وزن‌ها و داده‌های آموزشی است. به دلیل این دسترسی کامل، مهاجم می‌تواند گرادیان‌ها را محاسبه کرده و مستقیماً بهینه‌سازی pertubations (اختلال‌ها) را انجام دهد تا مثال‌های adversial به‌وجود آورد.
- **روش حمله** :از آنجایی که مهاجم به تمام اطلاعات مدل دسترسی دارد، می‌تواند به راحتی با استفاده از روش‌هایی مانند **Fast Gradient Sign Method (FGSM)**یا **Project Gradient Descent (PGD)** perturbations را به ورودی‌ها اعمال کند تا مدل را فریب دهد.
- **مزیت‌ها** :دسترسی به اطلاعات داخلی مدل این حملات را بسیار دقیق و مؤثر می‌کند. با استفاده از گرادیان‌ها، مهاجم می‌تواند دقیقاً اختلال‌هایی که مدل را فریب می‌دهند، تولید کند.
- **چالش‌ها** :در این نوع حملات، نیاز به دسترسی مستمر به مدل هدف وجود دارد، که این ممکن است در دنیای واقعی مشکلاتی ایجاد کند.

۲. **حمله جعبه سیاه(Black-box Attack)** :

- **تعریف** :در حملات جعبه سیاه، مهاجم هیچ دسترسی به معماری داخلی یا پارامترهای مدل هدف ندارد. تنها چیزی که در اختیار مهاجم است، رفتار ورودی-خروجی مدل است (یعنی ورودی‌ها و پیش‌بینی‌هایی که مدل برای آنها تولید می‌کند).
- **روش حمله** :در این نوع حمله، مهاجم نمی‌تواند مستقیماً مدل را مورد حمله قرار دهد و باید از روش‌های دیگری برای تولید مثال‌های adversial استفاده کند. از جمله این روش‌ها می‌توان به استفاده از مدل‌های جایگزین (surrogate models) یا استفاده از تکنیک‌هایی

مانند **Zeroth-Order Optimization (ZOO)** اشاره کرد که در آن مهاجم با حداقل درخواست از مدل، تلاش می‌کند مدل هدف را فریب دهد.

- **مزیت‌ها** :این نوع حمله بیشتر به واقعیت نزدیک است زیرا در بسیاری از مواقع مهاجم فقط دسترسی به خروجی مدل را دارد و نمی‌تواند به داده‌های داخلی آن دسترسی پیدا کند.
- **چالش‌ها** :به دلیل عدم دسترسی به اطلاعات داخلی مدل، این حملات معمولاً پیچیده‌تر و دشوارتر هستند و نرخ موفقیت کمتری دارند.

**استفاده از AdvGAN در حملات سفید و سیاه:**

مدل **AdvGAN** برای انجام حملات در محیط‌های سفید، نیمه‌سفید و سیاه طراحی شده است. در اینجا نحوه استفاده از AdvGAN برای حملات سیاه به‌ویژه با استفاده از استراتژی‌های مختلف توضیح داده شده است:

۱. **حملات نیمه‌سفید(Semi-whitebox Attacks)** :

- در این نوع حمله، شبکه **Generator** در AdvGAN با دسترسی به مدل هدف آموزش داده می‌شود. پس از آموزش **Generator**، این شبکه می‌تواند بدون نیاز به دسترسی بیشتر به مدل هدف، pertubations را برای هر ورودی تولید کند.
- این تفاوت مهمی با حملات سفید است که نیازمند دسترسی مداوم به مدل هدف هستند. در این حالت، مهاجم نیازی به درخواست‌های مکرر از مدل هدف ندارد.

۲. **حملات جعبه سیاه(Black-box Attacks)** :

- در حملات سیاه، AdvGAN از یک مدل **Distilled** برای شبیه‌سازی رفتار مدل هدف استفاده می‌کند. این مدل distilled با استفاده از خروجی‌های مدل سیاه آموزش داده می‌شود.
- **Distillation ایستا(Static Distillation)** : ابتدا یک مدل distilled با داده‌هایی که از داده‌های آموزشی مدل هدف جدا هستند، آموزش داده می‌شود. پس از آماده شدن مدل distilled، همان استراتژی حمله‌ای که در حملات سفید استفاده می‌شود، اعمال می‌شود.

به عبارت دیگر، شبکه **Generator**برای تولید مثال‌های adversial که مدل distilled را فریب دهند، آموزش داده می‌شود.

- **چالش** :این روش ممکن است همیشه موثر نباشد، زیرا مدل distilled ممکن است رفتار مدل هدف را به‌طور دقیقی شبیه‌سازی نکند، به‌ویژه در مورد مثال‌های adversial.

- **Distillation پویا(Dynamic Distillation)** : برای بهبود عملکرد در حملات سیاه، از استراتژی distillation پویا استفاده می‌شود. در این رویکرد، مدل distilled و **Generator**به صورت مشترک و در یک فرایند تکراری آموزش داده می‌شوند:
  - در هر تکرار، **Generator**به‌روز می‌شود تا مثال‌های adversial تولید کند که مدل distilled را فریب دهند.
  - سپس، مدل distilled بر اساس تصاویر آموزشی اصلی و مثال‌های adversial که توسط **Generator**تولید شده‌اند، به‌روز می‌شود.
  - این فرایند پویا به مدل distilled اجازه می‌دهد تا رفتار مدل هدف را در برابر مثال‌های adversial بهتر شبیه‌سازی کند، که موجب افزایش نرخ موفقیت حمله می‌شود.

- **مزیت** :استفاده از **Distillation پویا** باعث می‌شود که AdvGAN قادر باشد به طور مؤثر حملات سیاه را انجام دهد، بدون اینکه به ویژگی‌های انتقال‌پذیری (transferability) متکی باشد.

### ۲-۱-۵. دو مقاله دیگر در حوزه AdvGAN

- **مقاله اول:** **A novel approach to generating high-resolution adversarial examples**

ایده‌های توسعه‌یافته در این مقاله به طور مشخص بر اساس چارچوب اولیه‌ی AdvGAN طراحی شده‌اند. هدف اصلی این مقاله، بهبود نرخ موفقیت حملات و ایجاد نمونه‌های خصمانه با کیفیت بالا و قابل درک بصری است. این مقاله با بهره‌گیری از تکنیک‌های **PCA**و **KPCA**، ویژگی‌های اصلی نمونه ورودی را به latent variables نگاشت (map) می‌کند. این کار باعث می‌شود که نمونه‌های خصمانه‌ی تولیدشده شباهت زیادی به نمونه‌های اصلی داشته باشند و در عین حال موفق به فریب مدل‌های پیشرفته شوند.

یکی از دستاوردهای این مقاله، بهبود نرخ موفقیت حملات روی مجموعه داده‌های با وضوح بالا نظیر ImageNetو LSUN است. این مقاله موفق شده است **نرخ موفقیت حمله‌ی ۹۰.۳۰ %**را روی مدل ResNet152در مجموعه داده‌ی CIFAR-10 به دست آورد، که نسبت به AdvGAN اولیه (با نرخ موفقیت ۸۸.۶۹٪) اندکی بهبود یافته است. همچنین، این روش توانسته است نمونه‌های خصمانه‌ای تولید کند که از لحاظ بصری بسیار طبیعی به نظر می‌رسند و از نظر انتقال‌پذیری به سایر مدل‌ها نیز عملکرد بهتری دارند.

**پیشرفت‌های این مقاله نسبت بهAdvGAN :**

این مقاله با تمرکز بر دو چالش اصلیAdvGAN ، یعنی کار با مجموعه داده‌های با وضوح بالا و ایجاد تغییرات ظریف ولی موثر، چارچوب اولیه را گسترش داده است. برخلاف روش اولیه که بیشتر بر تولید نمونه‌ها برای داده‌های کوچک تمرکز داشت، این مقاله از ابزارهایی نظیر **PCA/KPCA**و افزایش داده (Data Augmentation)استفاده کرده است تا ویژگی‌های اصلی نمونه‌های ورودی بهتر حفظ شوند. در نهایت، با ادغام این روش‌ها با ساختار اولیه‌یAdvGAN ، نویسندگان موفق به تولید نمونه‌هایی شدند که هم قابل‌قبول از دیدگاه انسانی هستند و هم قادر به فریب دادن مدل‌های هدف با نرخ موفقیت بالا.

- **مقاله دوم:** [Generating Adversarial Examples in One Shot with Image-To-Image Translation GAN]

این مقاله با ارائه یک روش جدید برای تولید نمونه‌های خصمانه، به طور مستقیم چارچوب **AdvGAN** را گسترش می‌دهد. برخلاف روش‌های سنتی مبتنی بر بهینه‌سازی و حتی خود **AdvGAN**که به تولید **Perturbation**می‌پردازند و آنها را به تصاویر اصلی اضافه می‌کنند، این روش مستقیماً نمونه‌های خصمانه را از تصاویر ورودی تولید می‌کند. این ویژگی باعث می‌شود فرآیند تولید سریع‌تر و ساده‌تر باشد.

**ایده‌های کلیدی مقاله:**

۱. **تولید مستقیم نمونه‌های خصمانه:**

- در این روش، به جای تولید **Perturbation**و اعمال آن روی تصویر، نمونه‌های خصمانه به صورت مستقیم از تصاویر ورودی تولید می‌شوند. این امر نه تنها فرآیند را ساده‌تر می‌کند، بلکه باعث هم‌خوانی بهتر تغییرات با شکل‌ها و لبه‌های اصلی تصویر می‌شود.

۲. **کیفیت بصری بهتر:**

- نمونه‌های تولیدشده به دلیل تطابق بهتر با ویژگی‌های تصویر ورودی (مانند لبه‌ها و شکل‌ها)، ظاهری طبیعی‌تر دارند و تغییرات ایجادشده برای چشم انسان تقریباً غیرقابل تشخیص هستند.

۳. **عملکرد بالا**:

- این روش روی مجموعه‌داده‌های **MNIST** و **CIFAR-10** آزمایش شده و نشان داده است که عملکردی بهتر یا مشابه با **AdvGAN** از نظر نرخ حمله و میزان **Perturbation** دارد. همچنین در مقایسه با حملات مبتنی بر بهینه‌سازی مانند **حمله Carlini & Wagner (CW)** نتایج رقابتی ارائه داده است.

