به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

Trustworthy AI

تمرین شماره 1

نام و نام خانوادگی	محمد جواد رنجبر
شماره دانشجویی	۸۱۰۱۰۱۱۷۳
تاریخ ارسال گزارش	۰۱/۱۸

فهرست گزارش سوالات

فهرست اشکال

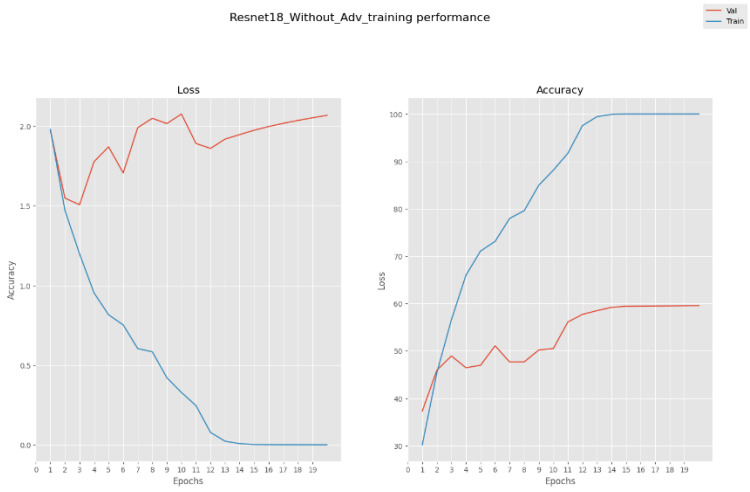
سوال ۱ – Generalization and Robustness

بخش ۱

ابتدا دیتاست Cifar10 را لود می‌کنیم و داده‌های train را با نسبت ۰.۸ به دو دسته train و validation تقسیم می‌کنیم. همینطور نورمالزیشن مناسب برای داده‌های Cifar10 را بر روی این داده‌ها اعمال می‌کنیم.

بخش ۲

ابتدا مدل resnet18 را از کتابخانه Pytorch، بدون استفاده از وزن‌های آموزش داده بارگذاری می‌کنیم. و سپس به تعداد ۲۰ دوره روی داده‌های استاندارد آموزش می‌دهیم. که نمودار عملکرد مدل در شکل ۱ قابل مشاهده است.



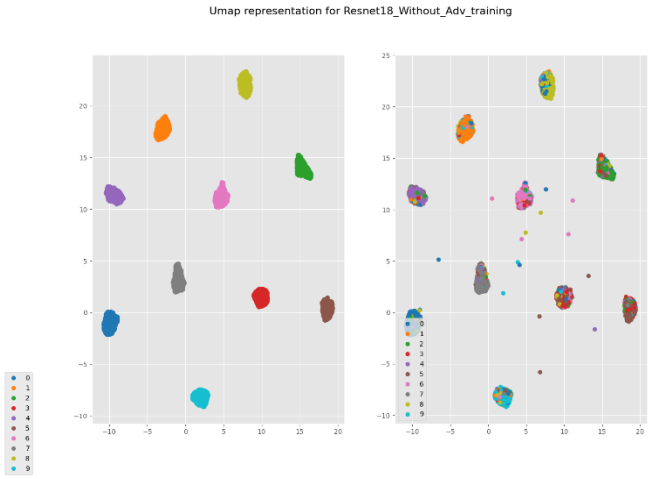
شکل ۱ عملکرد مدل استاندارد

نتیجه این شبکه روی داده‌های تست‌های به صورت زیر می‌باشد:

```
Accuracy of the Resnet18_Without_Adv_training on the 10000 test
images: 59 %
Accuracy of the Resnet18_Without_Adv_training on the 10000 train
images: 100 %
```

قابل ذکر است که شبکه‌ی ما overfit شده است، ولی با این حال دقت نزدیک به ۶۰ روی داده‌های تست داریم.