**توسعه‌های مقاله نسبت به AdvGAN :**

این مقاله با حذف نیاز به تولید و اعمال **Perturbation**، فرآیند تولید نمونه‌های خصمانه را ساده‌تر و سریع‌تر کرده است. علاوه بر این، با تمرکز بر حفظ ساختارهای اصلی تصویر ورودی، نمونه‌هایی تولید می‌کند که برای انسان طبیعی‌تر به نظر می‌رسند و هم‌زمان قدرت حمله‌ی بالایی دارند. این ویژگی‌ها این روش را بهبود مستقیمی برای **AdvGAN** تبدیل می‌کند.

## ۲-۲. پیاده‌سازی مدل AdvGAN

### ۲-۲-۱. دانلود و آماده‌سازی اولیه داده

CIFAR-10 یکی از دیتاست‌های استاندارد برای مسائل یادگیری ماشین و به‌خصوص یادگیری عمیق در حوزه پردازش تصویر است. این دیتاست شامل **60,000 تصویر رنگی** با ابعاد **32x32 پیکسل** است که در **10 کلاس مختلف** دسته‌بندی شده‌اند. هر کلاس شامل **6,000 تصویر** می‌باشد.

کلاس‌های موجود در این دیتاست عبارتند از:

۱. Airplane
۲. Automobile
۳. Bird
۴. Cat
۵. Deer
۶. Dog
۷. Frog

۸. Horse

۹. Ship

۱۰. Truck

این دیتاست به دو بخش تقسیم می‌شود:

- **داده‌های آموزشی:** 50,000 تصویر
- **داده‌های تست:** 10,000 تصویر

**فرضیات در پیاده‌سازی**

۱. تمامی داده‌ها در گوگل درایو ذخیره می‌شوند تا در اجرای‌های بعدی نیازی به دانلود مجدد نباشد.

۲. از ابزارهای torchvisionو torch.utils.dataبرای دانلود، بارگذاری، و تقسیم‌بندی داده‌ها استفاده می‌کنیم.

۳. نرمال‌سازی تصاویر با مقادیر میانگین و انحراف معیار از پیش تعیین‌شده برای CIFAR-10 انجام شده است:

- میانگین: (0.4914,0.4822,0.4465)
- انحراف معیار: (0.2470,0.2435,0.2616)

۴. از یک مدل از پیش آموزش دیده (Pretrained Model) با معماری **ResNet-20**استفاده شده است که برای CIFAR-10 بهینه شده است.

**گام‌های پیاده‌سازی**

۱. **دانلود و بارگذاری دیتاست:CIFAR-10**

- دیتاست از طریق torchvision.datasetsدانلود و در مسیر مشخصی در گوگل درایو ذخیره می‌شود.
- داده‌ها به سه مجموعه تقسیم شدند:
  - **آموزشی(Training)** : شامل ۴۵,۰۰۰ تصویر
  - **اعتبارسنجی(Validation)** : شامل ۵,۰۰۰ تصویر (درصدی برای اختصاص به دیتای Validation ذکر نشده بود پس این عدد را خود فرض کردیم.)

- **تست :(Test)** شامل ۱۰,۰۰۰ تصویر

  - از DataLoaderبرای بارگذاری دسته‌ای (batch) داده‌ها استفاده شده است.

۲. **نمایش نمونه تصاویر:**

  - 5نمونه تصویر تصادفی از مجموعه داده آموزشی نمایش داده شد.
  - تصاویر به دلیل نرمال‌سازی نیاز به تبدیل داشتند تا بتوان آن‌ها را به شکل قابل مشاهده به نمایش گذاشت.

۳. **بارگذاری مدل ResNet-20** :

  - مدل ResNet-20 از مخزن torch.hubبارگذاری شد.
  - این مدل از پیش آموزش دیده برای دیتاست CIFAR-10 طراحی شده و دارای توانایی خوبی در طبقه‌بندی تصاویر این دیتاست است.

۴. **محاسبه دقت مدل:**

  - مدل بر روی داده‌های تست بدون نیاز به آموزش مجدد (Fine-Tuning) ارزیابی شد.
  - دقت محاسبه شده برای مدل ResNet-20 روی داده‌های تست **۹۲.۱۲**٪ بود.

```python
target_model = torch.hub.load(
    "chenyaofo/pytorch-cifar-models",
    "cifar10_resnet20",
    pretrained=True
)
target_model.eval()

correct = 0
total = 0
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
target_model.to(device)

with torch.no_grad():
    for images, labels in test_loader:
        images = images.to(device)
        labels = labels.to(device)
        outputs = target_model(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

accuracy = 100.0 * correct / total
print(f"Accuracy of ResNet-20 on CIFAR-10 test images: {accuracy:.2f}%")
```

شکل ۷- بخشی از کد تعریف مدل هدف و ارزیابی

```
Mounted at /content/drive
Files already downloaded and verified
Files already downloaded and verified
```


```
/usr/local/lib/python3.10/dist-packages/torch/hub.py:330: UserWarning: You are about to download and run code from an untruste
  warnings.warn(
Downloading: "https://github.com/chenyaofo/pytorch-cifar-models/zipball/master" to /root/.cache/torch/hub/master.zip
Downloading: "https://github.com/chenyaofo/pytorch-cifar-models/releases/download/resnet/cifar10_resnet20-4118986f.pt" to /root
100%|██████████| 1.09M/1.09M [00:00<00:00, 130MB/s]
Accuracy of ResNet-20 on CIFAR-10 test images: 92.12%
```

شکل ۸- دانلود و تعریف مدل هدف و بررسی دقت آن روی مجموعه تست

### ۲-۲-۲. ایجاد تصاویر تخاصمی و نرخ حمله به مدل هدف

در این بخش، هدف تولید تصاویر تخاصمی با استفاده از روش **Fast Gradient Sign Method (FGSM)** و کتابخانه **CleverHans**بود. مراحل پیاده‌سازی و نتایج به شرح زیر است:

**گام‌های انجام شده:**

۱. **بارگذاری مدل هدف (Target Model)** :

- مدل از پیش آموزش داده‌شده برای دیتاست **CIFAR-10** بارگذاری شد. این مدل به حالت **eval** تغییر وضعیت داده شد تا خروجی‌ها در حالت ارزیابی تولید شوند.

۲. **تبدیل تصاویر مجموعه آزمایشی به تصاویر تخاصمی**:

- تصاویر ورودی با استفاده از تابع **fast_gradient_method** از کتابخانه CleverHans و پارامتر **epsilon=0.01** تغییر داده شدند.
- این تغییرات بر اساس گرادیان تابع هزینه مدل هدف اعمال شد.

```
epsilon = 0.01  # as requested
total = 0
num_changed = 0
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
target_model = target_model.to(device)

# Make sure the model is on GPU too
target_model.to(device)
target_model.eval()

all_adv_examples = []  # Optional: store adversarial examples for visualization
all_orig_images  = []  # Optional: store original images for visualization
all_labels       = []  # Optional: store labels
for images, labels in test_loader:
    # Move data to GPU
    images = images.to(device)
    labels = labels.to(device)



    # Now generate the adversarial samples
    images.requires_grad_()  # if needed for the gradient-based attack
    adv_images = fast_gradient_method(target_model, images, epsilon, np.inf)

    # Original predictions
    outputs = target_model(images)
    _, orig_pred = torch.max(outputs, dim=1)

    # Generate adversarial samples using FGSM with L-infinity norm
    # In CleverHans: fast_gradient_method(model, x, eps, norm)
    # for FGSM we use norm = np.inf
    adv_images = fast_gradient_method(target_model, images, epsilon, np.inf)
```

شکل ۹- کد مربوط به ایجاد نمونه‌های متخاصم

۳. **محاسبه نرخ موفقیت حمله:**

- نرخ موفقیت حمله به صورت درصدی از تصاویری که پیش‌بینی مدل روی آن‌ها تغییر کرده است، محاسبه شد.
- برای این منظور، پیش‌بینی مدل روی تصاویر اصلی و تصاویر تخاصمی مقایسه شد.

۴. **ذخیره‌سازی داده‌ها:**

- تصاویر اصلی، تصاویر تخاصمی و برچسب‌های آن‌ها به منظور تحلیل‌های بعدی ذخیره شدند.

```
    # Evaluate predictions on adversarial images
    adv_outputs = target_model(adv_images)
    _, adv_pred = torch.max(adv_outputs, dim=1)

    # Count how many images changed their label
    changed = (orig_pred != adv_pred).sum().item()
    num_changed += changed
    total       += labels.size(0)

    # (Optional) store some for visualization
    all_adv_examples.append(adv_images.cpu())
    all_orig_images.append(images.cpu())
    all_labels.append(labels.cpu())

# Attack success rate: ratio of images that changed classification
attack_success_rate = 100.0 * num_changed / total
print(f"FGSM Attack Success Rate (epsilon={epsilon}): {attack_success_rate:.2f}%")
```

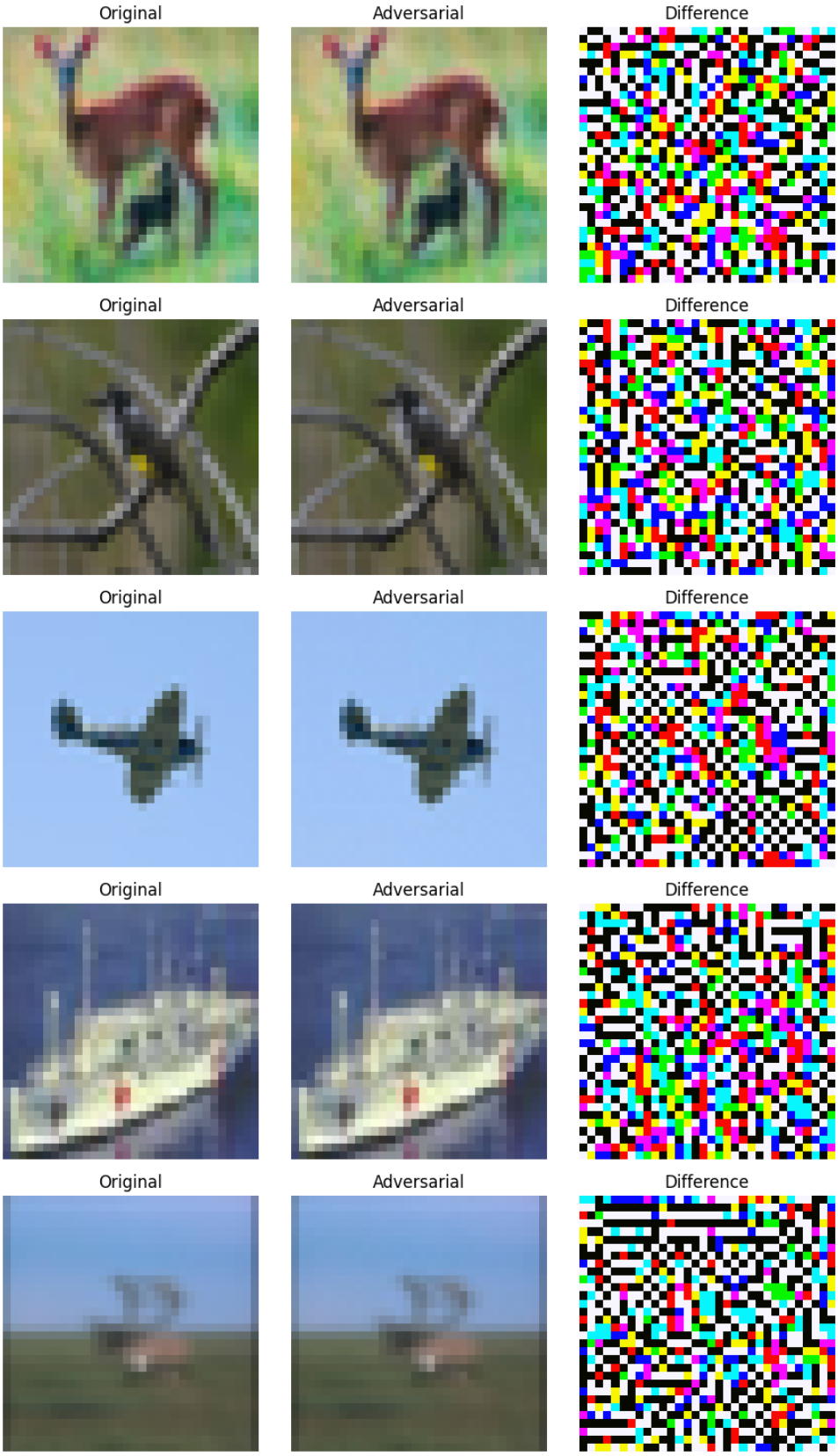
شکل ۱۰- کد مربوط به بررسی نرخ موفقیت حمله

```
FGSM Attack Success Rate (epsilon=0.01): 41.47%
```

شکل ۱۱- نرخ موفقیت حمله با روش **FGSM**

## ۵. نمایش نتایج:

- پنج تصویر به صورت تصادفی انتخاب و کنار تصاویر تخاصمی و اختلاف آن‌ها نمایش داده شدند.
- تصاویر ابتدا به حالت اولیه (بدون نرمال‌سازی) برگردانده شدند تا نمایش آن‌ها قابل فهم باشد.




شکل ۱۲- نمایش تصاویر اولیه و تصاویر تخاصمی

### ۲-۲-۳. پیاده‌سازی و آموزش مدل مولد و متمایزگر

در این بخش مدل مولد و متمایزگر مطابق معماری ذکر شده در مقاله (Pix2Pix) پیاده سازی شده است. در ادامه به تشریح مشخصات این معماری برای هر دو بخش مولد و متمایزگر و همینطور تمامی فرضیات استفاده شده در فرایند آموزش اعم از ضرایب مورد استفاده برای توابع هزینه پرداخته شده است.

**۱ .ساختار مدل‌ها و فرضیات**

۱.۱ ساختارGenerator

Generatorاستفاده‌شده در این پروژه یک معماری به‌سبک U-Net و همان مدل Pix2Pix با اندازه ورودی ۳۲×۳۲ است که شامل اجزای زیر می‌شود:

- **مسیر کاهشی(Encoder)** : لایه‌های متوالی کانولوشن با تعداد کانال‌های افزایشی ( 64 → 128 → 256 → 512) و فعال‌سازی .LeakyReLU از نرمال‌سازی Batch در تمامی لایه‌ها به‌جز لایه اول استفاده شده است.
- **مسیر افزایشی(Decoder)** : لایه‌های کانولوشن معکوس با اتصال‌های میان‌بر از مسیر کاهشی. این مسیر تعداد کانال‌ها را کاهش می‌دهد (3 ← 64 ← 128 ← 256 ← 512) و از فعال‌سازی ReLUبهره می‌برد. لایه خروجی از فعال‌سازی Tanh استفاده می‌کند تا تصاویر خروجی در محدوده [۱, ۱-] تولید شوند.

```python
class Pix2PixGenerator(nn.Module):
    """
    A simplified U-Net for 32×32 images, with fewer channels than typical
    large Pix2Pix. You can expand if you want bigger capacity.
    """
    def __init__(self, in_channels=3, out_channels=3):
        super(Pix2PixGenerator, self).__init__()

        # -- Encoder --
        # conv1: in_channels -> 64
        self.down1 = nn.Sequential(
            nn.Conv2d(in_channels, 64, kernel_size=4, stride=2, padding=1),
            nn.LeakyReLU(0.2, inplace=True)
        )
        # conv2: 64 -> 128
        self.down2 = nn.Sequential(
            nn.Conv2d(64, 128, kernel_size=4, stride=2, padding=1),
            nn.BatchNorm2d(128),
            nn.LeakyReLU(0.2, inplace=True)
        )
        # conv3: 128 -> 256
        self.down3 = nn.Sequential(
            nn.Conv2d(128, 256, kernel_size=4, stride=2, padding=1),
            nn.BatchNorm2d(256),
            nn.LeakyReLU(0.2, inplace=True)
        )

        # Bottleneck at 4×4 (for 32×32 input)
        self.down4 = nn.Sequential(
            nn.Conv2d(256, 512, kernel_size=4, stride=2, padding=1),
            nn.BatchNorm2d(512),
            nn.LeakyReLU(0.2, inplace=True)
        )

        # -- Decoder --
        # up4: 512 -> 256
        self.up4 = nn.Sequential(
            nn.ConvTranspose2d(512, 256, kernel_size=4, stride=2, padding=1),
            nn.BatchNorm2d(256),
            nn.ReLU(True)
        )
        # up3: 512 -> 128 (because we concat skip from down3 -> 256 + 256)
        self.up3 = nn.Sequential(
            nn.ConvTranspose2d(256 + 256, 128, kernel_size=4, stride=2, padding=1),
            nn.BatchNorm2d(128),
            nn.ReLU(True)
        )
        # up2: 128+128 -> 64
        self.up2 = nn.Sequential(
            nn.ConvTranspose2d(128 + 128, 64, kernel_size=4, stride=2, padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(True)
        )
        # up1: 64+64 -> out_channels
        self.up1 = nn.Sequential(
            nn.ConvTranspose2d(64 + 64, out_channels, kernel_size=4, stride=2, padding=1),
            nn.Tanh()
```

شکل ۱۳ – پیاده‌سازی معماری مدل مولد

```python
def forward(self, x):
    # Encoder
    d1 = self.down1(x)   # shape: [64, 16, 16]
    d2 = self.down2(d1)  # shape: [128, 8, 8]
    d3 = self.down3(d2)  # shape: [256, 4, 4]
    d4 = self.down4(d3)  # shape: [512, 2, 2]

    # Decoder
    u4 = self.up4(d4)    # shape: [256, 4, 4]
    # skip with d3
    u3 = self.up3(torch.cat([u4, d3], dim=1))  # shape: [128, 8, 8]
    # skip with d2
    u2 = self.up2(torch.cat([u3, d2], dim=1))  # shape: [64, 16, 16]
    # skip with d1
    u1 = self.up1(torch.cat([u2, d1], dim=1))  # shape: [3, 32, 32] => Tanh in [-1,1]

    return u1
```

شکل ۱۴ - پیاده‌سازی بخش **forward** مدل مولد

**فرضیات:**

- معماری Generator به‌منظور تطبیق با ابعاد کوچک تصاویر CIFAR-10 ساده‌سازی شده است.
- استفاده از اتصال‌های میان‌بر برای حفظ اطلاعات فضایی ضروری بوده و باعث می‌شود Generator بتواند نمونه‌های خصمانه قابل قبولی تولید کند.