حال نقشه ویژگی استخراج شده را به umap می‌دهیم و شکل زیر را نمایش می‌دهیم:



شکل ۲ بازنمایی **umap** برای داده‌های استاندارد

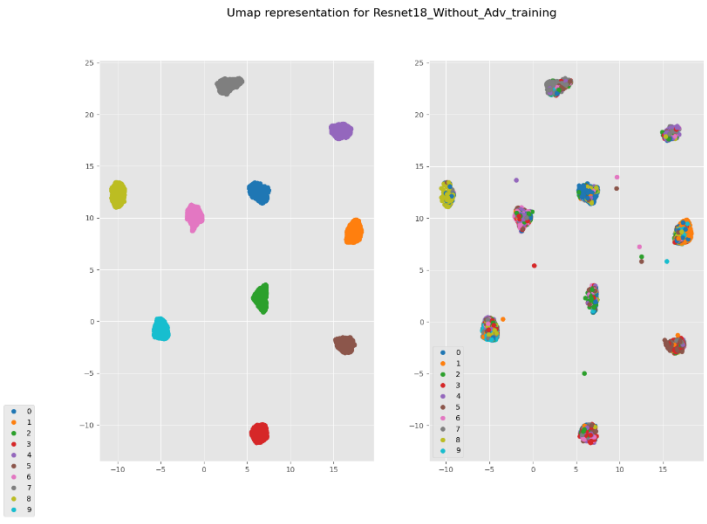
کلاس‌ها برای داده‌های آموزش استاندارد به صورت کامل جدایی‌پذیر هستند(البته overfit صورت گرفته است.) ولی برای داده‌های تست استاندارد تعدادی miss classification صورت گرفته است.

بخش ۳

حال اغتشاش‌های نویز گوسی، چرخش، flip و color jitter را به تصویر اعمال می‌کنیم، مشخص است که عملکرد مدل به شدت روی داده‌های تست بدتر شده است.

```
Accuracy of the Resnet18_Without_Adv_training on the 10000 test images: 33 %
Accuracy of the Resnet18_Without_Adv_training on the 10000 train images: 100 %
```

حال نقشه ویژگی استخراج شده را به umap می‌دهیم و شکل زیر را نمایش می‌دهیم:



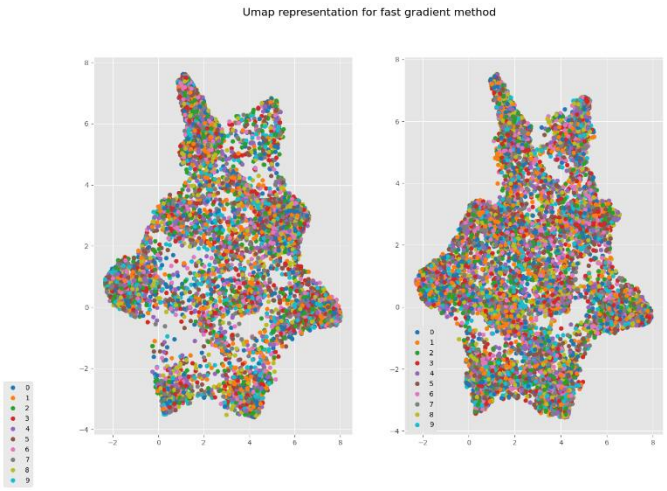
شکل ۳ بازنمایی **umap** برای مدل استاندارد روی داده نویزی

مشخص است که داده‌های آموزش تغییری نکرده‌اند. اما مرز جدایی ساز داده‌های تست به واسطه اضافه کردن نویز بهم ریخته است.