۱.۲ ساختار Discriminator

Discriminator از نوع PatchGAN است که برای ارزیابی پچ‌های تصویر با اندازه ۳۲×۳۲ طراحی شده است. اجزای اصلی این مدل عبارتند از:

- لایه‌های کانولوشن متوالی ($1 \rightarrow 512 \rightarrow 256 \rightarrow 128 \rightarrow 64 \rightarrow 3$) که ابعاد فضایی را کاهش داده و عمق کانال‌ها را افزایش می‌دهند.
- فعال‌سازی LeakyReLU و نرمال‌سازی Batch در همه لایه‌ها به‌جز لایه خروجی استفاده شده است.

```python
class PatchGANDiscriminator(nn.Module):
    def __init__(self, in_channels=3):
        super(PatchGANDiscriminator, self).__init__()
        # We'll produce an output of shape [N,1,4,4] or [N,1,2,2] for 32×32 input
        # (the exact size depends on how many conv layers we add).

        model = []
        # 1) 3 -> 64
        model += [
            nn.Conv2d(in_channels, 64, kernel_size=4, stride=2, padding=1),
            nn.LeakyReLU(0.2, inplace=True)
        ]
        # 2) 64 -> 128
        model += [
            nn.Conv2d(64, 128, kernel_size=4, stride=2, padding=1),
            nn.BatchNorm2d(128),
            nn.LeakyReLU(0.2, inplace=True)
        ]
        # 3) 128 -> 256
        model += [
            nn.Conv2d(128, 256, kernel_size=4, stride=2, padding=1),
            nn.BatchNorm2d(256),
            nn.LeakyReLU(0.2, inplace=True)
        ]
        # 4) 256 -> 512
        model += [
            nn.Conv2d(256, 512, kernel_size=4, stride=1, padding=1),
            nn.BatchNorm2d(512),
            nn.LeakyReLU(0.2, inplace=True)
        ]
        # final 512 -> 1
        model += [
            nn.Conv2d(512, 1, kernel_size=4, stride=1, padding=1)
            # no activation => raw output
        ]
        self.model = nn.Sequential(*model)

    def forward(self, x):
        return self.model(x)  # shape ~ [N, 1, 2, 2] or [N,1,3,3], etc.
```

شکل ۱۵- پیاده‌سازی معماری مدل متمایزگر

**فرضیات:**

- ارزیابی مبتنی بر پچ امکان تشخیص دقیق‌تر و پایدارتر را فراهم می‌کند.
- عدم استفاده از تابع فعال‌سازی در خروجی به دلیل انجام محاسبات هزینه در مراحل بعدی است.

### ۲ .توابع هزینه

سه تابع هزینه کلیدی در این پروژه پیاده‌سازی و استفاده شده‌اند:

۱. **CW Loss** :

این تابع هزینه برای اندازه‌گیری اثربخشی نمونه‌های خصمانه در برابر مدل هدف استفاده می‌شود CW . Lossاختلاف بین logit کلاس درست و بزرگ‌ترین logit کلاس اشتباه را جریمه می‌کند.

- **فرضیات :**فرمول‌بندی CW Loss بدون هدف‌گذاری (non-targeted) کافی بوده زیرا هدف اصلی کاهش دقت مدل هدف بر روی نمونه‌های خصمانه است.

۲. **LSGAN Loss** :

این تابع هزینه برای هر دو Generator و Discriminator اعمال می‌شود Generator .تلاش می‌کند خروجی‌های خود را به‌عنوان نمونه‌های واقعی به Discriminator نشان دهد، در حالی که Discriminatorبه دنبال تشخیص صحیح نمونه‌های واقعی و جعلی است.

- **فرضیات** LSGAN **:**به دلیل پایداری و گرادیان‌های صاف در این سناریوی آموزشی مناسب است.

۳. **Hinge Loss** :

برای اعمال محدودیت بر نُرم L2 تغییرات به‌منظور اطمینان از نزدیک ماندن نمونه‌های خصمانه به تصاویر اصلی، از این تابع هزینه استفاده شده است.

- **فرضیات :**مقدار ثابت c=1 به‌عنوان آستانه‌ای مؤثر برای اندازه تغییرات بدون نیاز به تنظیم اضافی در نظر گرفته شده است.

```
##########################################
# Loss Functions
##########################################
def cw_loss(logits, target, confidence=0):
    num_classes = logits.size(1)
    correct_logit = logits[torch.arange(logits.size(0)), target]
    mask = torch.eye(num_classes, device=logits.device)[target]
    max_other_logit = torch.max(logits.masked_fill(mask.bool(), float('-inf')), dim=1)[0]
    return torch.mean(torch.clamp(max_other_logit - correct_logit + confidence, min=0))

def lsgan_loss(predictions, is_real=True):
    target_val = 1.0 if is_real else 0.0
    target = torch.full_like(predictions, target_val)
    return nn.MSELoss()(predictions, target)

def hinge_loss(perturbations, c=1):
    norms = torch.norm(perturbations.view(perturbations.size(0), -1), dim=1)
    return torch.mean(torch.relu(norms - c))
```

شکل ۱۶- تعریف توابع هزینه

همچنین ضرایب آلفا و بتا و c در فرمول نهایی ترکیب همه توابع هزینه به ترتیب برابر ۱ و ۰.۱ و ۱ در نظر گرفته شده است.

```
alpha = 1.0  # weight for LSGAN
beta = 0.1   # weight for hinge
c = 1.0      # hinge threshold
```

شکل ۱۷- ضرایب در نظر گرفته شده برای آموزش

**فرمول نهایی Generator** به شکل زیر ترکیب شده است:

$$LossG = CW + \alpha \cdot LSGAN + \beta \cdot Hinge,$$

## ۳ .تنظیمات آموزشی

### ۳.۱ داده‌ها و نرمال‌سازی

- از مجموعه داده‌های CIFAR-10 استفاده شده است که به ۴۵,۰۰۰ تصویر آموزشی، ۵,۰۰۰ تصویر اعتبارسنجی و ۱۰,۰۰۰ تصویر آزمایشی تقسیم شده‌اند.
- نرمال‌سازی استاندارد CIFAR-10 در این قسمت انجام نشده و فقط با استفاده از to_tensor تصاویر وارد بازه [0,1] شده‌اند.

خروجی Generator در بازه‌ی $[-1,1]$ است که برای اعمال به مدل هدف باید دوباره در بازه‌ی $[0,1]$ پیمانه شود. (تبدیل از $[-1,1]$به $[0,1]$)

**فرضیات:**

- نرمال‌سازی استاندارد و معکوس آن برای پایداری آموزش و حفظ سازگاری در مقیاس ورودی‌ها کافی بوده است.
- از هیچ افزونه داده (Data Augmentation) استفاده نشده و تنها به تقسیمات استاندارد-CIFAR 10اکتفا شده است.

۳.۲ بهینه‌سازی و نرخ یادگیری

- هر دو Generator و Discriminator با استفاده از بهینه‌سازی Adam با نرخ یادگیری ۰.۰۰۱ و مقادیر بتا (۰.۵, ۰.۹۹۹) آموزش دیده‌اند.
- **فرضیات :**مقادیر پیش‌فرض برای Adam بدون تنظیمات اضافی اثربخش بوده‌اند.

۳.۳ فرآیند آموزش

- در هر تکرار، Discriminatorبا استفاده از هر دو مجموعه تصاویر واقعی و نمونه‌های تولیدشده توسط Generator به‌روزرسانی شد.
- Generatorبا استفاده از ترکیبCW ، LSGANو Hinge Loss و با وزن‌دهی مناسب آموزش داده شد.
- حلقه آموزشی شامل به‌روزرسانی‌های Generator و Discriminator در هر تکرار و ارزیابی‌های دوره‌ای روی مجموعه اعتبارسنجی بود.
- آموزش به مدت ۵۰ دوره انجام شد، مطابق با جزئیات پیاده‌سازی مقاله مرجع.

## ۴ .معیارهای ارزیابی

۴.۱ دقتDiscriminator

- دقت Discriminator به‌عنوان درصد تصاویر واقعی که به‌درستی به‌عنوان واقعی طبقه‌بندی شده‌اند و تصاویر جعلی که به‌درستی به‌عنوان جعلی طبقه‌بندی شده‌اند، محاسبه شد.
- **فرضیات :**دقت بالاتر نشان‌دهنده توانایی بهتر در تفکیک بین نمونه‌های واقعی و خصمانه است، اما نباید به حدی برسد که منجر به فروپاشی مد شود.

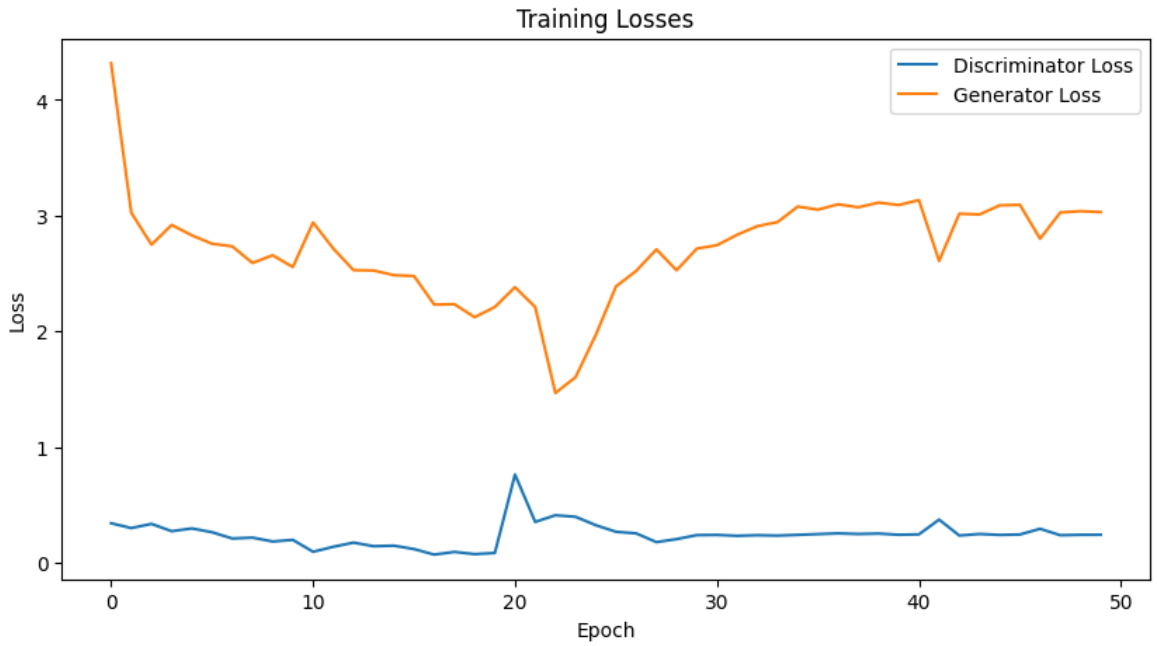
۴.۲ دقت مدل هدف

- دقت ResNet-20 بر روی نمونه‌های خصمانه در طول دوره‌ها پیگیری شد.

- **فرضیات :**کاهش دقت مدل هدف (و افزایش نرخ موفقیت حمله) نشان‌دهنده عملکرد بهتر Generatorاست.

## ۵ .نتایج و تحلیل نمودارها

### ۵.۱ نمودار Training Losses




شکل ۱۸- نمودار تغییرات **loss**

**Discriminator Loss:**

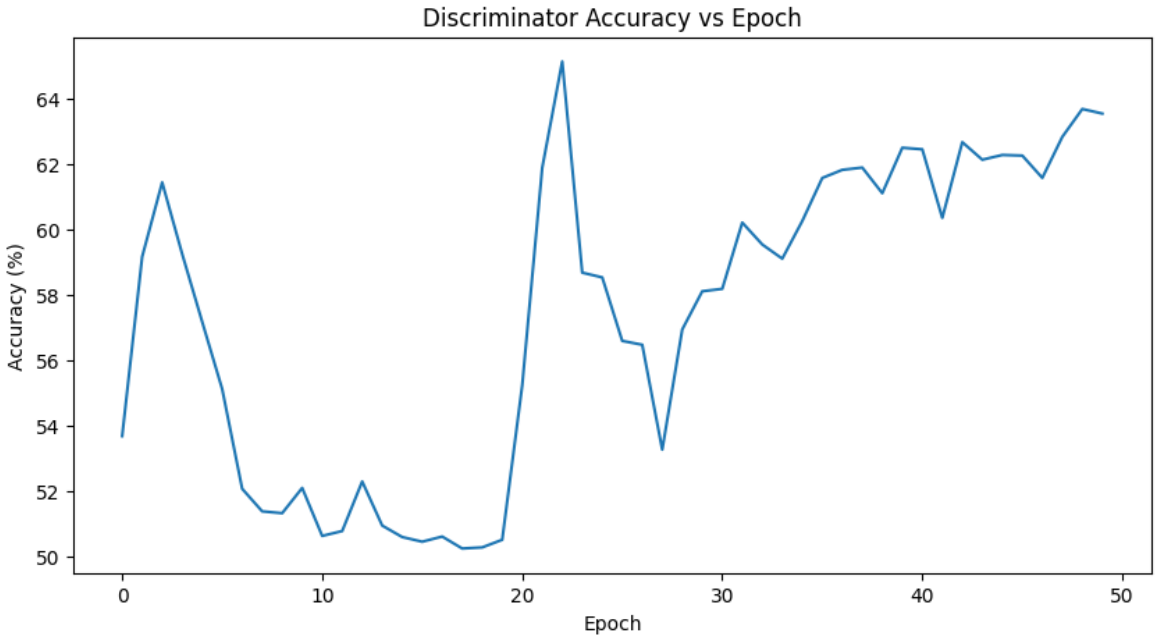
- در ابتدای آموزش مقداری نسبتاً کم (زیر ۱) دارد و با نوساناتی، گاهی نزدیک ۰.۵ و در میانه‌ی آموزش حتی تا حدود ۱.۰ بالا می‌رود.
- در ادامه به سمت مقداری زیر ۱ (بین ۰.۴ تا ۰.۶) تثبیت شده است.
- این افت‌وخیزها در اوایل، عادی است چون Generator و Discriminator هم‌زمان در حال یادگیری هستند.

- **Generator Loss:**
  - در اوایل کار بالای ۴ مشاهده می‌شود که نشان می‌دهد Generator ابتدا در فریب Discriminatorو حمله به مدل هدف دچار دشواری بوده است.

- پس از چند Epoch کاهش پیدا می‌کند (تا حوالی ۲) و سپس مجدداً روند صعودی ملایمی دارد.
- این روند صعودی می‌تواند ناشی از سنگین‌تر شدن CW Loss یا LSGAN Loss باشد، هنگامی که Generator سعی می‌کند حمله را بهبود دهد و Discriminator قوی‌تر شده است.

به طور کلی، **Discriminator**با وجود نوسان، به سطح معقولی از تمایز رسیده و **Generator**نیز توانسته نمونه‌های معناداری تولید کند؛ چرا که ضرر کلی آن تثبیت/متعادل شده و به فروپاشی منجر نشده است.

### ۵.۲ نمودار Discriminator Accuracy vs Epoch



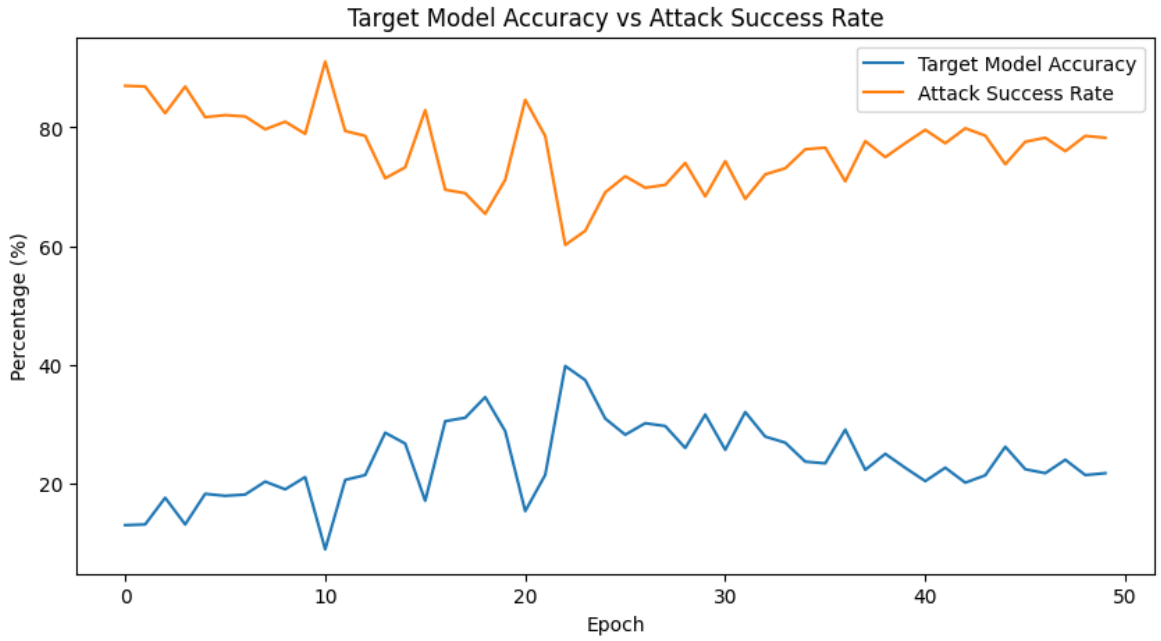

شکل ۱۹ - نمودار تغییرات دقت **Discriminator.**

دقت اولیه متمایزگر تا حدود ٪۵۵ است. سپس افتی تا زیر ٪۵۲ داشته و مجدداً رشد کرده است.

- بین بازه‌های ۲۰ تا ۲۵، افزایش چشم‌گیری (حتی بالاتر از ٪۶۴) دیده می‌شود که نشان می‌دهد در آن مقطع Discriminator در تمایز بهتر عمل کرده است.
- پس از چند نوسان، دقت حوالی ٪۶۰ تا ٪۶۴ نوسان می‌کند و در انتها کمی افت می‌کند (نزدیک ٪۶۲).

- چنین رفتاری برای یک GAN پیچیده طبیعی است، زیرا Generator و Discriminator دائماً در حال مقابله‌اند.
- دقت بیش از حد بالا نشان‌دهنده‌ی غلبه‌ی متمایزگر نیست و دقت خیلی پایین هم نشان‌دهنده‌ی ضعف شدید آن نخواهد بود؛ بنابراین حفظ عددی میان‌حد (٪۵۵ تا ٪۶۵) می‌تواند نشانه‌ی تعادل نسبی باشد.

### ۵.۳ نمودار Target Model Accuracy vs Attack Success Rate




شکل ۲۰- نمودار تغیرات **Attack Success** و **Target Model Accuracy**

- **نمودار آبی(Target Model Accuracy)** : دقت مدل هدف روی نمونه‌های خصمانه را نشان می‌دهد. ابتدا حدود ٪۲۰ شروع می‌شود، سپس نوسانی تا ٪۴۰ دارد و در طول آموزش بارها کم و زیاد می‌شود (گاهی تا ٪۱۵، گاهی نزدیک ٪۴۰). در اواخر دوره، حدود ٪۲۵ است. این یعنی مدل هدف غالباً قادر به تشخیص درست نیست) کاهش دقت چشمگیر نسبت به دقت عادی ~٪۹۳ یا ٪۹۴ روی CIFAR-10).
- **نمودار نارنجی(Attack Success Rate)** : درصد نمونه‌هایی که به طبقه‌بندی اشتباه منجر می‌شوند. این مقدار عموماً مکمل دقت مدل هدف است. در شروع نزدیک ٪۸۰ بوده، حتی در بعضی دوره‌ها به حدود ٪۹۰ و بیشتر هم رسیده و سپس نوسان کرده است (در پایان حدود ٪۷۵ تا ٪۸۰).

- به طور کلی، وجود حمله‌ی موفق نشان می‌دهد که Generator توانسته نمونه‌های مخربی بسازد که به‌خوبی دقت مدل هدف را پایین بیاورد.
- نوسانات بخشی ناشی از تغییرات مستمر در Discriminator است که روی یادگیری نهایی Generatorتأثیر می‌گذارد.

**۶ .فرضیات و محدودیت‌ها**

- **مدل هدف از پیش آموزش‌یافته :**فرض بر این بود که ResNet-20 به خوبی کالیبره شده و نماینده‌ای مناسب برای دسته‌بندی‌کننده‌های استاندارد CIFAR-10 است.
- **تعداد دوره‌ها :**انتخاب تعداد ۵۰ دوره بر اساس پیاده‌سازی مرجع صورت گرفت و بهینه‌سازی تجربی نشد.
- **وزن توابع هزینه :**وزن‌های ترکیبی برایCW ، LSGANو Hinge Loss به‌طور دستی انتخاب شدند و تنظیم بیشتری انجام نشد.
- **سخت‌افزار و زمان اجرا :**آموزش بر روی یک GPU انجام شد و محدودیت‌های زمانی ممکن است بر عمق کاوش در تنظیمات اثر گذاشته باشد.