حال fast gradient method را اعمال می‌کنیم، نتیجه این حمله بر روی داده‌های اموزش و تست به صورت زیر می‌باشد:

```
Accuracy of the network on the 10000 test images: 15 %
Accuracy of the network on the 40000 train images: 0 %
```

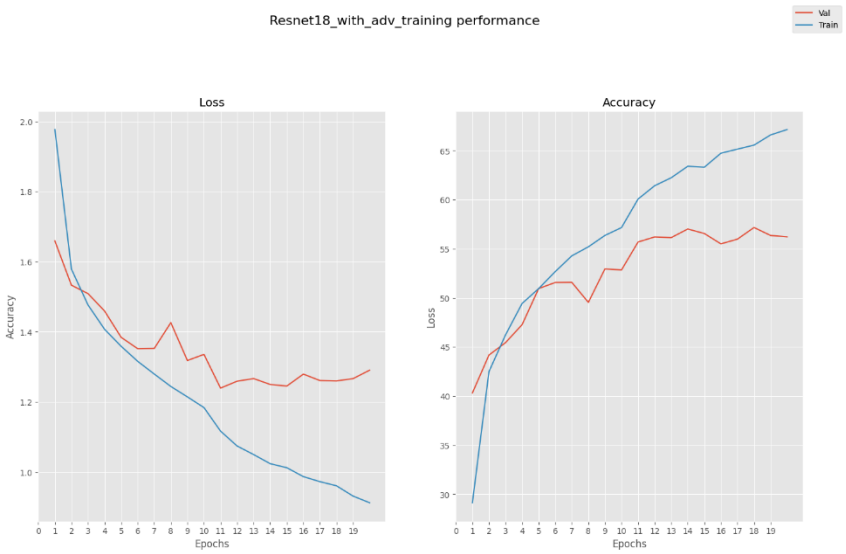
از انجا که این حمله با توجه به loss صورت می‌گیرد، به شدت دقت طبقه‌بند را پایین اورده و همچنین مرز جداساز داده‌ها غیر قابل تشخیص می‌شود.



شکل ۴ بازنمایی **umap** برای **FGSM**

بخش ۴

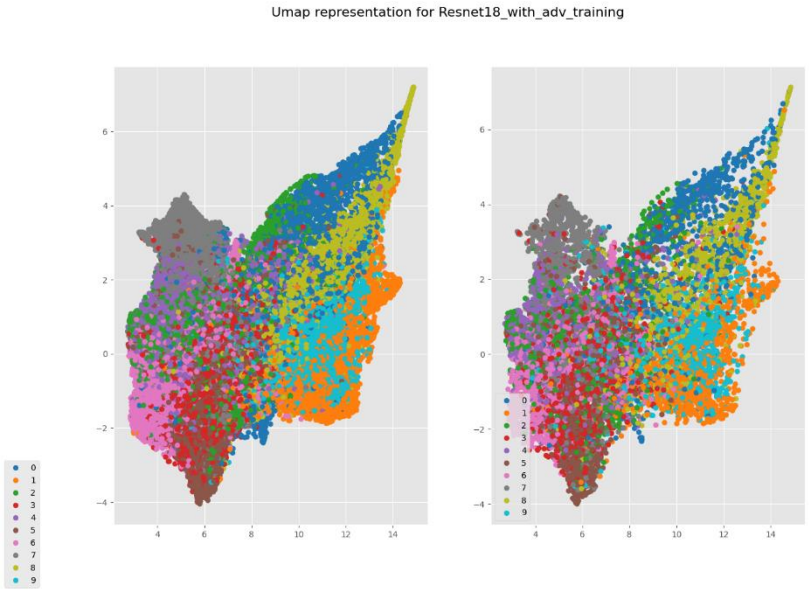
حال هم داده‌های آموزش و هم داده‌های ارزیابی را دارای تغییرهایی شامل نویز گوسی، چرخش، flip و color jitter می‌کنیم و مدل را آموزش می‌دهیم:



شکل ۵ عملکرد مدل با **adversarial training**

عملکرد مدل هم روی داده‌های آموزش و هم روی داده‌های تست adversarial نسبت به عملکرد مدل روی داده‌ی معمولی بدتر شده است.

```
Accuracy of the Resnet18_with_adv_training on the 10000 test images:
55 %
Accuracy of the Resnet18_with_adv_training on the 40000 train images:
67 %
```

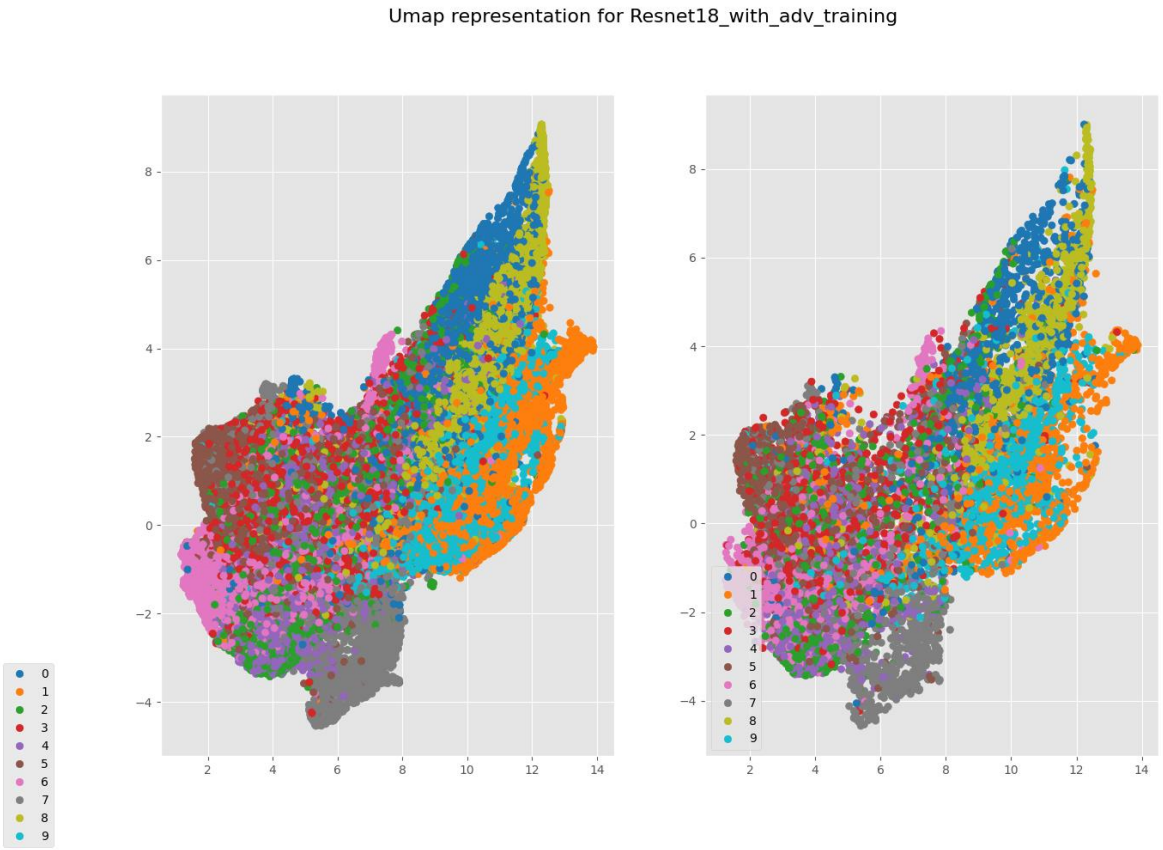


شکل ۶ بازنمایی **umap** برای **adversarial training**

مشخص است که مرزجدایی ساز داده‌ها پیچیده‌تر شده است ولی همچنان قابل جداسازی هستند.

همچنین عملکرد مدل بر روی داده‌های استاندارد و بدون تغییر به شکل زیر است:

```
Accuracy of the Resnet18_with_adv_training on the 40000 test images:
61 %
Accuracy of the Resnet18_with_adv_training on the 10000 train images:
50 %
```