### ۲-۲-۴. بررسی نرخ موفقیت حمله و هیستوگرام قطعیت

**مفروضات و نکات کلیدی در این مرحله:**

- **تعریف «حمله موفق** :حمله درصورتی موفق است که برچسب پیش‌بینی‌شده‌ی مدل هدف برای تصویر خصمانه با برچسب واقعی آن تفاوت داشته باشد(label=/predadv)
- **مدل هدف** :یک شبکه‌ی **ResNet-20** از پیش آموزش‌دیده بر روی CIFAR-10 که در حالت evalقرار دارد.
- **مشخصه‌ی بازه‌ی پیکسل‌ها** :تصاویر ورودی اصلی در بازه‌ی [0,1]هستند (پس از ()ToTensor بدون نرمال‌سازی خاص) Generator اما خروجی را در بازه‌ی [-1,1]تولید می‌کند و برای اعمال به مدل هدف، آن را دوباره به [0,1]نگاشت می‌کنیم.
- **تعریف** :**Confidence** حداکثر احتمال Softmax مدل (بیش‌ترین مؤلفه‌ی بردار خروجی شبکه).
- **تقسیم داده‌ی تست** :شامل ۱۰هزار تصویر درCIFAR-10 ، با ۱۰۰۰ نمونه در هر یک از ۱۰ کلاس

## ۱ .نرخ موفقیت حمله (کلی و به ازای هر کلاس)

### ۱.۱ .نرخ موفقیت کلی

با آزمون روی کل ۱۰هزار نمونه‌ی آزمایشی، حمله در **۷۷.۱۰٪** موارد موفق عمل کرده است. این بدان معناست که نزدیک به سه‌چهارم نمونه‌ها پس از اعمال نویزهای خصمانه، توسط مدل هدف اشتباه طبقه‌بندی شده‌اند.

### ۲.۱ .نرخ موفقیت به ازای هر کلاس

بر اساس نتایج به‌دست‌آمده از کد، برای هر کلاس CIFAR-10 (به‌ترتیب برچسب‌های ۰ تا ۹)، میزان موفقیت حمله به‌صورت زیر است:

جدول **1** - نرخ موفقیت حمله به ازای هر کلاس

| کلاس | نام کلاس (CIFAR-10) | تعداد نمونه در تست | تعداد شکست مدل | نرخ موفقیت حمله (%) |
|---|---|---|---|---|
| 0 | airplane | 1000 | 678 | 67.80% |
| 1 | automobile | 1000 | 997 | 99.70% |
| 2 | bird | 1000 | 917 | 91.70% |
| 3 | cat | 1000 | 640 | 64.00% |
| 4 | deer | 1000 | 849 | 84.90% |
| 5 | dog | 1000 | 322 | 32.20% |
| 6 | frog | 1000 | 832 | 83.20% |
| 7 | horse | 1000 | 921 | 92.10% |
| 8 | ship | 1000 | 563 | 56.30% |
| 9 | truck | 1000 | 991 | 99.10% |

**مشاهدات مهم:**

- بیش‌ترین موفقیت در حمله به کلاس‌های **automobile**(نزدیک ٪۹۹.۷) و **truck**(٪۹۹.۱) دیده می‌شود.
- کم‌ترین موفقیت مربوط به کلاس **dog**است (تنها ٪۳۲.۲).
- این نتایج نشان می‌دهد که برخی کلاس‌ها (مانند وسایل نقلیه) نسبت به نویزهای خصمانه‌ی تولیدیGenerator ، حساسیت بیشتری دارند و مدل هدف را آسان‌تر به خطا می‌اندازند؛ در مقابل، در برخی کلاس‌ها (نظیر سگ) تغییرات اعمالی کمتر موفق به تغییر پیش‌بینی مدل شده است.

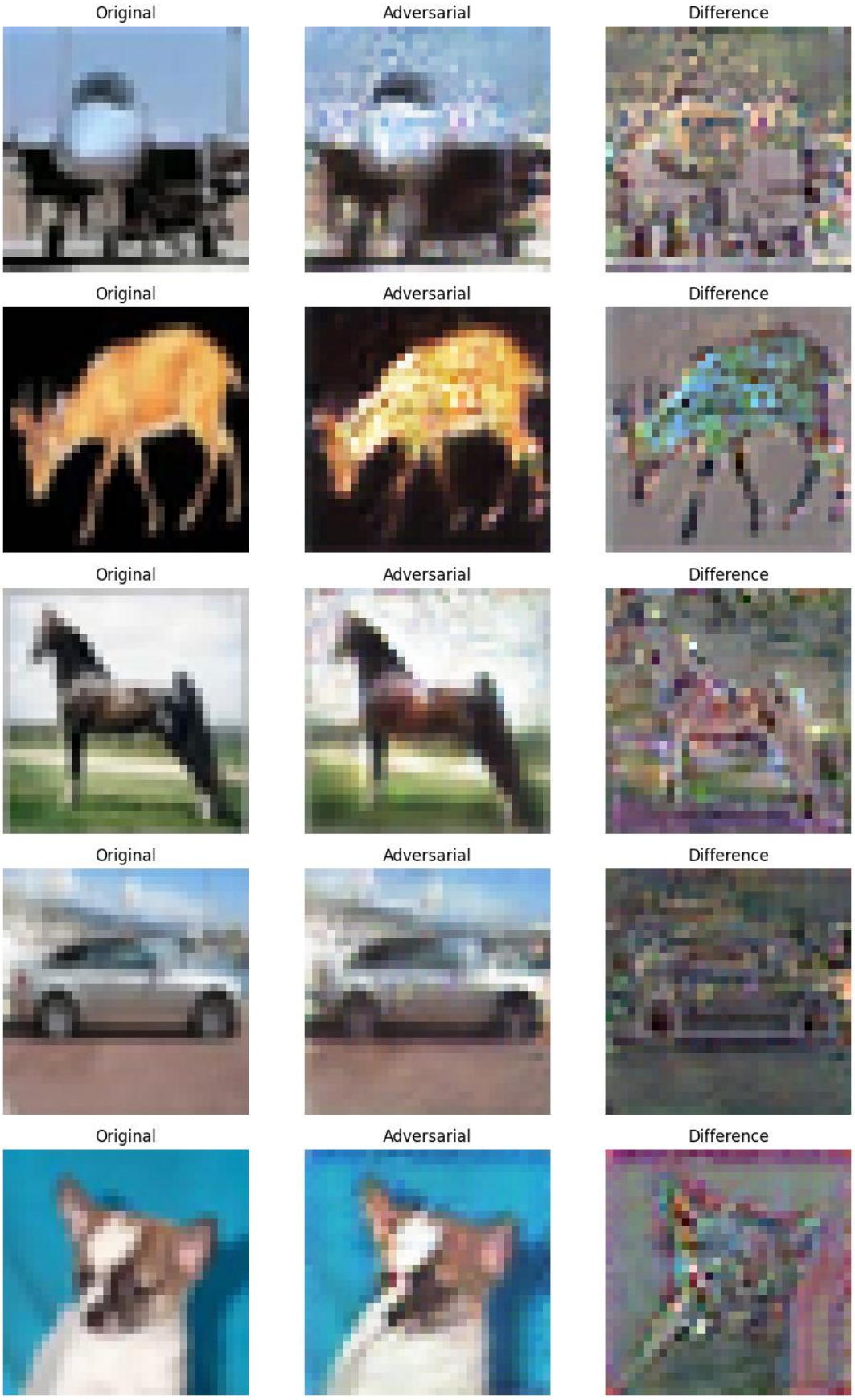
## ۲ .نمایش ۵ تصویر از مجموعه‌ی آزمایشی و نسخه‌ی خصمانه

در شکل زیر، پنج نمونه از مجموعه‌ی آزمایشی نمایش داده شده است. برای هر مورد، سه ستون مجزا وجود دارد:

۱. **Original**: تصویر اصلی در بازه‌ی [0,1]

۲. **Adversarial**: تصویر خروجی Generator که پس از تولید در بازه‌ی [-1,1]، به [0,1]نگاشت شده است.

۳. **Difference**: نقشه‌ی اختلاف Δ=(Adversarial−Original) که سپس برای دیده‌شدن بهتر به بازه‌ی [0,1]نرمال می‌شود.



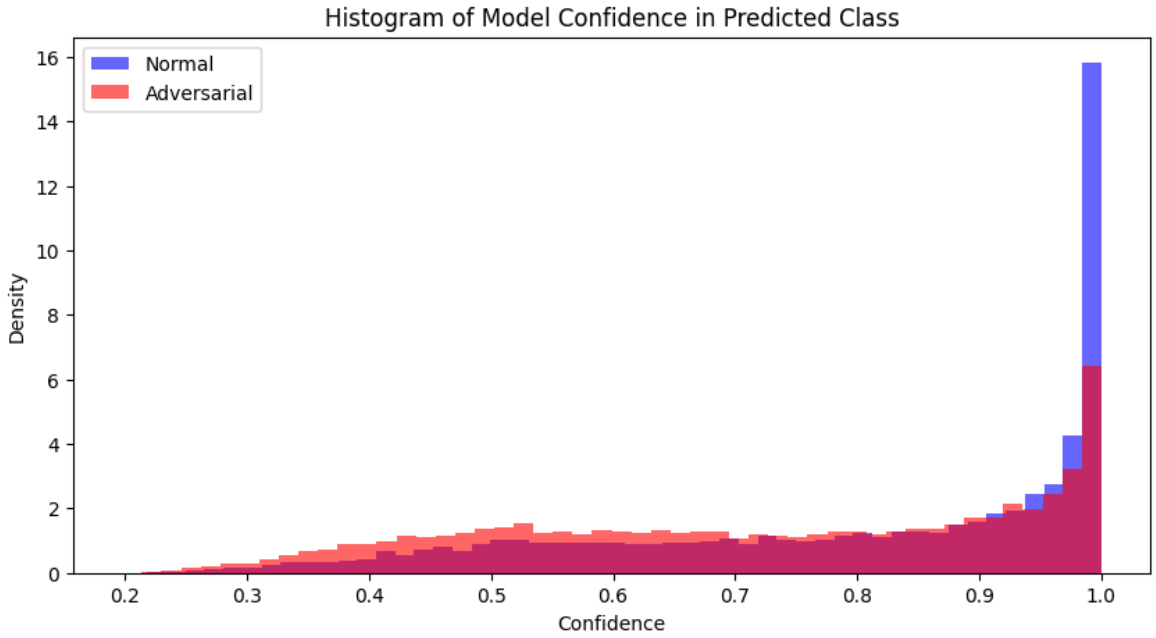

شکل ۲۱- نمونه‌هایی از تصاویر واقعی و خصمانه به همراه تفاوت آنها

**تحلیل کیفی:**

- تفاوت ظاهری بین تصویر اصلی و خصمانه غالباً در حد نویزهای محلی مشاهده می‌شود.
- نقشه‌ی اختلاف، نواحی رنگی را نشان می‌دهد که در آن‌ها نویز بیش‌تر/کم‌تری اعمال شده است.
- از دید انسان، تصویر خصمانه هنوز شباهت بسیار بالایی به نمونه‌ی اصلی دارد؛ اما برای مدل هدف به اندازه‌ی کافی تغییر ایجاد می‌کند تا طبقه‌بندی را مختل نماید.