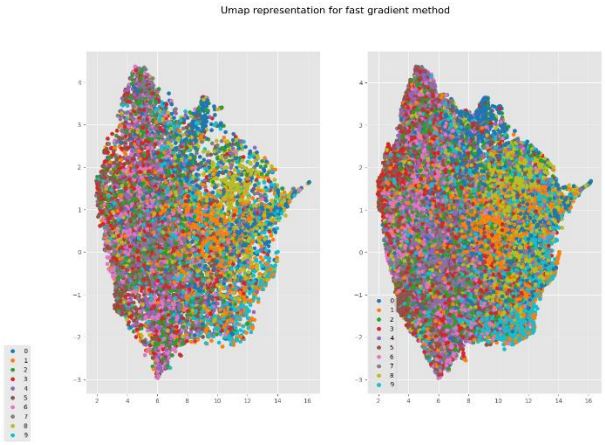


شکل7 بازنمایی **umap** مدل با **adversarial training** بر روی داده‌های ساده

عملکرد مدل روی این داده‌ها نیز بدتر شده است اما نسبت به مدل استاندارد بهتر است. از آنجا که مدل adv ویژگی‌های robust تر را یاد گرفته است نه ویژگی‌هایی که با کلاس همبستگی بالایی دارند، انتظار می‌رفت که عملکرد مدل از لحاظ دقت بدتر شود.

حال FGSM را امتحان می‌کنیم:

```
Accuracy of the network on the 10000 test images: 15 %
Accuracy of the network on the 40000 train images: 12 %
```



شکل ۸ بازنمایی **Umap** برای حمله **FGSM** بر روی مدل **adversarial**

این مدل در برابر حمله FSGM مقاوم‌تر و بهتر از مدل استاندارد عمل می‌کند که احتمالا به خاطر آموزش روی داده‌های نویزی بوده است.

بخش ۵

ابتدا تابع triplet loss را توضیح می‌دهیم:

در این تابع هر نمونه آموزشی، شامل سه داده می‌باشد. که شامل داده‌ی لنگر[۱] و داده‌ی مثبت و داده‌ی منفی هست که داده‌ی مثبت و لنگر متعلق به کلاس یکسان هستند و داده‌ی منفی متعلق به کلاس دیگری است. این تابع سعی می‌کند که تفاوت داده‌های متعلق به کلاس یکسان را بهم دیگر نزدیک کرده و داده‌های کلاس مختلف را از هم دور کند. به عبارت دیگر اگر داده‌ی x_a برای لنگر در نظر بگیریم و داده‌ی x_p و x_n به ترتیب داده‌های مثبت و منفی هستند، این تابع loss سعی می‌کند معادله زیر را برقرار کند:

$$\|x_a - x_P\|^2 - \|x_a - x_n\|^2 \leq 0$$

یک جواب معادله بالا این است که همه‌ی xها برابر با صفر باشند که نادرست است لذا یک margine به فرمول بالا اضافه می‌کنیم و به صورت زیر خواهد بود:

$$\|x_a - x_P\|^2 - \|x_a - x_n\|^2 + m \leq 0$$

و تابع $loss$ را به این صورت می‌توانیم بنویسیم:

$$loss = max(\|x_a - x_P\|^2 - \|x_a - x_n\|^2 + m, 0)$$

[۱] Anchor

حال برای حل مشکلاتی مانند مناسب نبودن مقدار m یکسان بین همه‌ی کلاس‌ها و اینکه گرادیان هر داده فقط برای آن داده و داده‌ی دوم در نظر گرفته می‌شود.

لذا از متریک زاویه‌ای برای این کار استفاده می‌کنیم که به صورت زیر عمل می‌کند:

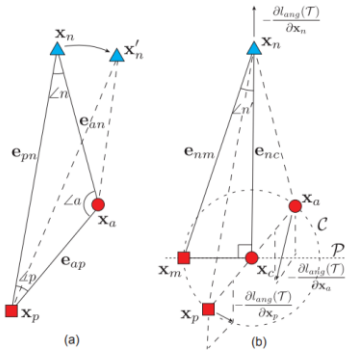
سه داده x_a و x_p و x_n را در فضای دو بعدی در نظر بگیرید. و مثلثی که دارای اضلاع $e_{an} = x_a - x_n$ و $e_{ap} = x_a - x_p$ و $e_{pn} = x_p - x_n$ هستند به وجود می‌آید. تابع خطای *tripel loss* ضلع بلندتر یعنی e_{an} را بیشتر نسبت به ضلع e_{ap} جریمه می‌کند. با توجه به خاصیت نمونه‌ها شرط $e_{ap} + m \leq e_{pn}$ با توجه به قانون کوسینوس‌ها اثبات می‌شود که زاویه n که توسط دو ضلع e_{ap} و e_{an} احاطه شده است، کوچکترین زاویه‌ی این مثلث است و یعنی $\angle n \leq \min(\angle a, \angle p)$. و از آنجا که $\angle n + \angle a + \angle p = 180^\circ$ پس می‌توانیم یک حد بالا برای $\angle n$ تعریف کنیم که این زاویه باید کمتر از ۶۰ درجه باشد. و شرط $\angle n \leq \angle \alpha$ برقرار باشد.

می‌توان این رابطه را به صورت زیر نیز نوشت:

$$\angle n \leq \frac{\|x_a - x_P\|}{2 * \|x_n - x_c\|} \leq \angle \alpha$$

و تابع *loss* به صورت زیر تعریف می‌شود:

$$loss = max([\|x_a - x_P\|^2 - 4 * tan^2 \alpha \|x_n - x_c\|^2, 0)$$

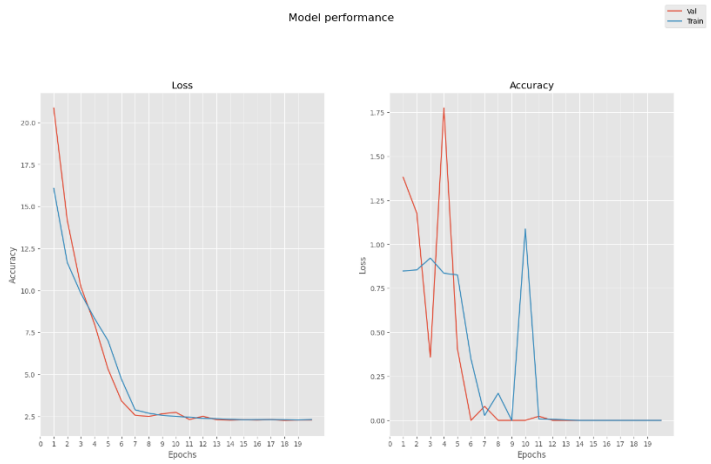


که این تابع نسبت به تابع *triple loss* برتری‌های زیر را دارد:

- محاسبه $\angle n$ شامل هرسه ضلع می‌شود.
- زاویه $\angle \alpha$ برای مقیاس‌های مختلف کار می‌کند.
- تخمین مارجین مناسب در تابع اصلی پیچیده بود ولی در این ساده‌تر است.

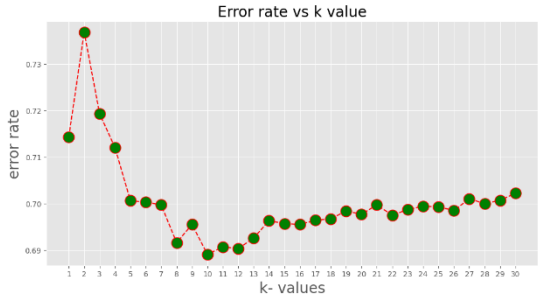
بخش ۶

حال از آنجا که metric learning داریم انجام می‌دهیم. باید لایه اخر شبکه به جای ۱۰ کلاس ۱۲۸ نورون بگذاریم و شباهت بردارهای خروجی را با مدل‌هایی مانند knn بسنجیم. همینطور اطمینان حاصل کنیم در هر batch حداقل یک داده‌ی از همه‌ی کلاس‌ها وجود دارد برای همین از batch sampler استفاده می‌کنیم. لذا مدل شبکه عصبی با خواص گفته شده آموزش دهیم. مشخص است که این شبکه به تنهایی نباید عملکرد خوبی داشته باشد از انجا که قصد یادگیری برای طبقه‌بندی نداریم و عملکرد مدل به شکل زیر است:



شکل ۹ عملکرد مدل اموزش دیده با **angular loss**

حال feature vector مورد نظر را باید روی مدل Knn آموزش دهیم. برای انتخاب K مناسب به این صورت عمل می‌کنیم که تعدادی مختلف مدل آموزش داده و k در نقطه زانویی را انتخاب می‌کنیم که در اینجا برابر با ۱۰ می‌باشد:



شکل ۱۰ پیدا کردن **k** مناسب

حال یک مدل knn با k=10 اموزش می‌دهیم و عملکرد این مدل روی داده‌های تست و ترین استاندارد به صورت زیر می‌باشد:

عملکرد مدل روی داده‌های آموزش:

	precision	recall	f1-score	support
0	0.62	0.39	0.48	6373
1	0.37	0.56	0.45	2647
2	0.48	0.35	0.40	5525
3	0.28	0.47	0.35	2364
4	0.53	0.36	0.43	5881
5	0.28	0.57	0.38	1993
6	0.54	0.40	0.46	5320
7	0.32	0.56	0.41	2269
8	0.57	0.44	0.49	5283
9	0.33	0.56	0.41	2345
accuracy			0.43	40000
macro avg	0.43	0.47	0.43	40000
weighted avg	0.48	0.43	0.44	40000

عملکرد مدل روی داده‌های تست:

	precision	recall	f1-score	support
0	0.49	0.29	0.36	1689
1	0.26	0.42	0.32	623
2	0.35	0.24	0.29	1446
3	0.14	0.27	0.18	509
4	0.41	0.28	0.33	1471
5	0.17	0.42	0.24	415
6	0.43	0.30	0.35	1438
7	0.18	0.38	0.24	482
8	0.46	0.33	0.38	1387
9	0.22	0.41	0.29	540
accuracy			0.31	10000
macro avg	0.31	0.33	0.30	10000
weighted avg	0.37	0.31	0.32	10000

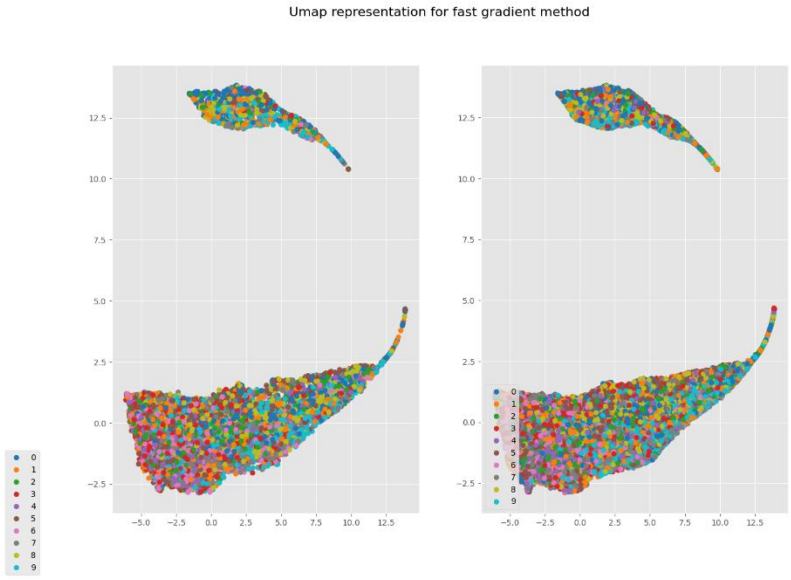
عملکرد مدل نسبت به حالت ساده و استاندارد بدتر شده است.