### ۳ .رسم نمودار هیستوگرام قطعیت مدل هدف در پیش‌بینی

در این مرحله، برای هر دو مجموعه‌ی تصاویر عادی و تصاویر خصمانه **Confidence**مدل هدف در کلاس پیش‌بینی‌شده محاسبه و سپس نمودار هیستوگرام (با تعداد سطل‌های ۵۰) رسم شده است:




شکل ۲۲- هیستوگرام قطعیت

**محور افقی** :مقدار اعتماد در بازه‌ی [0,1]

- **محور عمودی** :چگالی (Density) نمونه‌ها در هر بازه از اعتماد

**نتایج کلیدی:**

۱. برای تصاویر عادی، بخش قابل‌توجهی از نمونه‌ها اعتماد بالایی (نزدیک ۱.۰) دارند. این طبیعی است زیرا مدل هدف بر روی CIFAR-10 آموزش دیده و معمولاً با اطمینان بالا تصمیم می‌گیرد.

۲. برای تصاویر خصمانه، توزیع اعتماد پخش‌تر است و اوج (پیک) آن در محدوده‌ی پایین‌تر از ۱.۰ قرار دارد (بیش‌تر در محدوده‌ی ۰.۴ تا ۰.۷). این نشان می‌دهد که حملات خصمانه موفق شده‌اند بسیاری از نمونه‌ها را از اعتماد حداکثری دور کنند و مدل را به پیش‌بینی‌هایی کم‌اعتمادتر (یا اشتباه) سوق دهند.

**۴ .جمع‌بندی**

۱. **نرخ موفقیت کلی** حمله در این بخش به **۷۷.۱۰٪** رسیده است؛ عددی قابل‌توجه که بیان‌گر تأثیر موفق این حمله روی ResNet-20 از پیش آموزش‌دیده است.

۲. به‌صورت کلاس‌محور، حمله در برخی کلاس‌ها (نظیر خودرو یا کامیون) تقریباً به‌طور کامل موفق بوده و در برخی دیگر (مانند سگ) موفقیت کم‌تری داشته است.

۳. بررسی **۵ نمونه‌ی بصری** نشان داد که نویز اعمالی اغلب برای چشم انسان کوچک است، اما در عین حال **مدل هدف** را به‌طور قابل‌توجهی دچار اشتباه می‌کند.

۴. هیستوگرام قطعیت مدل هدف هم نشان داد که **اعتماد شبکه** برای نمونه‌های خصمانه به‌طور میانگین کمتر شده و از ۱.۰ فاصله گرفته است؛ امری که تأیید می‌کند سازوکار حمله (Generator) به خوبی توانسته است به اطمینان بالای مدل هدف آسیب بزند.

با توجه به این نتایج، می‌توان گفت ترکیب معماری Pix2Pix و PatchGAN با توابع هزینه‌ی CW+LSGAN+Hingeدر تولید نمونه‌های خصمانه برای CIFAR-10 کارآمد بوده و ضمن حفظ شباهت تصویری، درصد موفقیت بالایی در شکستن طبقه‌بندی مدل هدف به دست آورده است.

### ۲-۲-۵. پیاده‌سازی و بررسی مدل هدفدار (امتیازی)

۱. **حملات بدون هدف :(Non-Targeted Attacks)** هدف این نوع حملات باعث می‌شود مدل ورودی را به هر کلاس اشتباهی دسته‌بندی کند.

۲. **حملات هدفدار :(Targeted Attacks)** در این حملات، هدف مشخص این است که مدل ورودی را به یک کلاس خاص دیگر اشتباه دسته‌بندی کند.

در این بخش، برخلاف حمله‌ی غیرهدفدار که صرفاً تلاش می‌کرد مدل هدف در طبقه‌بندی نمونه دچار خطا شود، از یک حمله‌ی هدفدار (Targeted Attack) استفاده شده است. در حمله‌ی هدفدار، هر نمونه

طوری دستکاری می‌شود که برچسب پیش‌بینی‌شده‌ی مدل هدف به یک کلاس موردنظر، اینجا (label+1) برسد.

تفاوت اصلی پیاده‌سازی با بخش قبل در موارد زیر است:

- **انتخاب برچسب هدف(Target Label) :** در بخش غیرهدفدار، تابع هزینه‌ی CW سعی داشت logitکلاس درست را کمتر از ماکس سایر کلاس‌ها کند؛ اما اینجا با تابع کوچکی مثل get_target_labels، برچسب جدیدی برای هر نمونه تعریف می‌شود تا Generator به‌طور مشخص آن کلاس را هدف قرار دهد.
- **CW Lossهدفدار :**محاسبه‌ی تابع هزینه‌ی CW اکنون بر اساس **برچسب هدف** صورت می‌گیرد (در کد( cw_loss(logits_target, target_labels, ...)یعنی Generator تلاش می‌کند logitکلاس هدف را بالاتر از بقیه‌ی کلاس‌ها ببرد.
- **تعریف متریک حمله :**برای ارزیابی، به‌جای کاهش دقت مدل در پیش‌بینی لیبل اصلی، اکنون میزان موفقیت در پیش‌بینی کلاس هدف سنجیده می‌شود. در کد، اگر دقت مدل هدف در رسیدن به این کلاس بالا باشد، حمله هدفدار بهتر عمل کرده است.

سایر بخش‌های معماریGenerator ، Discriminator، و نیز توابع LSGAN و Hinge Loss مانند قبل هستند و تغییر خاصی ندارند. با همین تغییر کوچک در برچسب هدفدار و تابع هزینه‌ی مرتبط، رویکرد غیـرهدفدار قبلی به یک حمله‌ی هدفدار تبدیل می‌شود.

در نهایت نتایج خواسته شده به شکل زیر بدست می‌آید.

**۱ .نتایج کلی حمله‌ی هدفدار**

- **نرخ موفقیت کلی حمله:**

نرخ موفقیت حمله در این بخش برابر **۸۵.۴۵٪** است که نشان‌دهنده موفقیت بسیار بالا در حمله به مدل هدف است. این مقدار به‌طور قابل‌توجهی بالاتر از نرخ موفقیت حمله در بخش غیرهدفدار است (۷۷.۱۰٪). این تفاوت به این دلیل است که حمله‌ی هدفدار به‌طور خاص تلاش می‌کند که مدل هدف را به سمت یک کلاس خاص هدایت کند که نتیجه‌ی آن موفقیت بالاتری است.

- **نرخ موفقیت به ازای هر کلاس:**

نرخ موفقیت حمله در کلاس‌های مختلف در جدول زیر آمده است:

جدول 2 - نرخ موفقیت حمله به ازای هر کلاس مدل **targeted**

| کلاس | نام کلاس | نرخ موفقیت حمله(%) | تعداد موفقیت‌ها (از ۱۰۰۰ نمونه) |
|---|---|---|---|
| 0 | airplane | 75.60% | 756 |
| 1 | automobile | 100.00% | 1000 |
| 2 | bird | 98.70% | 987 |
| 3 | cat | 70.50% | 705 |
| 4 | deer | 92.20% | 922 |
| 5 | dog | 39.10% | 391 |
| 6 | frog | 96.70% | 967 |
| 7 | horse | 99.60% | 996 |
| 8 | ship | 82.10% | 821 |
| 9 | truck | 100.00% | 1000 |