حال عملکرد مدل را بر روی داده‌های adv امتحان می‌کنیم که برای داده‌ی تست به صورت زیر خواهد بود:

	precision	recall	f1-score	support
0	0.50	0.27	0.35	1826
1	0.21	0.31	0.25	675
2	0.23	0.20	0.21	1172
3	0.14	0.21	0.16	651
4	0.27	0.22	0.24	1227
5	0.18	0.35	0.24	515

6	0.42	0.27	0.33	1558
7	0.20	0.34	0.25	575
8	0.38	0.30	0.33	1248
9	0.21	0.38	0.27	553
accuracy			0.27	10000
macro avg	0.27	0.29	0.26	10000
weighted avg	0.32	0.27	0.28	10000

بر خلاف مدل استاندارد، عملکرد مدل بر روی داده‌های نویزی تغییر چندانی نداشت و نسبت به مدل استاندارد robust تر بود. در این بخش از آنجا که ما در حال متریک لرنینگ هستیم و در لایه آخر لیستی از ویژگی‌ها استخراج می‌کنیم، بازنمایی umap معنی خاصی نخواهد داشت. با این حال شکل آن به صورت زیر می‌باشد.



شکل ۱۱ بازنمایی **umap** برای **knn** با داده **adv**

حال همین مدل را بر روی داده‌ای که توسط FGSM نویزی شده است امتحان می‌کنیم که نتیجه‌ی آن به صورت زیر می‌باشد:

برای داده‌های آموزش:

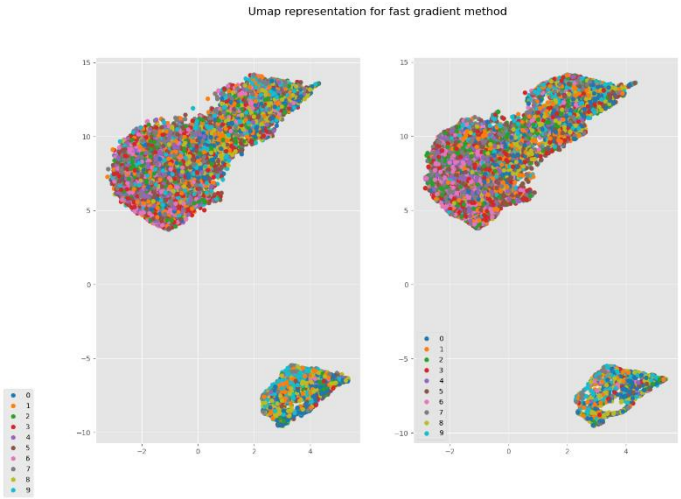
	precision	recall	f1-score	support
0	0.45	0.36	0.40	1284
1	0.19	0.41	0.26	473
2	0.40	0.28	0.33	1422
3	0.24	0.35	0.28	688
4	0.54	0.26	0.35	2034
5	0.23	0.46	0.31	483
6	0.50	0.33	0.40	1519
7	0.32	0.42	0.37	749
8	0.35	0.40	0.38	892
9	0.19	0.43	0.27	456

accuracy			0.34	10000
macro avg	0.34	0.37	0.33	10000
weighted avg	0.40	0.34	0.35	10000

برای داده‌های تست:

	precision	recall	f1-score	support
0	0.40	0.30	0.34	1322
1	0.15	0.26	0.19	557
2	0.22	0.16	0.19	1332
3	0.16	0.20	0.18	799
4	0.30	0.18	0.23	1631
5	0.17	0.27	0.21	633
6	0.36	0.22	0.27	1664
7	0.22	0.24	0.23	906
8	0.27	0.36	0.31	761
9	0.15	0.38	0.21	395
accuracy			0.24	10000
macro avg	0.24	0.26	0.24	10000
weighted avg	0.27	0.24	0.24	10000

مشخص است که مدل مورد نظر به شدت نسبت به مدل ساده بهتر عمل می‌کند. دلیل این هم مشخص است که به خاطر متریک لنرنیگ است که یعنی ویژگی‌هایی از مدل یاد می‌گیریم و داده‌هایی که این ویژگی‌ها شبیه به هم هستند را در یک دسته می‌گذاریم. و این مدل نسبت به مدل استاندارد به شدت robust تر عمل می‌کند. همینطور بازنمایی umap آن به شکل زیر خواهد بود:



شکل ۱۲ بازنمایی **umap** برای **FGSM** و **Knn**