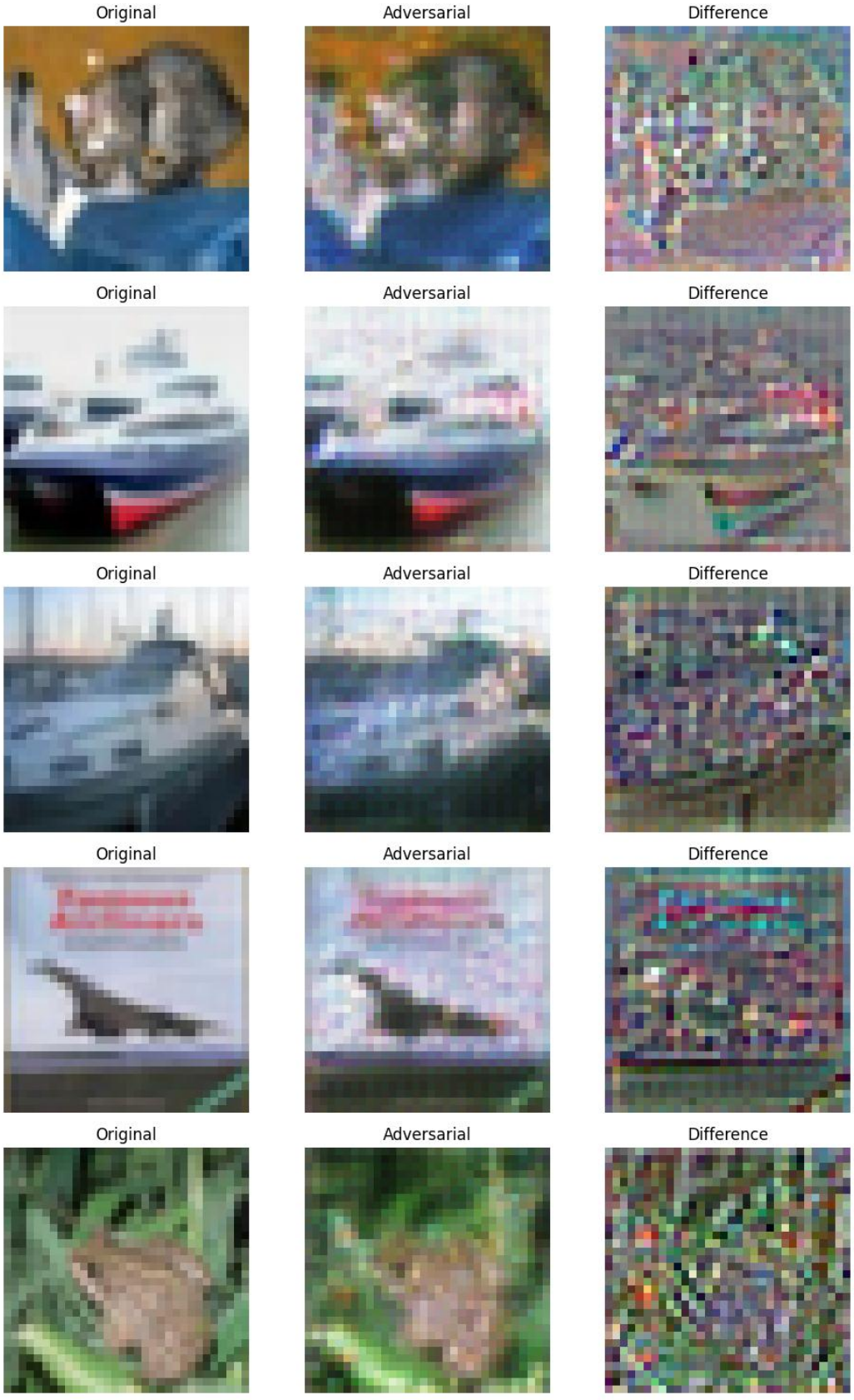
**تفاوت‌ها با حمله‌ی غیرهدفدار:**

- در حمله‌ی هدفدار، کلاس‌هایی مانند **automobile** و **truck** به‌طور کامل تحت تأثیر قرار گرفته و نرخ موفقیت حمله به ۱۰۰٪ رسید. این کلاس‌ها به‌طور خاص دارای ویژگی‌های بصری قابل‌تشخیص بیشتری هستند که به Generator این امکان را می‌دهد که تصاویر خصمانه‌ای ایجاد کند که به‌راحتی مدل هدف را فریب دهند.
- در کلاس‌هایی مانند **dog**، نرخ موفقیت به‌شدت کاهش پیدا کرده و تنها ۳۹.۱۰٪ موفقیت در تغییر پیش‌بینی مدل هدف مشاهده می‌شود. این امر ممکن است به دلیل ویژگی‌های پیچیده‌تر و کمتر قابل‌تشخیص در سگ‌ها باشد که برای تغییر دقیق‌تر نیاز به اصلاحات بیشتری دارد.
- در مجموع، نرخ موفقیت در کلاس‌هایی که بیشتر ویژگی‌های یکپارچه دارند (مثل **automobile**, **truck**, **frog**) به‌شدت بیشتر است.

**۲ .نمایش ۵ نمونه تصاویر و تفاوت‌ها**

در شکل زیر، پنج نمونه از مجموعه‌ی آزمایشی همراه با تصاویر خصمانه تولیدشده توسط Generator نمایش داده شده است. در این شکل سه ستون وجود دارد:

- **Original**: تصویر اصلی که به مدل داده می‌شود.
- **Adversarial**: تصویر خصمانه‌ای که توسط Generator ساخته شده است.
- **Difference**: نقشه‌ی اختلاف بین تصویر اصلی و تصویر خصمانه.



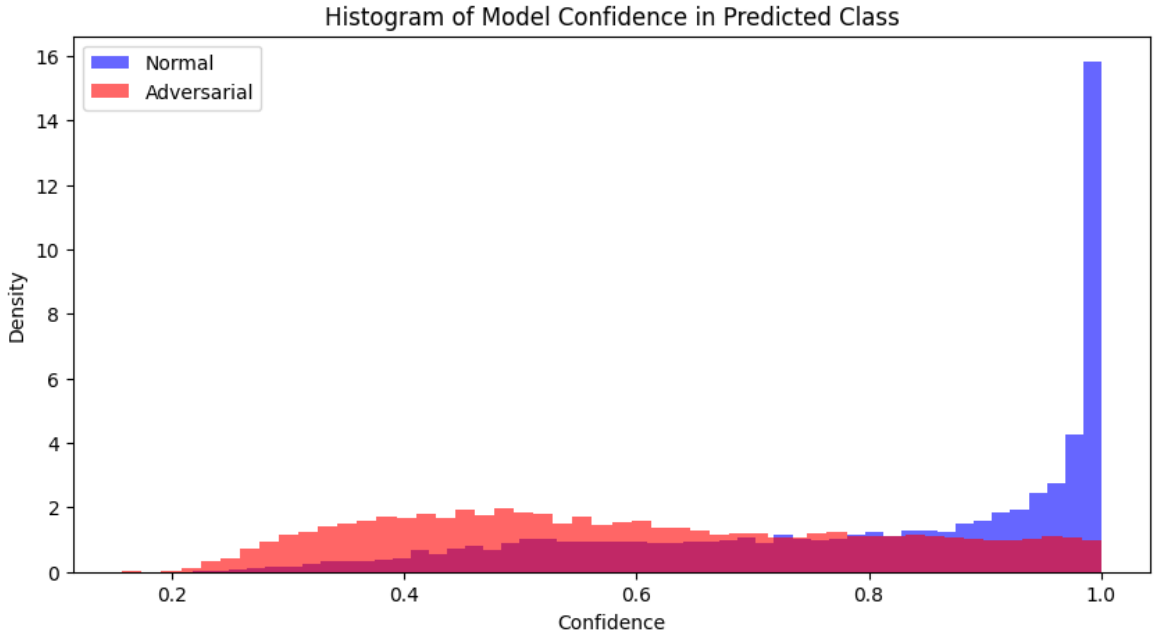

شکل ۲۳- نمونه‌هایی از تصاویر واقعی و خصمانه به همراه تفاوت آنها در مدل **targeted**

**تفاوت‌های مهم در تصاویر هدفدار:**

- در تمامی تصاویر خصمانه، نویز واردشده به‌گونه‌ای است که به‌طور معمول برای چشم انسان کم‌وبیش قابل‌مشاهده نیست.
- تفاوت‌ها در بخش سوم، یعنی نقشه‌ی اختلاف(Difference) ، به‌وضوح مشخص است. این نقشه‌ها به ما نشان می‌دهند که دقیقاً کدام قسمت از تصویر تغییرات بیشتری داشته‌اند و چطور این تغییرات بر نتایج پیش‌بینی تأثیر گذاشته است.

### ۳ .نمودار هیستوگرام قطعیت مدل هدف در طبقه‌بندی

نمودار زیر، توزیع **Confidence**(اعتماد مدل هدف در پیش‌بینی برچسب) را برای **نمونه‌های عادی** (آبی) و **نمونه‌های خصمانه** (قرمز) روی مجموعه‌ی آزمایشی نشان می‌دهد:




شکل ۲۴- هیستوگرام قطعیت مدل **targeted**

**تفاوت با حمله‌ی غیرهدفدار:**

- در **نمونه‌های عادی** (آبی)، اکثر پیش‌بینی‌ها با اعتماد بالا (نزدیک به ۱) انجام می‌شوند.
- در **نمونه‌های خصمانه** (قرمز)، بیشتر نمونه‌ها دارای اعتماد کم‌تر هستند. این به این معنی است که مدل هدف در پیش‌بینی برچسب موردنظر با اطمینان کمتری عمل می‌کند. همچنین مشاهده

می‌شود که تعداد زیادی از نمونه‌ها در بازه‌ی ۰.۴ تا ۰.۷ تمرکز دارند که نشان‌دهنده‌ی افت اعتماد مدل هدف در مواجهه با نمونه‌های خصمانه است.

### ۴ .جمع‌بندی

در مقایسه با حمله‌ی غیرهدفدار که صرفاً به هدف ایجاد اشتباه در مدل هدف تمرکز داشت، حمله‌ی هدفدار قادر است مدل هدف را به سمت کلاس خاصی هدایت کند. نتایج نشان‌دهنده موفقیت بالای حمله‌ی هدفدار است:

- **نرخ موفقیت کلی حمله** در حمله‌ی هدفدار **۸۵.۴۵٪** است که به‌شکل قابل‌توجهی بالاتر از **۷۷.۱۰٪** در حمله‌ی غیرهدفدار قرار دارد.
- **دقت بالا در کلاس‌های هدف‌دار** مانند **automobile**و **truck**، به این معنی است که Generatorتوانسته به‌طور مؤثر این کلاس‌ها را هدف قرار دهد.
- **نمودار هیستوگرام اعتماد** در حمله‌ی هدفدار نشان‌دهنده کاهش اعتماد مدل هدف به‌خصوص در نمونه‌های خصمانه است که بر اساس افت اعتماد و افزایش نویز در پیش‌بینی‌ها قابل‌مشاهده است.

نتایج کلی این گزارش تأیید می‌کند که حمله‌ی هدفدار به‌طور مؤثری توانسته بر عملکرد مدل هدف تأثیر گذاشته و با هدایت مدل به سمت کلاس‌های خاص، نرخ موفقیت بالایی کسب کرده است، اما از طرفی تصاویر تولید شده تفاوت بیشتری با واقعیت دارند و اندکی دور تر از واقعیت شده اند. شاید با انتخاب متفاوت پارامترهای بخش loss می‌توانستیم به تصاویری با تفاوت کمتر با واقعیت برسیم.